The application of machine learning methods in flight delay problems

Adam Szostek

Monika Jung

Salma Ennassar

Nassima EL Garn

Abstract

In our project we tried to perform machine learning tasks on a dataset consisting of 7 million records of domestic USA flights. We wanted to classify whether flights will be late/delayed **on arrival**, regress the number of minutes of that delay and finally try to find groups of correlated delayed flights using clustering. For the classification we used only features known before the plane's departure and managed to achieve very impressive results considering the complexity and significance of the problem, prioritising maximization of recall on delayed flights. We designed a regression experiment to quantify the linear propagation of delays, confirming the strong correlation between departure and arrival times and also classifying flight delay regression using only features known beforehand as a problem too complex even for advanced models. Finally, for Clustering, we applied unsupervised learning algorithms to the subset of delayed flights. We were able to segment the delayed flights, helping us understand the underlying behaviors and similarities of flights that arrive late.

1. Introduction

Nowadays, aviation has possibly become humanity's most efficient and time-saving way of traveling. In commercial flights, one of the main issues we still fight with are delays. Currently, the scheduled timetable always accounts for possible delays by creating pessimistic arrival times, therefore giving the pilots possibility to speed up and still arrive as planned. However, in 2023 in Europe on average commercial flights were late on arrival by approximately 15 minutes [1]. That means that delays are still a huge problem and can be very annoying to passengers especially on connecting flights. Coming from this conclusion we decided to try to tackle the problem of flight delays using machine learning.

We have specified three separate tasks we want to solve, each with different specifications. Using classification we want to try to predict beforehand, whether a flight would be delayed on arrival. An important specification here is to define what late truly means as it’s a subjective description. For the purpose of our task we defined late for flights on arrival as **any amount larger than zero**. In this scenario an obvious next choice is to decide what to prioritize while training our models. We came to the conclusion that classifying a flight as delayed and it eventually not being delayed is potentially less harmful than classifying it as not delayed and ending up delayed. Therefore, in our approach we prioritise recall more than precision to help our classification models predict the positive labels. For regression it was clear that regressing the number of minutes that a flight will be late on arrival using mostly categorical features is close to impossible, specifically for linear models. That is why we changed our approach to try to find correlation between departure delay and arrival delay, while also including additional categorical features for our models. That, as expected, yielded great results as the two values remain closely correlated. And finally in clustering we wanted to try to find clusters of **late** flights in order to help find patterns which those specific flights might have in common. That may prove to be very useful, as it can help develop methods or solutions to prevent these groups of flights from being late.

2. Data Description

The Kaggle Flight Delay Dataset 2024 [2], which was set up from the Bureau of Transportation Statistics (BTS) On-Time Performance Data for every domestic flight in the United States for the year 2024, is the source of our dataset. Approximately 7 million flight records and 35 characteristics are included, giving extensive information on both scheduled and actual flight operations description.

Structures and Key Features

The dataset includes different types of variables such as categorical, numerical, and time-based, which provide complete information about the flight operations. The main variables are:

- month - month of a flight
- day_of_month - day of a flight
- day_of_week - day of the week
- op_unique_carrier - unique carrier’s code
- origin - Origin airport of a flight
- origin_city_name - origin city name of a flight
- origin_state_name - origin state name of a flight
- dest - destination airport
- dest_city_name - destination city name of a flight
- dest_state_name - destination state name of a flight
- dep_time - scheduled departure time of a flight
- distance - distance of a flight

The comprehensive characteristics of this feature set allow for the investigation of various delay-related phenomena, such as the effectiveness of airlines, the efficiency of routes, and the trends of temporal delays.

Pre-processing and Data Handling:

In order to be able to work with our chosen dataset and to ensure data quality, we performed a series of preprocessing steps:

1. Handling Missing Values: This resulted in a dataset reduction from approximately 7.08 million to 6.97 million records, which is acceptable given the large quantity of data, therefore we simply dropped the rows.
2. Duplicate Removal: A check for duplicate entries confirmed that the dataset contained no duplicate records.
3. Time-based columns (crs_dep_time, dep_time, arr_time, crs_arr_time) were converted from HHMM strings to time objects in order to facilitate analysis.
4. To define the classification targets, two new variables were added:
    a. is_dep_delayed: flights with a dep_delay greater than 15 minutes were labeled as delayed
    b. is_arr_delayed: flights with an arr_delay greater than 0 minutes were labeled as delayed.
5. We also noted that our data is unevenly distributed between labels; approximately 36% of flights are delayed on arrival and 74% are not. That is very important for further steps we have taken when choosing and fine-tuning the models.
6. Based on exploratory analysis, a subset of features (mentioned above) was selected for modeling.
7. We analyzed the relationship between arrival and departure delays. Despite the presence of extreme delays (e.g. >300 minutes), these data points were preserved rather than removed as outliers. Keeping them ensures the model remains robust to significant delay events.

For data analysis and preprocessing we used the following libraries: pandas [9], numpy [8], matplotlib [7], seaborn [13].

3. Models

For all the models mentioned below we use implementation from the scikit-learn library [3], besides XGBoost models which come from the xgboost library [10].

3.1. Classification

Model 1: Support Vector Machine

For this task, one of the models we chose to try is SVM in order to explore non-linear decision decision boundaries using the ,,Kernel Trick”. SVM models have demonstrated their efficacy in high-dimensional areas and their robustness in managing outliers. We tested multiple different kernels (Linear, RBF, Sigmoid and Polynomial) to determine the relationship between features.

The SVM algorithm finds the best hyperplane that separates the two classes (Delayed vs. On-Time) with the maximum margin.

Model 2: Naive Bayes

Naive Bayes is a probabilistic classifier that is built on Bayes' theorem, nonetheless, it assumes that the predictors are independent. The model here predicts the probability of a class (in other words, whether the flight is delayed or on-time) based on the feature values that are recorded. We used the Gaussian Naive Bayes method, as most of our features are continuous (like e.g., departure delay, distance). The approach is to first compute the likelihood for each of the classes and then select the class that has the highest posterior probability.

Model 3: Random Forest

The Random Forest machine learning model is a collection of several decision trees that allows for a more stable and accurate outcome than an individual tree. Each decision tree is "trained" on a subset of data and features selected at random, thereby helping to identify non-linear patterns and also to avoid overfitting. Random Forests can be applied to datasets with high dimension and high noise content and produce features that allow for an explanation of how important each feature is, allowing for easier interpretation of the final results.

Model 4: XGBoost

XGBoost (extreme gradient boosting) is a machine learning algorithm which is known for its high effectiveness in machine learning tasks. It utilizes the sequential error fixing principle and very often outperforms deep learning models such as multi layer perceptrons.

XGBoost starts by building a simple first model (usually a decision tree) which tries to predict the given machine learning task. This model is not perfect and makes mistakes, but the mistakes are the actual key to the brilliance of this approach.

In the second step XGBoost analyzes the mistakes made by the first model and builds a second one whose sole purpose is to predict and correct those errors. Then the algorithm connects the predictions of the two models and this aggregated team is already better at predicting than each of them alone.

The steps are repeated many times and each new model tries to correct the mistakes of the aggregated team. The name “extreme” comes from its high efficiency and effectiveness.

XGBoost builds trees in parallel, utilizing our machine's resources and integrates regularization which prevents overfitting. It can also deal with missing and bad quality data on its own.

3.2. Regression

Model 1: XGB Regressor

We selected the XGBoost regressor because it is widely regarded as the state-of-the-art algorithm for tabular data. XGBoost is an ensemble method capable of discovering complex, non-linear relationships between flight features.

XGBoost works by successively creating an ensemble of decision trees. Each new tree corrects the errors (residuals) made by the previous trees by minimizing an objective function.

The model was implemented in a standard CPU-based Google Colab environment [4] using the xgboost library alongside pandas and sklearn. We used a preprocessed, shuffled dataset of 100,000 rows to manage computational resources.

The preprocessing involved handling time-based features. We converted columns like month, day_of_week, and dep_time into sine and cosine pairs. This allows the model to understand the cyclical nature of time

Model 2: Linear Regression, Ridge and Lasso

A linear regression model is the simplest method to perform that machine learning task. It uses an OLS (ordinary least squares) estimator, which fits a straight line onto the data, while trying to minimise the sum of squared errors. The errors are calculating the difference between actual target values and fitted values. It’s highly prone to outliers and noise in the dataset.

Ridge regression is very similar, but it modifies the loss function by adding to the sum of squared errors the coefficient vector squared and multiplied by a chosen value. Thanks to that it can keep the weights close to zero.

Lasso regression functions almost identically to the Ridge regression except it only uses the norm of the coefficient vector for regularization. In comparison to Ridge it can shrink a weight to zero, choosing only relevant features.

Model 3: Theil-sen Regression

This regression method relies on the median value of all the slopes between all the pairs of points in the training data. Thanks to choosing the median rather than the average it can be very robust to outliers and not get influenced by them. However this comes at a very high computational cost making it highly inefficient for medium/large datasets. For Theil-sen we used a subset of 50 thousand samples of our data.

Model 4: RANSAC Regression

Ransac regression uses random sampling from the data to fit a regression line and then tests that line against the rest of the points. Thanks to that it can calculate how many points fall within a certain threshold and are inliers and how many don't which makes them outliers. It then chooses the regression line with the highest number of outliers. It can be very robust to noise in the dataset thanks to its algorithm.

Model 5: Huber Regression

Huber regressor uses the same algorithm that Linear regression uses with a small modification to the loss function. For small errors (within a certain threshold) it uses the squared loss, but for larger errors it switches to the absolute error making it harder to be influenced by outliers. The approach may be more trivial than other linear regression methods, but it's very fast computationally and pretty robust to noise.

Model 6: Polynomial Regression

While linear regression assumes a straight-line relationship between the dependent and independent variables, polynomial regression models the relationship as an n-degree polynomial. We implemented this to test if introducing non-linearity into a linear model could capture delay patterns better than the standard OLS estimator.

3.3. Clustering

Model 1: K-means clustering

Our objective was to segment flights that had experienced a delay into natural groups. Since our dataset contains numerical features as well as categorical features, we selected the K-Prototypes algorithm instead of K-Means, because it handles mixed data types natively.

K-Prototypes integrates the K-Means and K-Modes algorithms. It minimizes a cost function that combines the Euclidean distance for numerical features and a matching dissimilarity measure for categorical features.

Model 2: BIRCH

The BIRCH algorithm was chosen for clustering of flight delays, as well as to help identify patterns associated with various types of delays. Because of its scalable nature, it is well suited to use with large-scale data (multi-million record size).
In preparation for clustering, all numerical variables (e.g., distance, departure delay) were standardized and PCA (principal components analysis) was used to reduce dimensionality and stabilise the clustering process. In addition, BIRCH builds a CF tree incrementally, which provides a summary of the denser regions of the data to create natural clusters, which requires the least amount of memory and computational effort.

Model 3: Gaussian Mixture

A GMM is a probabilistic model that assumes all data points are generated from a mixture of a finite number of Gaussian distributions. In comparison to for eg. KMeans, which assigns a point to a single cluster, it uses soft clustering, therefore giving each point a probability that it belongs to a certain cluster.

The scikit-learn implementation of Gaussian Mixture Models functions via the Expectation-Maximization algorithm, an iterative optimization technique. The process alternates between an Expectation step, which calculates the posterior probabilities of assignment for each data point, and a Maximization step that updates the component weights, means, and covariance matrices based on these derived responsibilities, continuing until convergence falls within a specified tolerance. To address issues with the algorithm's initial convergence, the implementation first runs a pre-clustering algorithm (eg. KMeans) , while also permitting multiple independent restarts to ensure a global maximum is approximated. The model's geometric flexibility is further constrained by the covariance type hyperparameter, which dictates the degrees of freedom for the resulting distributions.

Model 4: DBSCAN

The DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) identifies clusters based upon density by determining the regions where a grouping of data points exists due to density and classifying data points in low-density regions as noise. The algorithm requires setting two parameters: ε (epsilon), which sets the neighborhood radius around a given data point, and min_samples, which is the minimum number of neighboring points that make a given region dense. Core points are defined by the criteria for the ε and min_samples parameters. In turn, clusters are formed by linking together core points that are accessible via density. In addition, a point is marked as a border point if it is accessible through a core point but does not constitute a core point due to lack of density. All other points fall into the “noise” category. As a result, DBSCAN provides a mechanism for identifying clusters of multiple shapes and also for detecting outliers within datasets that may contain dense cluster formations along with other irregularly spaced or unevenly distributed patterns of data.

4. Methodology

4.1. Classification

We approached the classification task using four algorithms: SVM, Naive Bayes, Random Forest, and XGBoost. To prepare the data, for the first three algorithms we transformed categorical features using LabelEncoder. However, for the XGBoost model, we applied cyclic encoding to time-based variables to allow the model to better capture their periodic behavior.

For the Naive Bayes classifier, we used Gaussian Naive Bayes because we modelled most of our numerical features (e.g., Distance) were based on Continuous distributions. All of our categorical variables were converted using LabelEncoder so that they could be used with the Naive Bayes Classifier. RandomizedSearchCV was used to tune the Smoothing parameter and provide robust performance. StratifiedKFold was employed to maintain the class distribution across the folds while performing RandomizedSearchCV. Naive Bayes also served as a lightweight benchmark for comparing against the more complex models, allowing for a quick and easy understanding of how the Naive Bayes model performed compared to other models.

For hyperparameter optimization, we employed RandomizedSearchCV combined with StratifiedKFold for the Naive Bayes, Random Forest, and XGBoost models with f1-score as optimization criterion. This cross-validation strategy was important to preserve the correct percentage of samples for each class, therefore addressing the dataset's uneven distribution.

The SVM model required a slightly different workflow due to its high computational cost. We trained it on a subset of 50,000 random records using the RAPIDS suite on a GPU to accelerate the process. Additionally, the data was scaled using StandardScaler, and we applied SelectKBest (k=4) to retain only the month, day_of_week, origin_city_name, and dep_time. Unlike the other models, SVM was tuned using Optuna with class_weight='balanced' to prioritize the minority class, explicitly targeting Recall and F1-score optimization.

The categorical variables in our Random Forest Model were encoded using LabelEncoder. To achieve a balanced distribution of classes throughout our validation process, we used RandomizedSearchCV and StratifiedKFold when doing our hyperparameter tuning. Due to the size of our dataset, a smaller representative subset was used during hyperparameter tuning, and once we found the best parameters, we retrained the best model on our entire set of training data. During hyperparameter tuning, we tested the number of trees, maximum depth, splitting rules, and feature sampling; each of these parameters contributed to Random Forest's ability to efficiently model nonlinear relationships while also having sufficient robustness against overfitting.

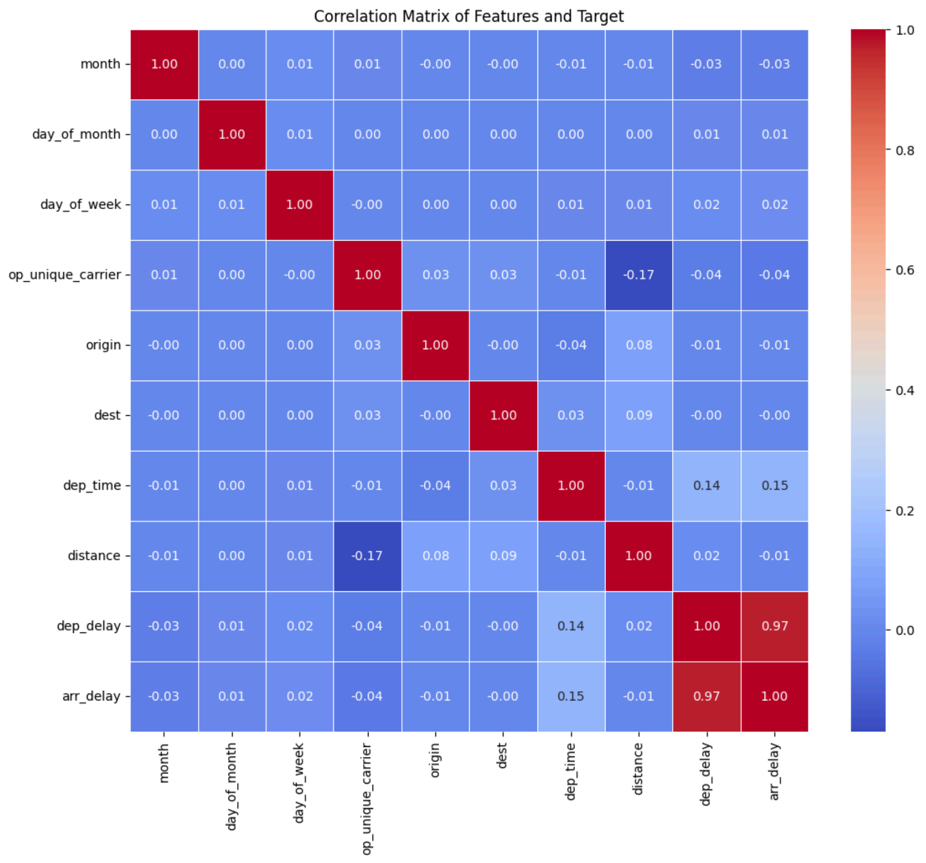
To evaluate the performance of our models, we used the following metrics: Accuracy, ROC AUC, PR AUC, as well as Precision, Recall, and F1-score specifically for the positive class (Class +), representing delayed flights.

While accuracy can give us a general sense of performance, we didn't rely on it due to the imbalanced nature of our dataset, where high accuracy can be achieved simply by predicting the majority class (non-delayed). Consequently, our primary focus was on metrics that highlight the detection of the minority class:

- Precision was used to monitor the accuracy of our model
- Recall was used in order to minimize the false negatives. In the context of our problem a delayed flight is considered more costly than falsely flagging a delay.
- F1-score balances the trade-off between detecting delays and maintaining prediction quality.
- The PR AUC was particularly valuable as it provides a more robust view of performance on imbalanced datasets compared to ROC AUC.

4.2. Regression

For regression we first analysed the correlation between all the variables and the target (arrival delay) to see how to approach the task.



It can be clearly seen that no variables except for dep_delay and maybe in a very small effect dep_time have any linear correlation with the target.

Consequently we decided to design an experiment to test two distinct hypothesis:

1. The control; We fitted the data **using also the departure delay column** to test how much of the arrival delay is just simply propagation of the departure delay.
2. The experiment; We hypothesized that a more complex, non-linear model like XGBoost Regressor might be able to predict the arrival delay based on structural factors **excluding the departure delay column.** That tests whether the model can rely on features strictly non-linearly correlated with the dependent variable.

For preprocessing, we encoded time-related features as cyclic function values (sine and cosine) and applied categorical encoding to high-cardinality features (dest, origin, op_unique_carrier) to ensure both linear and tree-based models received dense numerical vectors.

Except Theil-sen all linear regression models were fitted on a subset of 1 million samples, while the computationally heavy theil-sen was trained on 50 thousand samples. The XGB Regressor was fitted on 100 thousand samples. Afterwards we performed the hyperparameter tuning. For each model we used a RandomSearchCV over 5 folds to find the best combination of hyperparameters. For evaluating the final, optimized versions of the algorithms we used Mean Absolute Error, Mean Squared Error, Root Mean Squared Error, R-squared over 5 folds.

4.3. Clustering

For clustering we used four different previously described algorithms.

In terms of Gaussian Mixture, the approach was a bit more complex. Because of the inability of the algorithm to handle categorical data a more advanced technique was incorporated. We first trained a small pytorch [12] based autoencoder neural network on our data in order to compress the training examples into a latent space. That has many benefits for the algorithm including smaller computational cost and ensuring that the model receives dense numerical vectors on entry. That proved to be extremely beneficial for the model which provided clusters that were easy to interpret and useful for analysing delay root causes.

After fitting a baseline model of gaussian mixture (we used 1 million samples), we then optimised the hyperparameters of: the number of clusters (components), covariance type, initialization options (between KMeans and random), initialization iterations and covariance regularization. We created a parameter distribution space for the parameter sampler which then searched for 20 iterations for the best combination of parameters. For the optimized metric we used BIC (bayesian information criterion) which balances between awarding the model for fitting the data correctly and penalising it for extensive overfitting relative to the sample size. That resulted in finding a model that we considered the best and then evaluated it on a 50 thousand sample subset. We used a smaller sample size because scores like silhouette are very computationally heavy.

In the K-Means algorithm, first, we filtered the dataset to focus only on delayed flights, therefore excluding flights on-time from this specific analysis. Second, we performed feature engineering on the dep_time variable. The original text format (HH:MM:SS) was converted into a numerical feature representing the total minutes past midnight to streamline mathematical modeling.

We used the Elbow Method, analyzing the cost function for k ranging from 2 to 8. Based on that we choose k = 5, which provides the optimal balance between model complexity and variance.

For the DBSCAN algorithm, we initially normalized the numerical features and then applied PCA to reduce their dimensionality while preserving 95% of the variance. Due to the high computational cost of DBSCAN on large datasets, we chose a representative sample of 10,000 flights to execute the algorithm effectively. We subsequently conducted a systematic exploration of hyperparameters within a specified range for ε (the distance threshold) and min_samples (the least number of points needed to create a dense area). For every parameter set, we calculated the resulting number of clusters, the detected noise level, and the silhouette score only if a minimum of two clusters were formed. The dataset was extremely sparse, leading DBSCAN to naturally categorize many points as noise, which is anticipated for high-dimensional flight delay data featuring weak density patterns. According to the outcomes of the grid search, we chose the parameters that yielded the most significant cluster configuration while improving the silhouette score. The final parameters were subsequently employed to adjust the DBSCAN model on the chosen sample and produce the conclusive clustering outcomes.

For the BIRCH model we firstly performed feature scaling on the numerical variables (distance, departure delay) to ensure that each feature contributed equally towards clustering. Then we applied PCA to reduce the number of dimensions, while retaining the maximum variance and simplifying the clustering space in which BIRCH functions.

BIRCH was chosen because of its speed and scalability for extreme volume datasets, as our dataset contained several million records of flight data. Furthermore, BIRCH constructs an incrementally built CF (clustering feature) tree representing the condensed area of the dataset, thus allowing BIRCH to create its clusters naturally at a very low computational expense.

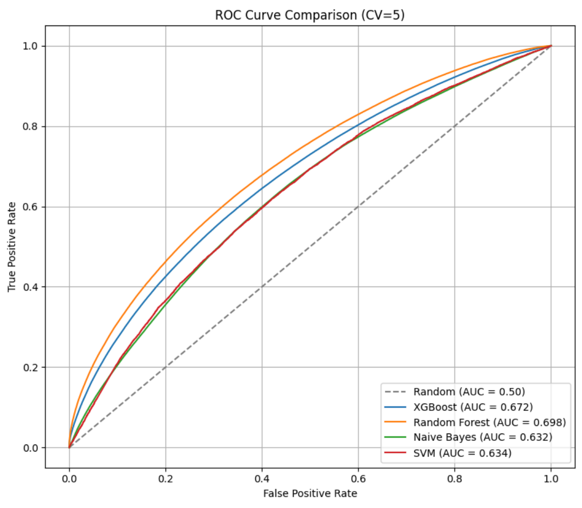
Once we fit the model, we assigned all delayed flights to a CF and calculated the summary statistics (mean and median) of the major delay-related features to interpret the behavioural differences between the clusters.

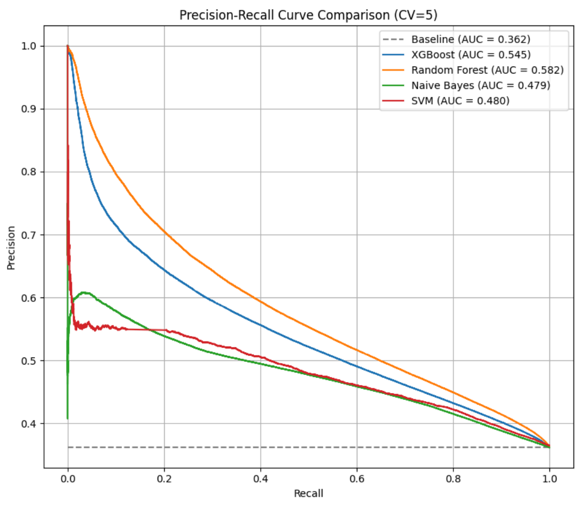
5. Results

5.1 Classification

Model	Accuracy	ROC AUC	PR AUC	Precision (Class +)	Recall (Class +)	F1-Score (Class +)
XGBoost	0,6284	0,6718	0,5455	0,4891	**0,6054**	0,5411
Random Forest	0,6852	0,6979	0,582	0,6602	0,2681	0,3813
Naive Bayes	0,6445	0,6324	0,4787	0,5161	0,2814	0,3642
SVM	0,5836	0,6344	0,4801	0,451	0,6506	0,5327

From this result we can conclude that for our case and for our definition of the problem we want to solve, XGBoost is by far the best model. Simply using accuracy will tell us that it’s the second worst model, however accuracy is a very poor metric, especially when considering problems with a class imbalance or problems where we want to favour one label over another. In our case as mentioned previously we wanted to prioritise recall of the positive class, as that is the most useful approach for this problem. And in that metric XGBoost and SVM are the best, but between the two models we choose XGBoost as the best, because the difference is just 5 percentage points and other metrics like precision, ROC AUC and PR AUC speak in favour of it. That made it a model with great recall, but also a balance in other metrics.





5.2 Regression

Model	MAE	MSE	RMSE	R2 Score
Ridge Regression	9,63	191,53	13,84	0,94
Lasso Regression	9,63	191,53	13,84	0,94
Polynomial Regression	9,57	189,3	13,76	0,94
Theil-Sen Regressor	**9,52**	**189,13**	**13,75**	**0,95**
RANSAC Regressor	9,57	191,73	13,85	0,94
Huber Regressor	9,58	191,7	13,85	0,94
XGB Regressor *Without departure delay column	27,33	333,9	57,74	-0,0185

In this table it can be clearly seen that the Theil-Sen regressor is the best, but by a very small margin. Nevertheless, we choose it as the best for further analysis. The XGB Regressor results and their comparison as part of the experiment discussed before, will be analysed in the conclusions part.

5.3 Clustering

Name	Silhouette	Calinski-Harabasz	Davies-Bouldin
Gaussian Mixture	0,059	1411,38	3,01
BIRCH	0,1108	17678,6	1,4483
K-Prototypes	0,0924	4875,1407	2,8072
DBSCAN *classified 99% of points as noise	0,8748	1687,7018	0,184

It's worth noting that, in the K-Prototypes algorithm, the metrics were calculated only on the numerical portion of the data (as scikit-learn metrics do not natively support the K-Prototypes distance function).

6. Conclusions

Firstly, our greatest limitation was the computational weight of the algorithms connected with the size of our dataset. Seven million samples is a lot even for the simplest of models and therefore often throughout this paper the reader can easily notice that we had to make sacrifices in the number of rows used for training to be able to train and evaluate the models. A great help came from the CUML library that allowed us to use GPU-based computations as well as the nativeness of that approach in the XGBoost library. However still it can be concluded that the result may have been different if we would have been able to use the whole dataset for all models.

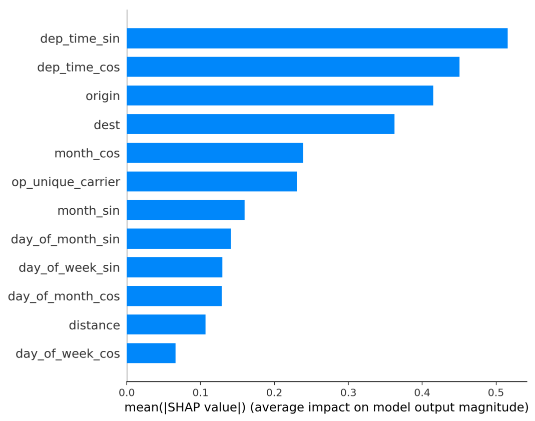
Regression may seem like it didn’t give any valuable insights, however we can derive the following conclusions. A complex, non-linear model (like XGBoost regressor) cannot correctly regress the number of minutes a flight will be late on arrival, whereas a simple linear model can calculate it using purely just the departure delay. This is easily visible while comparing metrics like the R-squared which for XGB Regressor is roughly below 0, whereas for linear models is around 0.94 - 0.95. That metrics tells us that around 95% of the variation in arrival delay can be explained by the departure delay. Further proof can be found below in SHAP values analysis and ALE analysis.

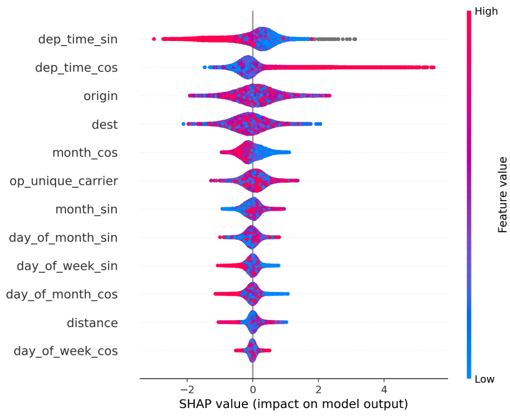
This proves that delays are stochastic, random operational events, rather than something that we can predict beforehand, regardless of model complexity. This is a valuable conclusion.

Classification yielded quite positive results, especially the XGBoost model which managed to balance other metrics with high recall, where main pressure in our methodology was put on it.

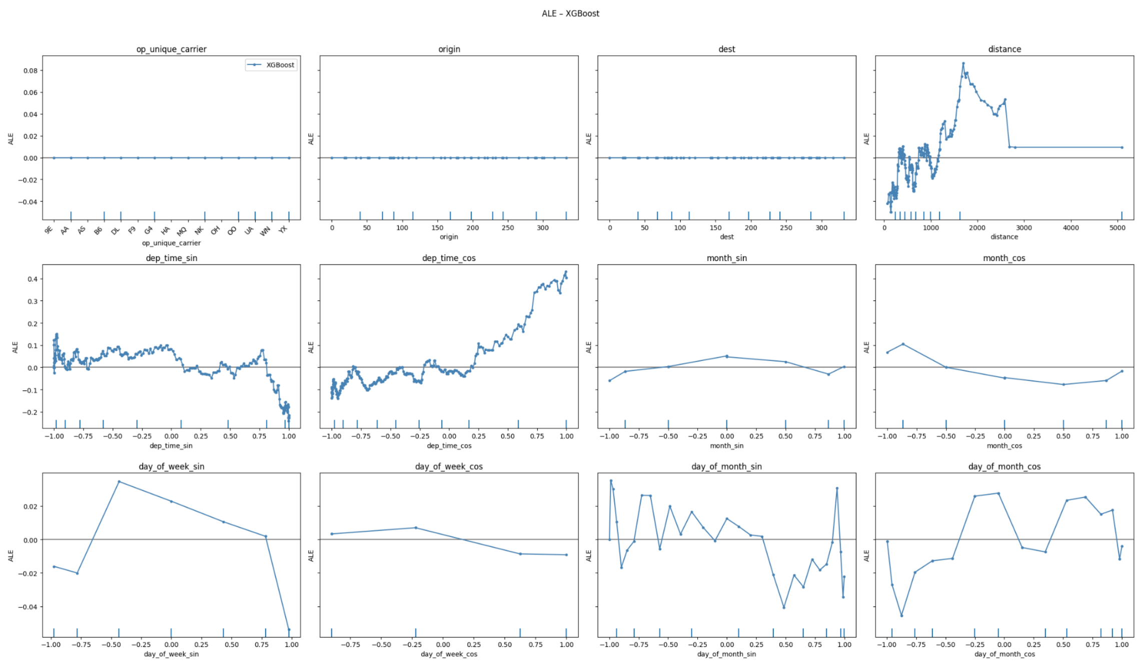
Clustering proved to be great for finding common features in delayed flight groups, which we will analyse below.

6.1 XGBoost



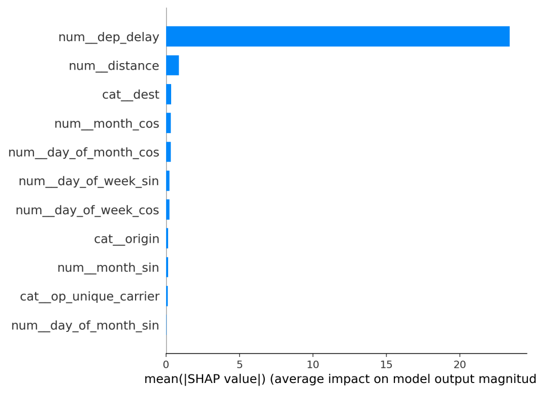


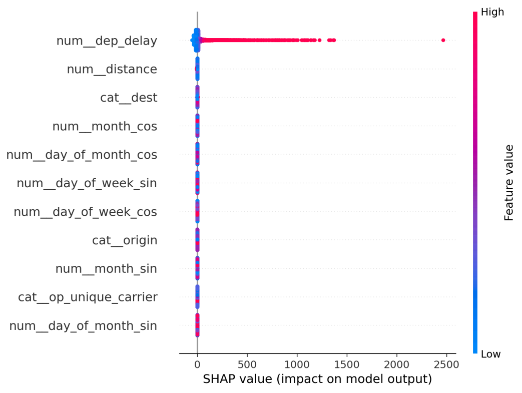
To understand the decision-making process of the XGBoost classifier (which we concluded as the best classification model for our problem), we employed SHAP values.
The time of day a flight departs is the strongest indicator of whether it will be delayed. This supports the common knowledge that delays often propagate and accumulate throughout the day, making later flights riskier than morning flights.
Following time, the origin and dest (destination) variables act as the second most critical features. This indicates that specific airports play a big role in the probability of a delay.
Moreover, the length of the flight is less relevant to delay probability than when it leaves and where it is going. A short flight leaving a congested airport in the evening is at higher risk than a long flight leaving a quiet airport in the morning.



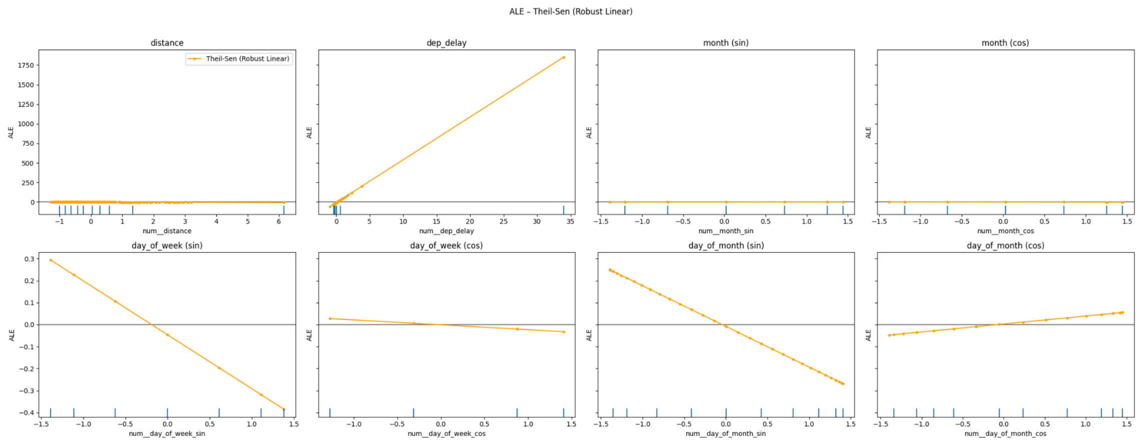
The ALE plot for the XGBoost classifier is a valuable insight into the model decision process. We can see that the departure time and distance have an explainable effect on the models predictions. Based on the departure time (encoded as cyclic functions, explanation: cosine in a 24 hour cycle will be around -1 at noon and +1 at midnight, whereas sine is -1 at around 6 pm and +1 at around 6am) flying at night can be more prone to arrival delays than during the day. That would be logical considering the effect of stacked delays throughout the day. In terms of the trip's distance, mostly affected are the medium-haul flights (1000-1800 miles), whereas the long-haul flights (more than 2500 miles) show an almost perfectly straight line indicating neutral effect. It is also worth noting that flights in the winter, based on the months (+1 winter, -1 summer for cosine), can be a bit more affected by delays. The rest of the features provide no reliable or stable patterns for explainability. Decoding day of week (by mapping the x-axis sine values back to specific days: Monday is 0.0, Friday is about 0.43, and Wednesday is about 1.0), we can find that Fridays and Mondays are the most prone to delays, whereas Wednesdays and Weekends are generally safer.

6.2 Theilsen





The SHAP summary plot identifies num_dep_delay (Departure Delay, num signifies a numerical feature) as the single dominant feature influencing the model's predictions.
This result confirms our earlier suspicions discussed in the methodology section and previous conclusions: arrival delay is almost entirely linearly correlated with departure delay and all other features are insignificant.



The Accumulated Local Effects (ALE) plots for the Theil-Sen model provide a direct visualization of how a robust linear model interprets the feature space. The plot for dep_delay confirms that this feature is the most important feature in the model's predictions. The ALE curve shows a steep, positive linear slope, where an increase in departure delay correlates directly with an increase in the predicted arrival delay.

The plots for distance and the cyclical month features (month_sin, month_cos) are flat lines at y=0. This indicates that the model found no linear correlation between flight distance or the time of year and the specific minutes of delay when dep_delay is present.

While the plots for day_of_week and day_of_month have a negative slope, a closer look at the y-axis shows their insignificance. The effect of these features fluctuates between -0.4 and +0.3. Compared to the dep_delay effect, a contribution of less than 1 minute is statistically negligible.

6.3 K-Prototypes

Based on the evaluation of our clustering models, we decided to come up with further conclusions for K-Prototypes, even though metrics may indicate that it’s slightly worse than BIRCH. However, BIRCH in our scenario was fitted using PCA - dimensionally reduced data and K-prototypes not, so it was our primary choice as explainability on BIRCH in that case is non-existent.

We made a detailed analysis of the centroids (using the mean for numerical features and mode for categorical features), which allowed us to identify and label five distinct profiles of delayed flights.

- Cluster 0 ("WN/CLT Local Shuttles"): This was the largest cluster, containing 10,610 flights. It represents short-distance trips (avg. 382 miles) and is overwhelmingly dominated by the Southwest (WN) carrier operating out of Charlotte (CLT).
- Cluster 1 ("WN/DFW Long-Haul"): Comprising 5,174 flights, this cluster captures long-distance routes (avg. 1,463 miles) dominated by Southwest out of the Dallas (DFW) hub.

- Cluster 2 ("WN/DFW Medium-Haul"): This cluster (8,740 flights) is also dominated by Southwest at DFW but captures medium-distance flights (avg. 945 miles). Ideally, the algorithm naturally separated the medium-haul routes from the long-haul routes in Cluster 1.
- Cluster 3 ("DL/LAX Transcontinental"): This was the smallest cluster (2,543 flights) but the most distinct. It characterizes the longest flights in the dataset (avg. 2,385 miles) and is the only cluster dominated by Delta (DL) out of Los Angeles (LAX). The distinct nature of these transcontinental flights was confirmed by a strong separation in the PCA plot.
- Cluster 4 ("WN/ATL Local Shuttles"): Containing 9,144 flights, this profile mirrors Cluster 0 but focuses on short-distance flights (avg. 535 miles) dominated by Southwest out of Atlanta (ATL).

By mapping the clusters to specific delay columns, two different delay reasons were discovered:

- In Cluster 0 (WN/CLT) and Cluster 2 (WN/DFW-Medium), the primary delay reason (accounting for ~47% of the delay) was late_aircraft_delay. This indicates a cascade effect where delays are caused by the late arrival of the aircraft from a previous flight.
- In Cluster 1 (Long-Haul), Cluster 3 (Delta/LAX), and Cluster 4 (Atlanta) the dominant issue was carrier_delay. This highlights a shared operational pattern: despite the distance disparity between the long-haul flights (Clusters 1 & 3) and the short-haul Atlanta routes (Cluster 4), they are both primarily affected by internal carrier operations rather than the cascade effect of arriving aircraft.

References

[1] Eurocontrol, 2023. *EUROCONTROL Data Snapshot #44: Causes of Flight Delays*. Brussels: EUROCONTROL. Avaliable at: https://www.eurocontrol.int/publication/eurocontrol-data-snapshot-44-causes-flight-delays [9.11.2025].

[2] Patil, H., 2024. *Flight Delay Dataset — 2024*. Kaggle. Avaliable at: https://www.kaggle.com/datasets/hrishitpatil/flight-data-2024/data [9.11.2025].

[3] Scikit-learn. (2024). scikit-learn: Machine Learning in Python. Scikit-Learn.org. https://scikit-learn.org/stable/

[4] Google. (2019). Google Colaboratory. Google.com. https://colab.research.google.com/

[5] PyPI · The Python Package Index. (n.d.). PyPI. https://pypi.org/

[6] Melton, J., Buxton, S., Teorey, T., Lightstone, S., Nadeau, T., Celko, J., Witten, I., Frank, E., Simsion, G., Witt, G., Schiller, J., Voisard, A., Halpin, T., Evans, K., Hallock, P., Maclean, B., Ceri, S., Fraternali, P., Bongio, A., &

Brambilla, M. (n.d.). Designing Data-Intensive Web Applications. https://mitmecsept.wordpress.com/wp-content/uploads/2017/04/data-mining-concepts-and-techniques-2nd-edition-impressao.pdf

[7] Matplotlib. (2024, May 30). Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation. Matplotlib.org. https://matplotlib.org/

[8] Numpy. (2024). NumPy. Numpy.org. https://numpy.org/

[9] Pandas. (2018). Python Data Analysis Library. Pydata.org. https://pandas.pydata.org/

[10] XGBoost developers. (2022). XGBoost Documentation — xgboost 1.5.1 documentation. Xgboost.readthedocs.io. https://xgboost.readthedocs.io/en/stable/

[11] Optuna. (n.d.). Optuna - A hyperparameter optimization framework. Optuna. https://optuna.org/

[12] PyTorch. (2024). PyTorch. Pytorch.org. https://pytorch.org/

[13] Waskom, M. (2024). Seaborn: Statistical data visualization. Seaborn.pydata.org. https://seaborn.pydata.org/