ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/abdulmohsen-eid-alshammari/683

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 9 & Iss: 2

Research Article

Flight Delay Prediction Using Machine Learning

Abdulmohsen Eid Alshammari*

Citation: Eid Alshammari A. Flight Delay Prediction Using Machine Learning. *J Artif Intell Mach Learn & Data Sci 2026* 9(2), 3441-3446. DOI: doi.org/10.51219/JAIMLD/abdulmohsen-eid-alshammari/683

Received: 26 April, 2026; **Accepted:** 01 May, 2026; **Published:** 04 May, 2026

***Corresponding author:** Abdulmohsen Eid Alshammari, Data Science, University of Hail, Saudi Arabia

A B S T R A C T

Flight delays are a real and persistent problem in the aviation industry. They cost airlines and passengers billions of dollars every year and cause huge issues for airports and air traffic controllers. The ability to predict whether a flight will be delayed - even approximately - has clear practical value for all parties concerned in this field.

In this project, we built a machine learning pipeline to tackle this prediction problem as a binary classification task: will a given flight be On Time or Delayed? We worked with a publicly available airline dataset that, after cleaning and preprocessing, contained 39,750 records and 21 features. One thing that became clear early on is that the raw dataset alone was not enough - we needed to engineer additional features to give the models something more meaningful to learn from.

We tested eight different classifiers: Logistic Regression, Random Forest, XGBoost, LightGBM, Gradient Boosting, CatBoost, a hyperparameter-tuned version of XGBoost and finally a Stacking ensemble that combined the best individual models. The Stacking model came out on top with an F1 Score of 0.6842 and an AUC-ROC of 0.7698. Perhaps the most interesting finding was that a single engineered feature - the historical delay rate of the departure airport - turned out to account for more than half of the model's predictive power.

1. Introduction

1.1. Why this problem matters

Anyone who has waited for hours at the airport due to a delayed flight knows how bad the experience can be. But the problem goes beyond mere personal inconvenience. Flight delays disrupt connecting flights, cause passengers to miss important appointments and events and create successive operational problems for airlines.

The economic scale of the problem is staggering. Studies estimate that flight delays cost the U.S. aviation system alone over 28 billion dollars per year when your account for airline costs, passenger time losses and indirect economic effects. Globally, the numbers are even larger. Despite decades of investment in air traffic management technology, delays remain stubbornly common - particularly during peak travel seasons and at congested hub airports.

Predicting delays in advance - even imperfectly - would allow airlines to reroute aircraft proactively, notify passengers before they reach the airport and optimize crew scheduling.

For passengers, a reliable delay prediction tool could help with travel planning and reduce the stress of uncertainty. This is what motivated us to take on this prediction problem as our graduation project.

1.2. The prediction problem

We treated flight delay prediction as a binary classification problem. For each flight record in our dataset, the goal is to predict one of two outcomes: On Time (class 0) or Delayed (class 1). The challenge is that flight delays result from a complex mix of factors - weather at the origin and destination, air traffic congestion, mechanical issues, late-arriving aircraft, crew availability and so on. Many of these factors are either not recorded in standard passenger datasets or are not known far enough in advance to be useful for prediction.

Our dataset contains passenger demographic information and airport details, but not real-time weather or traffic data. This is a realistic constraint - in practice, a prediction system that relies only on scheduled flight information and historical airport data would be far more deployable than one requiring live weather feeds. So, we deliberately worked within these constraints and tried to see how well we could do with what was available.

1.3. What we did

Our work involved several distinct steps. We started by cleaning and preprocessing the raw dataset. We then spent considerable time on feature engineering, creating new variables that we believed would help the models learn better patterns. After splitting the data and balancing the classes using SMOTE, we trained and compared eight machine learning models. We tuned the best-performing individual model using RandomizedSearchCV and then built a Stacking ensemble as a final step. Throughout, we paid attention to multiple evaluation metrics rather than just accuracy, since accuracy alone can be misleading with binary classification problems.

1.4. Paper organization

The remainder of this paper is divided as follows. Section 2 reviews related work. Section 3 describes the dataset and preprocessing steps. Section 4 explains our decisions regarding feature design. Section 5 covers experiment setup, including models, class balancing and evaluation metrics. Section 6 presents our results. Section 7 discusses what these results mean and where the approach falls short. Section 8 concludes with a summary and suggestions for future work.

2. Related Work

2.1. Early statistical approaches

Research on flight delay prediction has a fairly long history, going back to at least the early 2000s when large-scale flight records became publicly available. Early work tended to rely on statistical regression models and simple rule-based systems. These studies were useful for identifying which factors - weather, congestion, time of day, season - were most strongly associated with delays, but they struggled to capture the complex interactions between variables.

One consistent finding from this early literature, which our own results reinforce, is the importance of airport-level factors. Some airports are simply more delay-prone than others, due to their geographic location, runway capacity, traffic volume and local weather patterns. This motivated our decision to include historical airport delay rates as engineered features.

2.2. Machine learning takes over

Starting around 2015, machine learning methods began to dominate the flight delay prediction literature. Random Forest and Gradient Boosting classifiers became popular choices because they handle non-linear relationships well and are relatively robust to irrelevant features. Studies by Choi, et al.[1] on U.S. domestic flights reported AUC-ROC scores in the 0.72-0.78 range using these methods - numbers that are very close to what we achieved, which gives us some confidence that our results are reasonable.

More recent work has explored deep learning approaches. Ye, et al.[2] used LSTM networks to capture temporal dependencies between consecutive flights on the same aircraft. The idea is that a delayed inbound flight will almost certainly cause a delayed outbound flight, so modelling these sequences adds useful information. Our current approach does not model these sequences - each flight is treated independently - which is one of its main limitations.

3. Dataset and Preprocessing

3.1. The dataset

We used the Airline Dataset (Updated v2), which is publicly available on Kaggle. It was originally collected to support research on airline passenger behaviour and flight operations. Each row in the dataset represents one flight record associated with a specific passenger and includes a mix of demographic information about the passenger and operational details about the flight. Table 1 summarizes the key numbers.

Table 1: Dataset Overview.

Attribute	Details
Dataset Name	Airline Dataset Updated - v2
Total Records (Raw)	98,619
Records After Filtering	65,677
Records After Cleaning	39,750
Number of Features (Final)	21
Target Variable	Flight Status (0=On Time, 1=Delayed)
Source	Kaggle - flight-delay-prediction-abdulmohsen

The raw dataset has three possible values for the target variable Flight Status: On Time, Delayed and Cancelled. We excluded the Cancelled class from our analysis. Cancellations are fundamentally different from delays - they typically result from severe weather events, security incidents or major mechanical failures - and mixing them with delays would confuse the classifier. After removing cancelled flights, we were left with 65,677 records to work with.

2.3. The role of feature engineering

Several papers have highlighted how much feature engineering can improve results on aviation datasets. Kim and Hansen[3] showed that adding network centrality measures and route-level delay statistics led to substantial performance gains over models trained on raw flight data alone. This is consistent with our experience: the single biggest improvement in our pipeline came not from choosing a better algorithm, but from adding AirportDelayRate as a feature.

The general lesson seems to be that domain knowledge matters a lot in this problem. The best features are not necessarily the ones that sound most sophisticated - they are the ones that capture what actually causes delays, even if only indirectly.

2.4. Ensemble methods

Ensemble methods have consistently outperformed individual classifiers across a wide range of prediction tasks and flight delay prediction is no exception. The theoretical reason is well understood: by combining models that make different kinds of errors, you can reduce both variance and bias compared to any single model. Wolpert's Stacking framework is particularly appealing because it lets the meta-learner figure out which base models to trust on which types of inputs.

We were encouraged by findings from Kim and Hansen[3] and others showing that Stacking ensembles consistently beat individual gradient boosting models on aviation data. Our results confirm this pattern - the Stacking model achieved the highest F1 Score and AUC-ROC in our experiments, beating even the tuned XGBoost model.

3.2. Preprocessing steps

3.2.1. Target encoding: The Flight Status variable was converted to a binary integer: 0 for On Time and 1 for Delayed. This is required for compatibility with the scikit-learn API and enables probability-based metrics like AUC-ROC.

3.2.2. Dropping useless columns: We removed three columns immediately: Passenger ID, First Name and Last Name. These are personally identifiable fields that clearly cannot generalize to new passengers - a model that memorizes passenger names would be useless in practice.

3.2.3. Parsing the departure date: The Departure Date column was stored as a string. We parsed it into a proper datetime format and extracted three new columns: Month, Day and DayOfWeek. The original date column was then dropped since the raw date string is not useful to a machine learning model in its original form. Some rows had dates that could not be parsed correctly and were dropped along with any other rows containing missing values. After this step, the dataset contained 39,750 records.

3.2.4. Encoding categorical variables: Nine categorical columns were encoded using Label Encoding: Gender, Nationality, Airport Name, Country Name, Airport Country Code, Airport Continent, Continents, Arrival Airport and Pilot Name. We chose Label Encoding rather than One-Hot Encoding because several of these columns have very high cardinality - for example, Pilot Name has hundreds of unique values. One-Hot Encoding would have produced an unmanageably wide feature matrix. Tree-based models, which make up the bulk of our model comparison, handle label-encoded categorical well since they can find the relevant split points regardless of the numeric values assigned.

4. Feature Engineering

4.1. Why we needed new features

After initial preprocessing, the dataset had 14 features. When we ran a quick baseline XGBoost model on these features, the results were underwhelming. Digging into the data, it became clear that most of the raw features - passenger age, gender, nationality - do not have a strong direct relationship with flight delays. Delays are fundamentally an operational and geographic phenomenon, not a passenger one.

This led us to think about what information a domain expert would use to assess delay risk. The answer seemed obvious: the historical delay behaviour of the airport. An airport that has been running on-time 95% of the time in the past is much more likely to run on time tomorrow than one that has been delaying 60% of its flights. This reasoning motivated the creation of our delay rate features.

4.2. The engineered features

We created seven new features in total. **(Table 2)** describes each one along with our reasoning for including it.

Table 2: Engineered Features.

Feature	Description	Why We Added It
Month	Month extracted from departure date	Flight delays change a lot by season
Day	Day of the month	Some days are busier than others
DayOfWeek	0=Monday to 6=Sunday	Weekends have very different traffic patterns
Season	Winter/Spring/Summer/Fall grouping	Smoother seasonal signal than raw month
IsWeekend	1 if Friday or Saturday, else 0	Peak travel days tend to have more delays
IsInternational	1 if nationality differs from airport country	International flights behave differently
AirportDelayRate	Historical average delay rate per airport	By far the strongest predictor we found
CountryDelayRate	Historical average delay rate per country	Captures country-level delay patterns
ContinentDelayRate	Historical average delay rate per continent	Useful when country data is sparse
MonthDelayRate	Historical average delay rate per month	Captures monthly seasonality in delays

4.3. The most important feature: AirportDelayRate

AirportDelayRate deserves special attention because it turned out to be far more important than any other feature - not just the engineered ones, but all 21 features combined. It is computed simply by grouping the dataset by Airport Name and taking the mean of the binary Flight Status variable. The result is a number between 0 and 1 representing the fraction of flights from that airport that were delayed in our dataset.

When we ran feature importance analysis on the tuned XGBoost model, AirportDelayRate had an importance score of approximately 0.52. In other words, this single feature accounts for more than half of what the model is actually using to make its predictions. The next most important feature, CountryDelayRate, had an importance of only about 0.08.

One important caveat: because we compute AirportDelayRate from the same dataset we use for training and testing, there is a

potential data leakage issue. In a real production system, you would compute these rates from historical data that strictly predates the flights you are trying to predict. We acknowledge this limitation but note that it is common in research settings where longitudinal data is not available.

4.4. Temporal features

The Season feature groups months into four seasons and provides a smoother signal than raw Month values. IsWeekend flags Fridays and Saturdays as peak travel days - from our own experience as travellers, these days tend to have more congestion and therefore more delays. DayOfWeek gives the model access to the full weekly pattern, not just the weekend/weekday distinction.

4.5. IsInternational

This binary feature is a rough proxy for whether a flight is international or domestic, based on whether the passenger's nationality matches the country of the departure airport. It is an imperfect measure - a Saudi passenger flying within Saudi Arabia would be correctly flagged as domestic, but a foreign resident flying domestically would be incorrectly flagged as international. Despite this imprecision, the feature contributes some useful signal, showing up as the fifth most important feature in our XGBoost model.

5. Methodology

5.1. Train/Test split

We divided the dataset into training and test sets at a 70/30 ratio using stratified random sampling. Stratification ensured that both sections contained the same proportion of delayed and on-time trips. The test set was kept aside until the end and was only used to report the final results. I was careful not to leak any information from the test set into the training process.

5.2. Handling class imbalance with SMOTE

Even though the overall class distribution is roughly balanced after filtering, we applied SMOTE to the training set as a precaution. SMOTE creates synthetic minority class samples by interpolating between existing real samples in feature space. The key rule is that SMOTE must only be applied to the training set - applying it to the test set would give an unrealistically optimistic picture of performance on real data.

To check whether SMOTE actually helped, we ran XGBoost with and without it. Without SMOTE: F1=0.6102. With SMOTE: F1=0.6584. That is a meaningful improvement of about 7.9%, so we kept SMOTE in the pipeline.

5.3. Models we compared

We trained eight models in total. **(Table 3)** gives a quick overview of each one.

Table 3: Models Used in This Study.

Model	Type	Notes
Logistic Regression	Linear	Used as a simple baseline
Random Forest	Bagging Ensemble	200 trees, good general performance
XGBoost	Boosting	Default settings first, then tuned
LightGBM	Boosting	Faster than XGBoost on large data
Gradient Boosting	Boosting	Sklearn implementation
CatBoost	Boosting	Handles categoricals well
XGBoost (Tuned)	Boosting	After RandomizedSearchCV optimization
Stacking Ensemble	Meta-learning	Combines GB + LGB + RF + XGB(Tuned)

Logistic Regression was included as a linear baseline - if it performs competitively, that suggests the problem is mostly linear and simpler models might suffice. Random Forest provides a strong non-linear baseline through bagging. The four gradient boosting variants (XGBoost, LightGBM, Gradient Boosting, CatBoost) represent the current state of the art for tabular classification problems. Finally, the Stacking ensemble combines the predictions of the four strongest individual models using Logistic Regression as a meta-learner.

5.4. Hyperparameter tuning

We selected XGBoost for hyperparameter tuning because it is one of the most widely used and well-understood gradient boosting implementations, with a rich set of hyperparameters to explore. We used RandomizedSearchCV with 30 random configurations and 3-fold cross-validation, optimizing for F1 Score. The search covered seven hyperparameters: n_estimators, learning_rate, max_depth, subsample, colsample_bytree, min_child_weight and gamma.

The best configuration found was: n_estimators=200, learning_rate=0.05, max_depth=4, subsample=0.7, colsample_bytree=0.7, min_child_weight=5, gamma=0.1. Interestingly, the optimal depth (4) was shallower than our default setting (8), suggesting that the default model was overfitting. The higher regularization parameters (min_child_weight=5, gamma=0.1) also point in this direction.

5.5. Evaluation metrics

We evaluated all models using eight metrics: Accuracy, Balanced Accuracy, Precision, Recall, F1 Score, AUC-ROC, Matthews Correlation Coefficient (MCC) and Log Loss. We used F1 Score as our primary ranking metric because it balances Precision and Recall - in a delay prediction context, both false positives (predicting delay when the flight is actually on time) and false negatives (predicting on time when the flight is actually delayed) have real costs. AUC-ROC served as our secondary metric because it measures discrimination ability across all classification thresholds, not just the default 0.5 cutoff.

6. Results

6.1. Overall Model Comparison

(Table 4) shows the performance of all eight models on the test set, ranked by F1 Score. The results paint a pretty clear picture.

Table 4: Model Performance on Test Set (ranked by F1 Score).

Model	Accuracy	F1 Score	AUC-ROC	MCC	Log Loss
Stacking (Best) ★	0.6846	0.6842	0.7698	0.3692	0.6021
Gradient Boosting	0.6787	0.6800	0.7574	0.3575	0.5661
XGBoost (Tuned)	0.6793	0.6787	0.7635	0.3587	0.6004
LightGBM	0.6741	0.6755	0.7523	0.3483	0.5719
CatBoost	0.6752	0.6768	0.7526	0.3505	0.5892
Random Forest	0.6660	0.6657	0.7369	0.3320	0.5901
XGBoost	0.6573	0.6584	0.7309	0.3146	0.6004
Logistic Regression	0.5754	0.5726	0.6230	0.1509	0.6673

The Stacking ensemble came out on top with F1=0.6842 and AUC-ROC=0.7698. Gradient Boosting was a close second (F1=0.6800) and the tuned XGBoost ranked third (F1=0.6787). Logistic Regression was by far the weakest model, with F1=0.5726 and AUC-ROC=0.6230 — barely better than random guessing on the AUC metric.

One result that surprised us a bit was that CatBoost (F1=0.6768) performed similarly to LightGBM (F1=0.6755) despite CatBoost's reputation for handling categorical features well. We think this is because we label-encoded all categoricals before training, which removes the advantage CatBoost would normally have from handling raw categoricals internally.

6.2. Feature importance

(Table 5) shows the top 10 features by importance in the tuned XGBoost model. The dominance of AirportDelayRate is striking.

Table 5: Top 10 Feature Importances (Tuned XGBoost).

Rank	Feature	Importance	Type
1	AirportDelayRate	~0.52	Engineered
2	CountryDelayRate	~0.08	Engineered
3	Month	~0.06	Temporal
4	Nationality	~0.05	Demographic
5	IsInternational	~0.04	Engineered
6	Pilot Name	~0.04	Operational
7	Arrival Airport	~0.03	Operational
8	Airport Name	~0.03	Operational
9	MonthDelayRate	~0.03	Engineered
10	Day	~0.02	Temporal

AirportDelayRate alone accounts for approximately 52% of the total importance. The next nine features combined account for the remaining 48%. This tells us something important: for this dataset, the strongest signal for whether a flight will be delayed is not anything about the passenger - it is about which airport the flight is departing from and that airport's historical delay behaviour.

The low importance of demographic features like Age (~0.02) and Gender (~0.01) is consistent with our intuition. There is no particular reason why a passenger's age or gender should influence whether their flight is delayed - delays are an operational phenomenon driven by airport capacity, weather and network congestion.

6.3. Effect of hyperparameter tuning

The tuning process improved XGBoost's F1 Score from 0.6584 (default) to 0.6787, a gain of about 3.1%. The AUC-ROC improved from 0.7309 to 0.7635. These are meaningful improvements, though not transformative. The direction of improvement - toward shallower trees with more regularization - suggests that the default XGBoost configuration was overfitting to the training data.

6.4. The stacking ensemble

The Stacking ensemble, which combines Gradient Boosting, LightGBM, Random Forest and tuned XGBoost as base learners with Logistic Regression as the meta-learner, achieved the best results overall. The improvement over the best individual model (Gradient Boosting) is modest in absolute terms - about 0.004 F1 points and 0.012 AUC points - but consistent across all metrics. This is typical for stacking: the gains are usually incremental rather than dramatic, but they are reliable.

7. Discussion

7.1. Why the results are as they are

Our best model achieved an F1 score of 0.6842 and an ROC AUC of 0.7698. These numbers are good but not excellent. To put it in context: an AUC of 0.77 means that our model ranks a randomly selected delayed flight higher than a randomly selected on-time flight about 77% of the time - much better than random (0.50) but far from perfect (1.00). We believe the main reason for the performance ceiling is the absence of weather data. Weather is widely known as the primary cause of flight delays and our dataset does not include it. No amount of computational complexity can make up for the loss of the most important predictors. Studies that include weather features usually report AUC scores of 0.85 or higher. The secondary reason is that our model treats each flight individually. In reality, delays propagate through the network - if a plane is delayed at one airport, it delays all subsequent flights on that plane's schedule.

7.2. The airport effect

One of the most interesting findings of this study is just how dominant the airport-level delay rate is as a predictor. When we first saw that AirportDelayRate accounted for 52% of XGBoost's feature importance, we double-checked our code to make sure we had not made an error. The result held up.

This finding has a practical implication: if you want a simple, interpretable rule for predicting flight delays, just look at the historical delay rate of the departure airport. A flight departing from a high-delay airport is much more likely to be delayed than one departing from a low-delay airport, regardless of anything else. Of course, this is not the full story - the remaining 48% of importance comes from other features - but the airport effect is by far the strongest single signal in the data.

7.3. Ensemble vs. individual models

The consistent advantage of the Stacking ensemble over individual models is worth reflecting on. The base learners (Gradient Boosting, LightGBM, Random Forest, XGBoost) each make somewhat different errors. Some may be better at handling certain airport types, certain passenger demographics or certain seasonal patterns. The meta-learner - the Logistic Regression trained on the base model predictions - learns to weight these models optimally, producing a combined prediction that is more reliable than any individual one.

That said, the gains from stacking in this case are modest. If we were deploying this in a real system where inference speed matters, we might opt for the tuned XGBoost alone rather than the full Stacking ensemble, since the performance difference is small and the ensemble is more complex to maintain.

7.4. Limitations

We want to be honest about the limitations of this work

- **No weather data:** This is the biggest gap. Weather is the primary cause of delays and its absence puts a ceiling on achievable performance.
- **Data leakage risk in delay rates:** Computing AirportDelayRate from the same dataset used for training and testing is not ideal. In production, historical rates should be computed from data that predates the prediction period.
- **Static model:** We trained one model on the full dataset. A deployed system would need to be retrained regularly as delay patterns change over time.
- **No sequence modelling:** Each flight is treated independently, ignoring the cascading nature of real-world delays.
- **Single dataset:** Results may not generalize to other airlines, time periods or geographic regions.

8. Conclusion

This project set out to build a machine learning pipeline for predicting flight delays as a binary classification problem. The dataset contains 39,750 flight records that have been cleaned and processed, we compared eight classification models and found that a Stacking ensemble combining Gradient Boosting, LightGBM, Random Forest and tuned XGBoost achieved the best results, with an F1 Score of 0.68 and AUC-ROC of 0.76.

Several things have become clear to us through this project, feature engineering mattered more than model selection. The single most impactful step in our entire pipeline was adding AirportDelayRate - a simple historical average computed from the data. No algorithmic choice came close to matching the effect of that one feature. Second, ensemble methods consistently beat individual classifiers, confirming what the literature has been saying for years. Third, hyperparameter tuning produced meaningful but not dramatic gains, suggesting that the default configurations of modern gradient boosting libraries are already quite good.

Looking ahead, adding weather data at the origin and destination airports would likely be the single highest-impact improvement. Modelling flight sequences rather than treating each flight independently would capture delay propagation effects. If SHAP values are used for interpretability, it will be easier to explain individual predictions to end users. And deploying the model as a real-time API - one that updates its airport delay rates from a live feed - would turn this from a research exercise into a practical tool.

Despite its limitations, we believe this project demonstrates that useful delay prediction is achievable even with relatively simple, readily available data - as long as you invest in the right features.

9. References

1. Choi S, Kim YJ, Kim S. Flight delay prediction using machine learning algorithms. Journal of Air Transport Management, 2021;90: 101837.
2. Ye B, Liu B, Tian Y, et al. A methodology for predicting aggregate flight departure delays in airports. Sustainability, 2020;12: 2749.
3. Kim J, Hansen M. Ensemble methods for aviation delay prediction. Transportation Research Record, 2022;2676: 54-65.
4. Chen T, Guestrin C. XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 785-794.
5. Chawla NV, Bowyer KW, Hall LO, et al. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 2002;16: 321-357.
6. Dietterich TG. Ensemble methods in machine learning. Proceedings of the First International Workshop on Multiple Classifier Systems, 2000: 1-15.
7. Gui G, Liu F, Sun J, et al. Flight delay prediction based on aviation big data and machine learning. IEEE Transactions on Vehicular Technology, 2020;69: 140-150.
8. Ke G, Meng Q, Finley T, et al. LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 2017;30.
9. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 2011;12: 2825-2830.
10. Prokhorenkova L, Gusev G, Vorobev A, et al. CatBoost: Unbiased boosting with categorical features. Advances in Neural Information Processing Systems, 2018;31.
11. Rebollo JJ, Balakrishnan H. Characterization and prediction of air traffic delays. Transportation Research Part C, 2014;44: 231-241.
12. Wolpert DH. Stacked generalization. Neural Networks, 1992;5: 241-259.