Model documentation and write-up

1.1. Who are you (mini bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

I am an incoming second year graduate student at New York University. I'm working towards a BS in Data Science. I'm very interested in all things data science and wanted to get involved in the competition in order to test my skills.

1.2. What motivated you to compete in this challenge?

There were 3 reasons motivating our interest:

- Access to real-world data from NASA
- Relevance of the problem being tackled
- Opportunity to showcase and practice some of the techniques learned in graduate school

1.3. High level summary of your approach: what did you do and why?

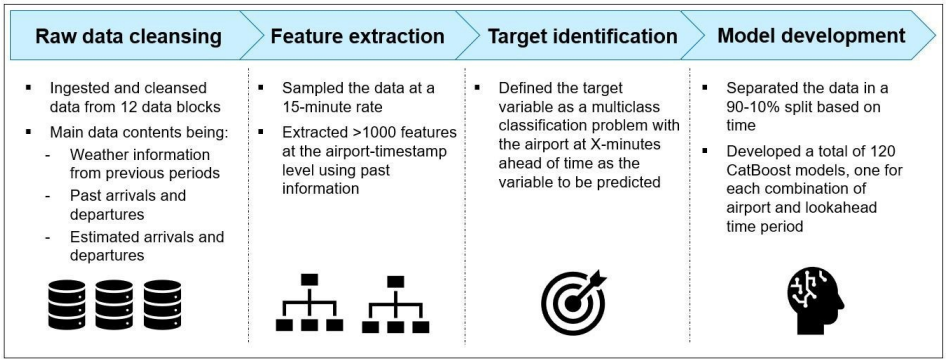
We cleansed the raw data, created a master table at an airport-timestamp period with a sampling rate of 15 minutes between each observation and appended past features from each of the 12 raw data blocks provided. That is, each row of the master table contains data for a given airport-timestamp that was available before the given timestamp (past information). Next, we added the targets to be modelled, in this case the airport configuration 30 minutes ahead of time, 60 minutes … and for each lookahead period. Next, we built a total of 120 CatBoost multiclass classifiers that allowed us to predict the likelihood of each configuration at an airport-lookahead level, i.e. 10 airports x 12 lookahead periods = 120 models.

Finally, we created a set of functionalities that allowed these models to be consumed by the DrivenData runtime environment in order to retrieve live predictions from the previously developed models.

We followed this approach as it is the typical data science pipeline of cleansing, feature extraction, train / test split and modelling.

1.4. Do you have any useful charts, graphs, or visualizations from the process?

Yes, a high-level process flow of our solution can be seen below:



1.5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

The three of the most impactful parts of our code are (1) the master table creation, (2) model training and (3) prediction serving functionalities. They are attached as screenshots and discussed below, and the code can be found in the zipped repository attached to the submission.

Master table creation: This function is called when the main.py script from code/ is executed and follows a sequence of steps from data loading to feature extraction and target appending to end up with a 130Mb master table consisting of a combination of airport-timestamp in each row with past information (features) and future information (targets) for the given airport and timestamp

```
def CreateMaster(data_path: str, airports, start_time: str, end_time: str, with_targets=False) -> pd.DataFrame:
    """
    Loads all the raw tables from start_time to end_time into a dictionary
    and sequentially extracts features for each information block merging it into
    a combined master table at the airport-timestamp level with a 15minute aggregation level

    :param str data_path: Parent directory where the data is stored
    :param List[str] airports: List indicating which airports to create the master table for
    :param str start_time: Timestamp to read from
    :param str end_time: Timestamp to read up to
    :param Bool with_targets: Bool indicating whether to include the targets - only for training

    :return pd.Dataframe master_table: Dataframe at an airport-timestamp level with all the relevant features
    """

    # Load raw data and store it in a dictionary that maps airport + key -> pd.DataFrame
    raw_data = LoadRawData(data_path=data_path, airports=airports, start_time=start_time, end_time=end_time)

    # Create cross join of all the dates between start_time and end_time at a 15min frequency
    master_table = CrossJoinDatesAirports(airports=airports, start_time=start_time, end_time=end_time)

    # Extract features for the selected data blocks and append them to the master table
    master_table = ExtractAirportconfigFeatures(master_table, raw_data['airport_config'])
    master_table = ExtractRunwayArrivalFeatures(master_table, raw_data['arrival_runway'])
    master_table = ExtractRunwayDepartureFeatures(master_table, raw_data['departure_runway'])
    master_table = ExtractLampFeatures(master_table, raw_data['lamp'])
    master_table = ExtractETDFeatures(master_table, raw_data['etd'])
    master_table = ExtractERAFeatures(master_table, raw_data['tfm_estimated_runway_arrival_time'])
    master_table = ExtractERAFeatures(master_table, raw_data['tbfm_scheduled_runway_arrival_time'],
                                      column='scheduled_runway_arrival_time')
    master_table = ExtractGufiTimestampFeatures(master_table, raw_data['first_position'], 'first_position')
    master_table = ExtractGufiTimestampFeatures(master_table, raw_data['mfs_runway_arrival_time'], 'mfs_runway_arrival_time')
    master_table = ExtractGufiTimestampFeatures(master_table, raw_data['mfs_runway_departure_time'], 'mfs_runway_departure_time')
    master_table = ExtractGufiTimestampFeatures(master_table, raw_data['mfs_stand_arrival_time'], 'mfs_stand_arrival_time')
    master_table = ExtractGufiTimestampFeatures(master_table, raw_data['mfs_stand_departure_time'], 'mfs_stand_departure_time')

    # Adjust master table in order not to have errors in edge cases in prediction time
    master_table = Adjust(master_table)

    # In case we want the master table for training we include the targets
    if with_targets:
        master_table = AddTargets(master_table)

    return master_table
```

Model training: Once the master table has been created, we iterate through each combination of possible airports and targets (i.e. lookahead periods) to build a multiclass classifier with the CatBoost library. Furthermore, in order to find the best set of parameters we allow the eta and depth to change for each of the 120 models to find the best possible combination of these 2 parameters.

```
# Build catboost models at the timestamp-airport level for each airport and lookahead
for airport in AIRPORTS:
    for target in [f'target_{l}' for l in LOOKAHEADS]:

        # Filter to stick only with relevant rows
        current = master_table[(master_table[target].isna() == False) &
                               (master_table.airport == airport) &
                               (master_table.day.isin(
                                   master_table[master_table.in_test.isna() == False]['day'].unique()) == False)
                               ].reset_index(drop=True)

        # Define size of train vs validation sets
        n = current.shape[0]
        start_test = int(n * 0.95)
        end_test = int(n)

        # Split train and validation sets
        X_train = current[features].iloc[lambda x: (x.index < start_test) | (x.index > end_test)]
        y_train = current[target][lambda x: (x.index < start_test) | (x.index > end_test)]
        X_val = current[features].iloc[start_test:end_test]
        y_val = current[target][start_test:end_test]

        # Initialize a big score for the loss
        best_score = 200

        # Iterate for different values of eta and depth and stick with the best
        for eta in [0.005, 0.01, 0.02, 0.03, 0.05]:
            for depth in [3, 4, 5, 6, 7]:

                # Initialize CatBoostClassifier
                model = CatBoostClassifier(eta=eta,
                                           n_estimators=6000,
                                           task_type='CPU',
                                           thread_count=-1,
                                           depth=depth,
                                           l2_leaf_reg=20,
                                           min_data_in_leaf=1000,
                                           grow_policy='Lossguide',
                                           max_leaves=11,
                                           has_time=True,
                                           random_seed=4,
                                           loss_function='MultiClass',
                                           boosting_type='Plain',
                                           class_names=current[target].unique())
```

Submission to Driven Data's environment: Finally, once the feature extraction process is done and the models trained, the code below allows us to retrieve live predictions, i.e. given a batch of new raw data for a particular timestamp we iterate for every airport and lookahead period to generate the predictions for the specified timestamp.

```
# Initialize empty placeholder to store all predictions
predictions = pd.DataFrame()

# Iterate through each airport to append the predictions
for airport in to_submit['airport'].unique():
    to_predict = master_table[master_table['airport'] == airport].copy()

    for l in LOOKAHEADS:

        current_model = CatBoostClassifier()
        current_model.load_model(os.path.join(models_path, 'version_7', f'model_{airport}_target_{l}'))
        classes = current_model.classes_
        probabilities = current_model.predict_proba(to_predict[current_model.feature_names_])
        current_predictions = to_predict[['airport', 'timestamp']].copy()
        current_predictions['lookahead'] = l

        # Add the predictions for every class as a new column
        for i in range(len(classes)):
            current_predictions[classes[i]] = probabilities[:, i]

        # Stack the lookahead columns into a single column
        current_predictions = current_predictions.set_index(
            ['airport', 'timestamp', 'lookahead']).stack().reset_index()
        current_predictions.columns = ['airport', 'timestamp', 'lookahead', 'config', 'active']

        # Concatenate current predictions with the previously computed ones
        predictions = pd.concat([predictions, current_predictions])

predictions = predictions.groupby(['airport', 'timestamp', 'lookahead', 'config']).first().reset_index()

to_submit = to_submit.merge(predictions, how='left', on=['airport', 'timestamp', 'lookahead', 'config'])

# Adjust submission in case there are additional configs in submission format file
to_submit['active'] = to_submit['active'].fillna(0)

cat_sum = to_submit.groupby(['airport', 'timestamp', 'lookahead']).active.sum().reset_index()
cat_sum.columns = ['airport', 'timestamp', 'lookahead', 'cat_sum']

to_submit = to_submit.merge(cat_sum, how = 'left', on = ['airport', 'timestamp', 'lookahead'])
to_submit['active'] = to_submit['active'] / to_submit['cat_sum']
to_submit.drop(columns = 'cat_sum', inplace = True)
```

1.6. Please provide the machine specs and time you used to run your model.

- **CPU (model):** Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz 2.11 GHz
- **GPU (model or N/A):** Tesla P100-PCIE-16GB
- **Memory (GB):** 16Gb of GPU and 32Gb in CPU
- **OS:** Windows 10
- **Train duration:** 45min in GPU, 15h in CPU
- **Inference duration:** 4h in DrivenData's environment for 1 week of data (max was capped to 8 by organizers)

1.7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

The process to replicate the data pipeline and model pipeline is detailed in the README.md within the repository. The only two manual points to take into account are:

1. Data in the *data_path* should follow the specified structure, i.e. one folder for each airport and within the folder the .csv raw files (i.e. not the binary files as provided by the competition)
2. In order to speed up training, before executing code/main.py it is advised to change the CPU to GPU setting in line 79 of code/main.py

1.8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No, all the tools utilized are detailed in the requirements.txt attachment within the repository.

1.9. How did you evaluate performance of the model other than the provided metric, if at all?

Performance was evaluated using the aggregated log-loss in a held out 10% period of the provided training data, this allowed to benchmark our models, evaluate which features contributed and to fine-tune CatBoost's parameters.

1.10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

We tried different approaches that did not make it to the final solution, among them range:

1. Feeding past predictions to future models, i.e. including as a variable for the 60minutes lookahead period the 30min prediction – did not improve the score
2. Trying a global model, i.e. one in which airport was a feature and all the other variables were considered as a cross-section of our population – This made the training process too expensive computationally and was error prone in case additional airports were added / dropped in the productization stage
3. Several different dimensionality reduction techniques which did not add any predictive power

1.11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

We would try to understand the geometry of the runways, i.e. how they are set up within the airport. IN our approach we treated the codes as independent representations of an airport's configuration, but there is geometry in the angles / cross relations. In that sense,

we could have incorporated more well-curated features such as “typical transition” features detailing which state is more likely to come given the previous sequence of steps treating the configurations as a time series more than separate classes.