Delay prediction 2023

Model and solution lifecycle

Marc Susagna Holgado
15/12/2022

Part 1: Solution overview

Solution overview

- Task: Delay prediction for 2023 assuming 2022 schedules and a demand increase

- Solution schema:
  - y = f(x_demand, x_other) where y is leg delay
  - Get x_demand for 2022 and increase it by a factor (consider aircraft capacity) → Obtain x_demand’
  - Predict on the new dataset with x_demand’ based on 2022 legs

- There are two relevant business questions:
  - How many flights are delayed? Fit a classification model, y is binary
  - How is the delay time distributed? Fit a regression model, y is zero inflated and with a long tail
    - This can help answer business questions as: How many flights have more than 15 min delay?
    - Regression model usually predicts E[Y|X] but for business questions we might need P(Y|X) (quantiles)

The model (1/3)

- Data preparation: Join the following datasets
  - Delay.csv: Aggregate delay types per leg
  - Fis.csv: Date casting
  - Aircraft capacity: New dataset containing max pax seats per aircraft subtype

- Feature extraction:
  - Total pax (our x_demand!), aircraft pax occupancy
  - Date features: departure month (UTC), day of month + circular (LOC), hour of day + circular (LOC), day of week
  - Time difference (offblock to airborne…)
  - Number of flights and flights cancelled on same departure airport and day
  - Some new categorical (scheduled != actual) and ratios to prevent multicollinearity

- Data Split:
  - Train: 80% of 2021. CV used here for within class model selection.
  - Validation: 20 % of 2021. Used for between class comparison and model lifecycle (more on it later)
  - Test: 100% of 2022. Explained later on model evaluation.

The model (2/3)

- Model development and selection:
  - V0.0.1: Linear and logistic regression with x = {total_pax, aircraft pax occupancy, dep ap}
  - Next versions: Onboard all features, RFs, XGB
    - All models have a stored blueprint in model registry with: skeleton, relevant metrics…

- Model evaluation
  - ML metrics (accuracy, R2) are OK for model selection… but we want business-relevant metrics of the solution
  - Task reminder: Delays in 2023 with 2022 schedules and increased demand… replicate this for 2022!
    - Train model on all 2021 data
    - Get total pax increase between 2021 and 2022 (call the increase “x”)
    - Get flights from 2022 that existed also in 2021 (join on: flight carrier, flight, departure DD/MM)
    - Replace pax in 2022 by pax from 2021 for the same flight.
    - Apply: new_pax = min(total_num_pax_seats , pax_2021*(1+x))
    - Predict on this contrafactual 2022 dataset
    - Compare distribution of delays in the original 2022 flights (before step 3) vs the predicted one

The model (3/3)

- Model training:
  - Train on all data: 2021 and 2022

- Model deployment / serving:
  - Take model artifact: Trained model, model blueprint, metrics, source code and 2022 data
  - Deploy with a Flask API that gets user input to show:
    - Model metrics
    - Any prediction of 2023 delays according to a manually input of expected demand increase
      - Applied as for model evaluation, i.e. considering total available pax_seats

- Model Monitoring:
  - Not implemented. An example: Increase demand leads to increased delays? Are users giving bad feedback?

Part 2: Actual implementation, auditing and model lifecycle

The repository

- Link: https://github.com/marcsusagna/delay_prediction
- It has 5 main scripts:
  - Model_development_main.py: A jupyter notebook template to experiment and populate model registry
  - Model_evaluation_main.py: A main file to get test and business metrics for a given model
  - Train.ps1 / .sh: Training pipeline deployment
  - Test_model_in_dev.ps1 / .sh: Deploys a front end with a specific model version to check behavior
  - Ci_cd.ps1 / .sh:
    - Performs Continuous Integration: Run unit tests
    - Performs Continuous Delivery: Creates docker image for front end deployment
    - Performs Continuous Deployment:
      - Deploys to test environment. Asks for user input (UAT)
      - Then Deploys to Production

Model lifecycle: First deployment

- Goal: Deploy the latest version of the model and boot up a front end API to interact with it

- Steps:
  - Run data pipeline and model training
    - Option 1: Locally (need to run 3 python scripts)
    - Option 2: Run training into a deployable container (need to run 1 shell script) Why?
      - PROD data more complete, non obfuscated values…
      - Your local machine doesn’t meet the requirements

  - Deploy model to PROD environment
    - Run ci_cd.ps1

Model lifecycle: Developing and deploying a new version

- Steps:
  - Branch off master, upgrade new version
  - Create new model: feature extraction, model, hyperparam… → Store its blueprint in model registry
  - Run model evaluation to populate performance metrics → Update model blueprint
  - Run train (locally or deploying it) → Store trained model in model registry
  - Deploy front end with new model in DEV environment
  - Upgrade current model version, raise PR, merge to master
  - Run CI / CD pipeline to promote to PROD

- This way all changes have an audit trail and previous versions of the model can be recovered easily

Model lifecycle: Model retraining

- If PROD performance deteriorates (user feedback, performance…) we can just retrain the model by:
  - Executing train.ps1
  - Executing ci_cd.ps1

- This could be done based on alert based triggered

→ The solution allows for continuous deployment

- If new data comes in (e.g: Solution is "predict next year with the last 2 years data"):
  - Having rolling windows to pick correct data
  - Run train.ps1 and ci_cd.ps1 in a scheduled manner so that PROD model is always up to date

Part 3: Evaluating the final solution v0.0.7

Comparing v0.0.1 to v0.0.7

- Comment on
    - ML metrics
    - Delay frequency
    - Delay time distribution

- Did not spend much time on it due to:
    - Simplistic model (missing relevant features: weather, airport contextual features…)
    - Lack of expert knowledge

Part 4: Discussion, improvement and next steps

Discussion

- Discussion on the design:
  - No causality and missing many features → x_demand effect might be misleading
  - How to choose a model: Accuracy / R2 score vs evaluation (overfitted model)
  - Evaluation setup: 2022 into 2021 (no overestimate score) vs 2021 into 2022 (replicate 2023 procedure)

- Better model
  - More data: performance improves significantly if 80 % of all data as training
  - Features: Weather, airport features on the day (num flights, pax, delays, num employees)
  - Proper Grid search and hyperparameter tuning
  - Handle outliers / model long tailed distributions

- Code:
  - Use open source popular solutions
    - CI/CD: Jenkins
    - Model registry: MLflow

Discussion

- Model serving
    - Better demand feature
        - X is correlated, increase in pax would most likely increase other features (e.g baggage weight)
        - Distribution of the pax increase does not need to be necessarily uniformly over legs
            - By airport, route, month of the year…
        - Use aggregated values for demand instead of leg pax: Rolling avg pax over an airport

    - Business value metrics:
        - Number of pax affected / aggregated delay time
        - Regression: Model P(Y|X) instead of E[Y|X], so that you can give predictive quantiles
            - Possible solution: Zero inflated models
        - Client perspective of delay: Delay by flight not leg (current is airline perspective)

    - Provide expected demand increase
        - Monthly ARIMA on pax