# Using weather to help predict airport configuration

# Raw data

- 10 airports
- Weather data
- Aircraft actual departure/arrival time for each aircraft
- Aircraft estimated/scheduled departure/arrival time for each aircraft
- First time when aircraft is tracked for each aircraft
- Aircraft arrive/depart from gate/runway for each aircraft
- Weather prediction 25 hour lookahead @ 30 minute increment
- Current airport configuration

# Provided infrastructure (baseline code given)

- Recency-weighted historical forecast
  - Achieves .098 log-loss error
    - After hyperparameter grid search for each lookahead/airport: ~.091 log-loss error
  - Input: Takes in current configuration and distribution of past airport configuration
  - Output: Probabilities of future airport configurations (30-minute intervals over 360 minutes)
  - Hyperparameters:
    - Weight: weight for the current configuation
    - Hedge: weight for a uniform distribution
    - Discount: weight for the "current configuration" (changes/discounts as time grows)
      - Discount_rate = (discount)^ (360/minutes)

We always add a uniform distribution because if you are wrong and the value is zero log loss will be infinity: VERY VERY BAD. To compensate we give everything a really tiny weight "just in case"

# My piece

- **Multi<u>class</u>** calculation: calculate the airport configuration
  - Input:
    - Forecasted weather data @ prediction time (forecast_time, temperature, wind direction, wind_speed, wind_gust, cloud ceiling, visibility, cloud, lightning_prob, precipitation) = 10 features
    - Current weather data (… same …) = 10 features
    - Prediction time = 1 feature
    - Average estimated aircrafts landing rate per minute for the past 1 hour = 1 feature
    - Average estimated aircrafts taking off rate for the past 1 hour = 1 feature
    - Average estimated aircraft landing rate for the next 1 hour = 1 feature
    - Average estimated aircrafts taking off rate for the next 1 hour = 1 feature
    - Current configuration (?) --> between 10 to 40 feature (different for each airport)

# My piece Cotd.

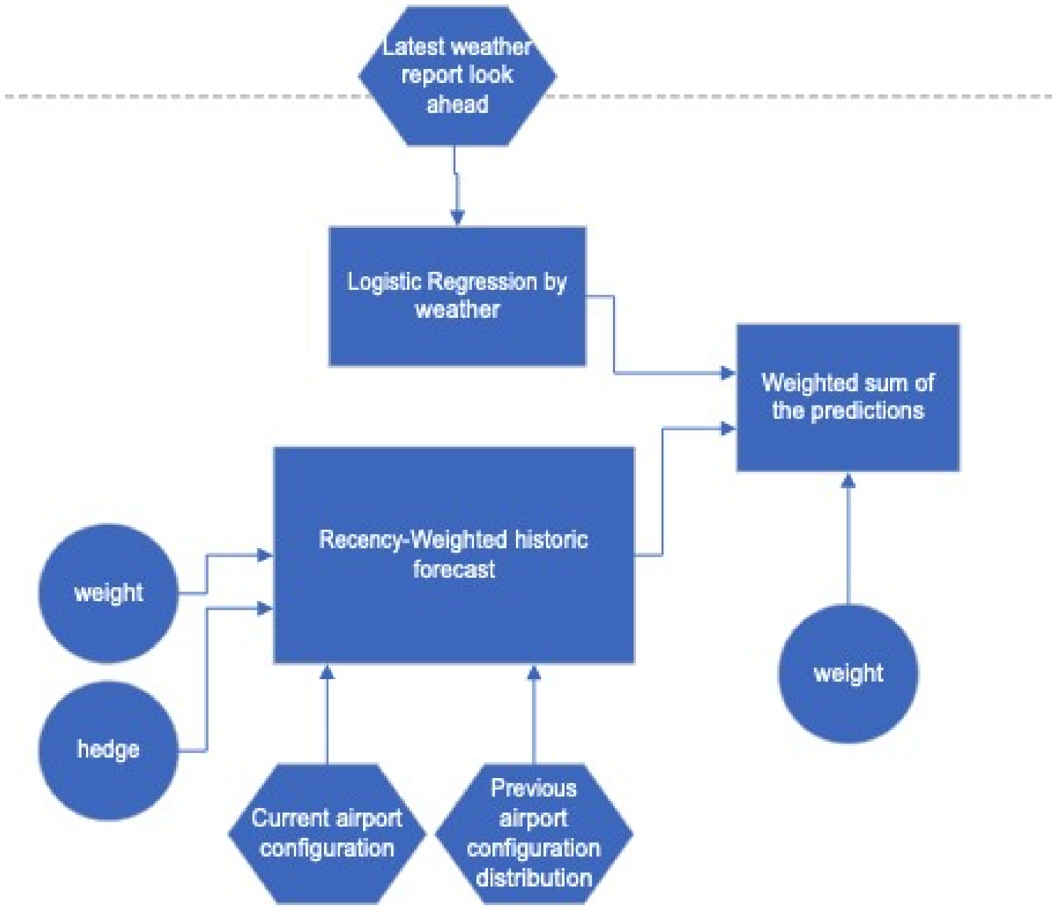
- Output:
  - Lookahead configuration.
    - # configuration for each airport ranges from 12 to 42
    - Kalt: 27, kclt: 13, kden: 42, kdfw: 31, kjfk:14, kmem: 31, kmia: 28, kord: 38, kpnx: 18, ksea: 12

- Note: we are training a total of 10 LR models --> 1 for each airport. Each airport has <u>a different number</u> of number of possible classes/labels.

# Use logistic regression

- Why: I want **probabilities**, not just classification, of the liklihood of each configuration.
- Use kernel trick to do x^2.
- Logistic regression minimizes log-loss ==> the same minimization metric as competition
  - If we use random forest, for example, we won't account for the fact that one wrong that is prediction with high probability can significantly skew/increase the error.

# Summary of idea

# "Above and beyond":

- Random forest
  - Use random forest to determine the weight between regency and logistic regression
    - Takes the same input as Logistic Regression **PLUS current configuration** (discrete variable with.
- AdaBoosting:
  - Regenccy --> high bias/low variance
  - Logistic Regression --> ??
  - Random Forest --> low bias/high variance

# Computation needed

1. Hyperparameter tuning: 2D hyperparameter tuning using ROAR for regency
   - Use threading
   - Without threading: takes about 5 hours to tune 10*17 using 100 data points per trial for 10 airports * 12 lookahead per airport. (hyperparameter is <u>decoupled</u> with respect to each [airport, lookout] combination because sum of log-loss is communicative)
     - Want to increase to 20*30 using 500 data points each
       - Hopefully threading+ROAR will significantly decrease time needed
2. Training the 10 Logistic Regression model
3. 1D hyper parameter tuning for LR weights

# Considerations for recurrent neural network

Pros:

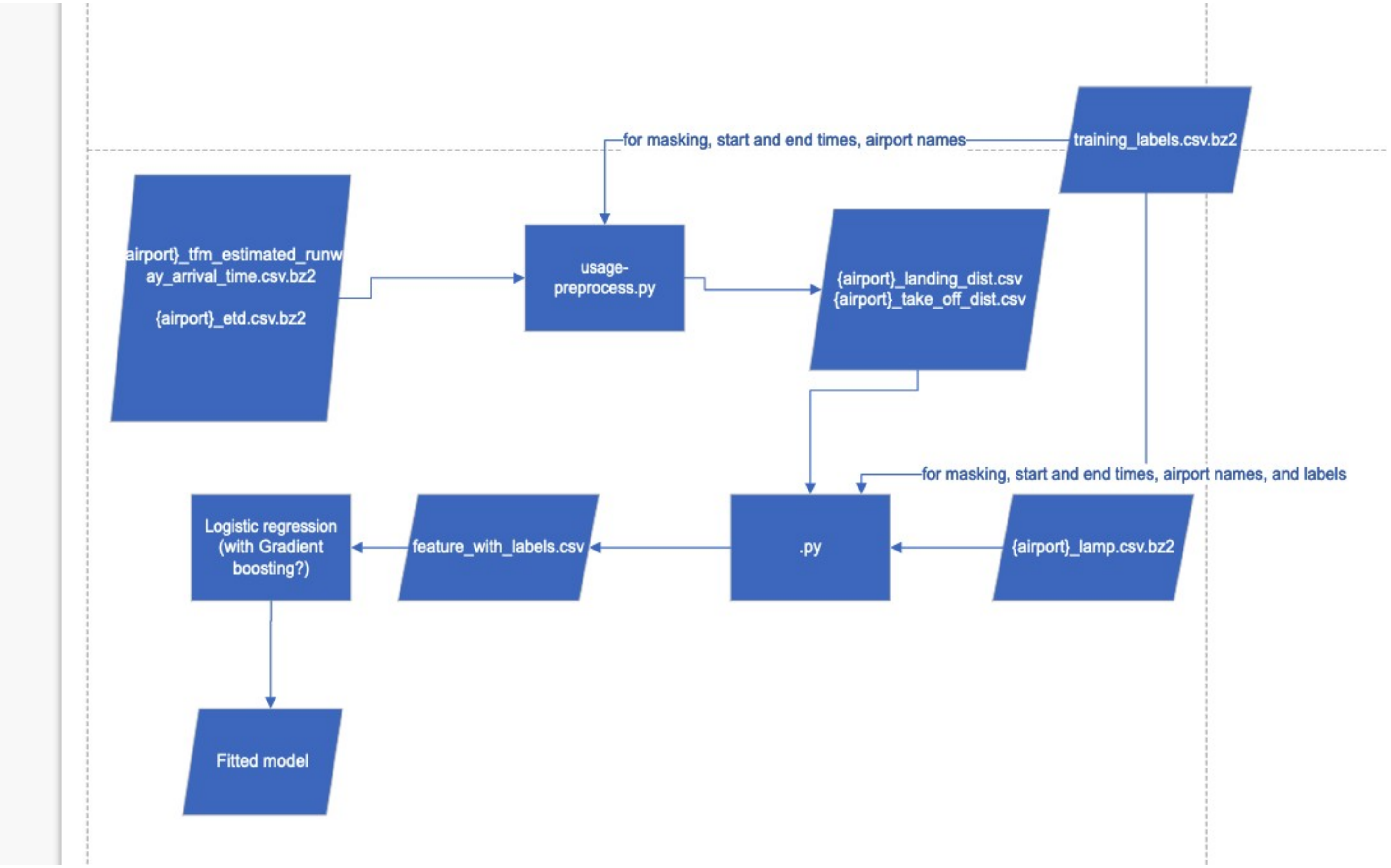
- Our data and predictions are of a temporal nature

Cons:

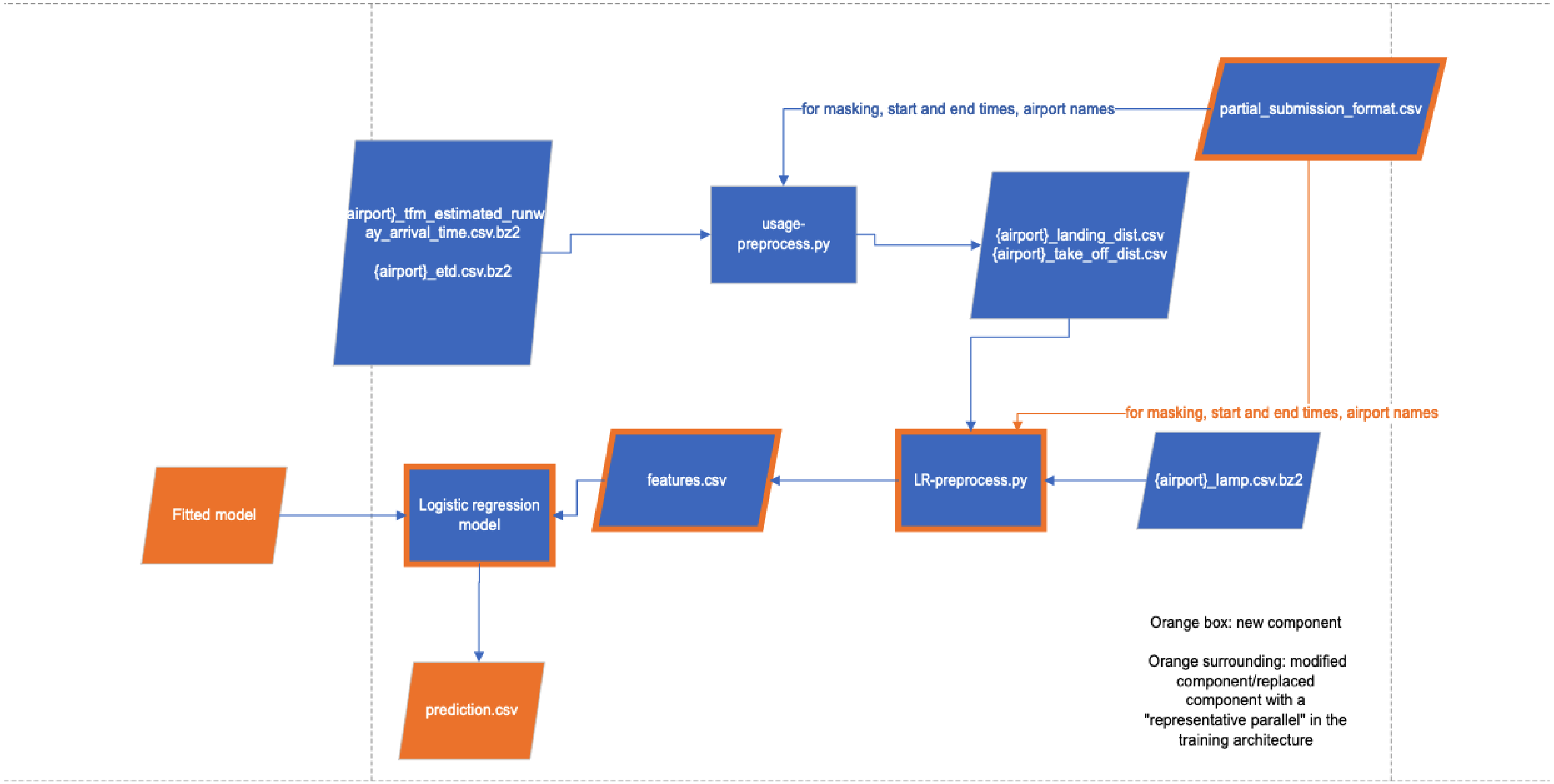
- Training dataset split/specialized 10 airports * 12 lookahead is insufficient to train a large neural network
  - Generalizing across lookahead and airports will likely introduce significant model errors (bias) that cannot be overcome by the complexity of a RNN
- I don't have any experiences doing it
- Higher risk of messing something up during deployment phase if some data are incomplete.

# Current progress – Last Updated April 2 2022 (20 days from competition due date)

# Current training architecture



# Current testing architecture

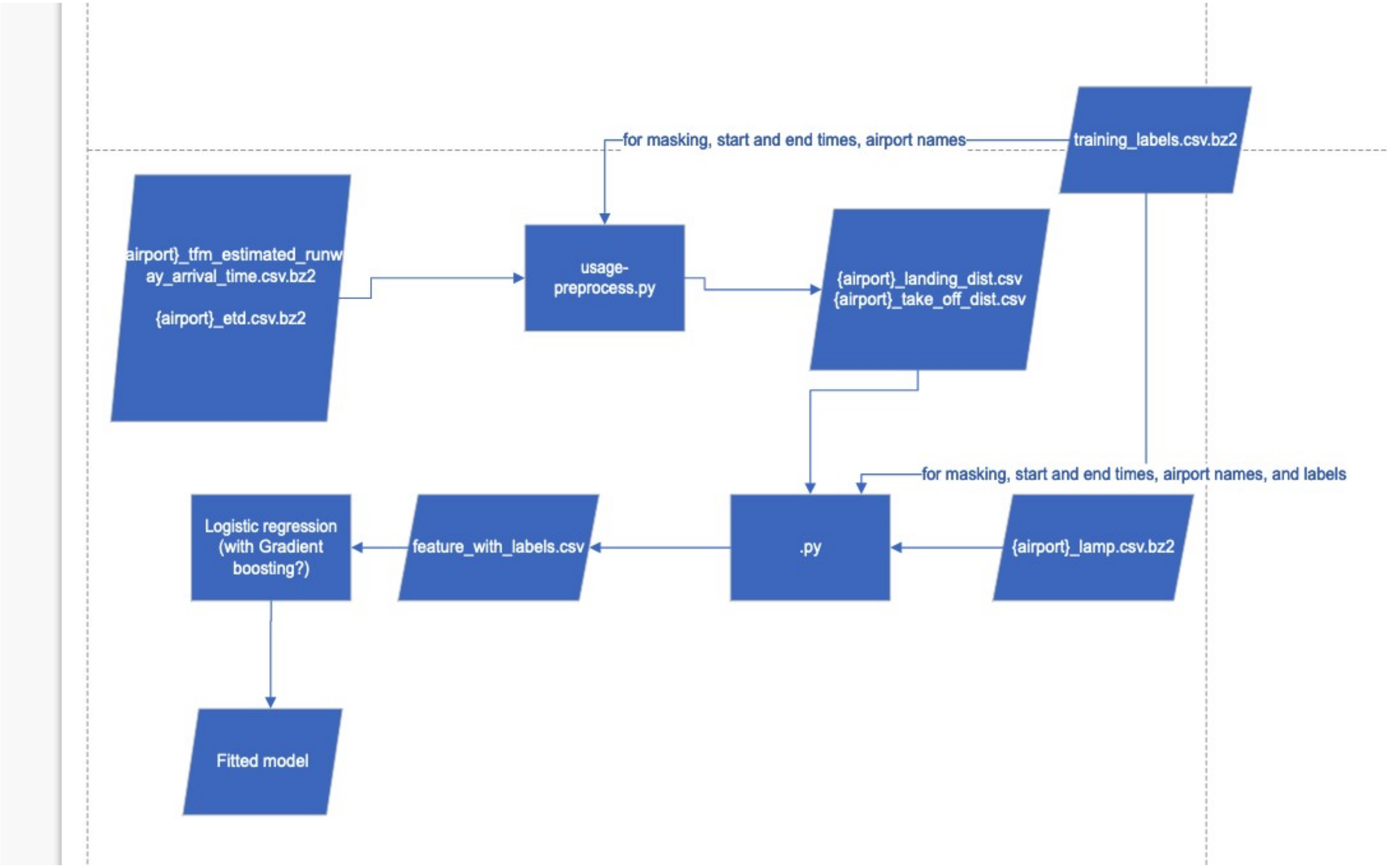


# Tasks – last updated April 2

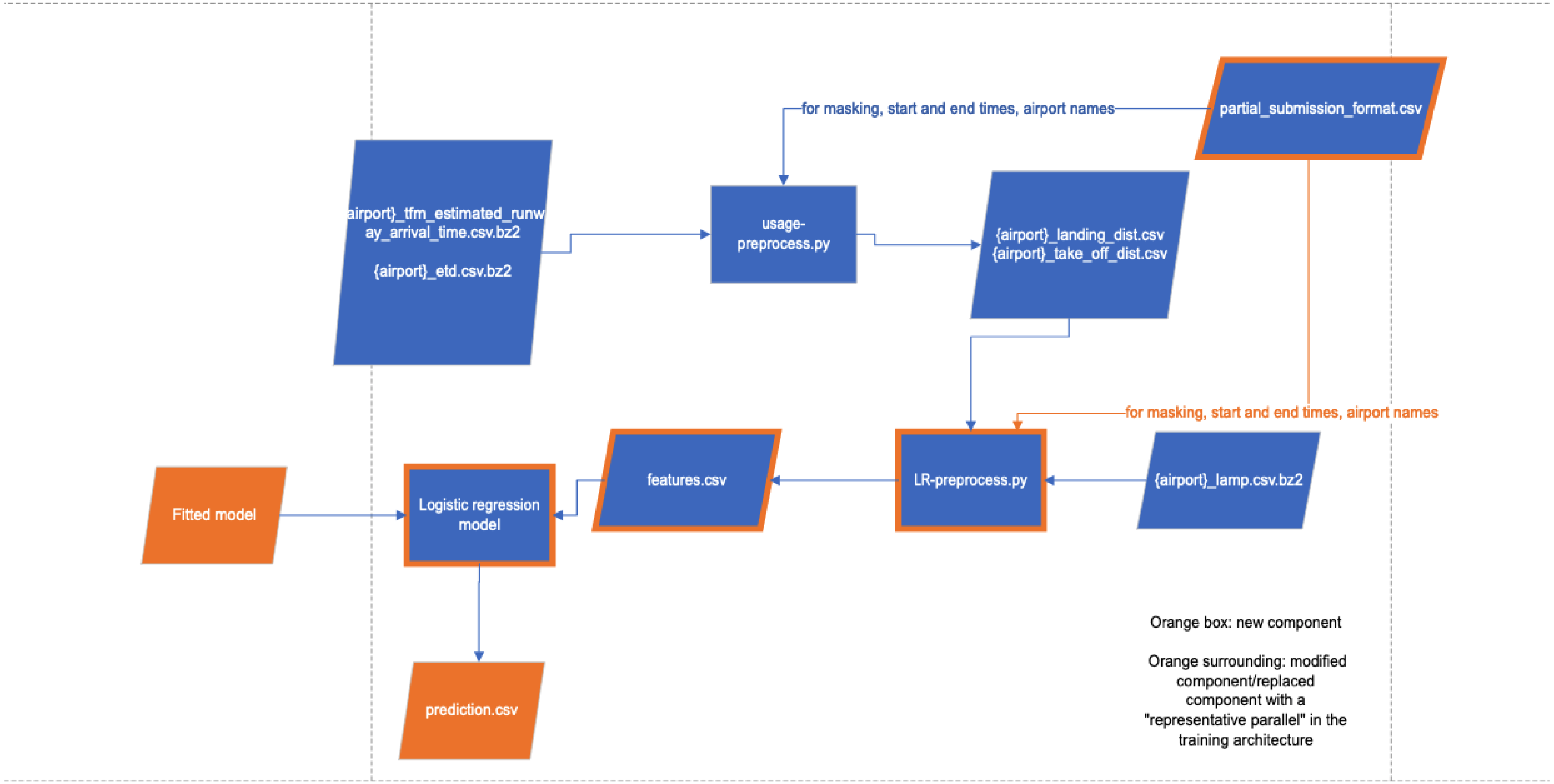
- Training (in progress):
  - Usage preprocess for aircraft departure and arrival data -- Complete
  - Preprocessing for Logistic Regression -- Complete (UPDATE in progress: implement Lyod White sin_cos approach for wind_direction angles: 0 – 360 degrees is not linear as 0 == 360. Map to 2D using sin(x) cos(x))
  - Logistic Regression with Sci-kit learn – in progress (testing+debugging)
  - XGBoosting – todo
    - Submit request for XGBoost python package to NASA -- todo
- Testing (todo)
  - Usage preprocess for aircraft departure and arrival – todo
  - Preprocess for model fitting – todo
  - Collect data and verification – todo
- Deploy (todo)
  - Ensure code runs on simulated bench -- todo
  - Ensure code runs despite potential specified data outages -- todo
  - Verify NASA accept pull request for XGBoost python package -- todo
  - Deploy (submit) to NASA for final verification – todo

# Current progress – Last Updated April 21 2022 (1 days from competition due date)

# Current training architecture



# Current testing architecture



# Tried

- Tested temporal decay (not very good
- Added term for classes not in training data

# Tasks – last updated April 2

- Training (in progress):
  - Usage preprocess for aircraft departure and arrival data -- Complete
  - Preprocessing for Logistic Regression -- Complete (Complete: implement Lyod White sin_cos approach for wind_direction angles: 0 – 360 degrees is not linear as 0 == 360. Map to 2D using sin(x) cos(x))
  - Logistic Regression with Sci-kit learn – complete (testing+debugging)
    - Standardize data to mean=1, variance=1 - complete
    - Upweight May and June datapoints during training ✉ $1/6^{th}$ data accounts 4/9 of the weights, the rest $5/6^{th}$ accounts 5/9 - complete
    - Hyperparameter training on temporal data, add term for classes not in training data - complete
  - XGBoosting – todo (prone to overfitting: discarded, use Logistic regression instead
    - Submit request for XGBoost python package to NASA – todo (used scikit learn gradient boosting instead)
- Testing (complete)
  - Usage preprocess for aircraft departure and arrival – complete
  - Preprocess for model fitting – complete
  - Collect data and verification – todo
- Deploy (complete)
  - Ensure code runs on simulated bench -- complete
  - Ensure code runs despite potential specified data outages -- complete
  - Verify NASA accept pull request for XGBoost python package – disgarded
  - Deploy (submit) to NASA for final verification – complete

# Validation results (Logistic Regression) – temporally split off the last 1/4th of the dataset

- Baseline (C=2) mean score = .1
- Baseline (C=1) mean score = .1
- Baseline (C=.5) mean score: 0.09846897685474114
- Baseline (C=.2) mean score = 0.09629876618410152
- C=.1 mean score: 0.09482931040417467
- C=.05 mean score: 0.09351667526087541
- C=.025 mean score: 0.0923271718253887
- C=.0125 mean score: 0.09138862781015332
- C=.005 mean score .0090
- C=.0025 mean score: 0.09058057530870296
- **C = .001 mean score: 0.09102370100469091 – final configuration used for competition**
- C = .0001 mean score: 0.09853771147536321