Prize recipient solution documentation guide

This document details the information required to submit your Phase 1 solution materials.

I. **Code submission and result reproducibility**
You package up and send us your code, documentation of dependencies, and any other assets you used. We review your package and make sure that it works and that we can fully reproduce the workflow from raw data to a submission comparable to your best submission. See the Competition prize winner submission template for more guidance on packing up and organizing your code (however, please note that the template's write-up questions differ from the questions for this challenge).

II. **Basic information for winner announcement**
Provide your preferred information for use in announcing the winners of the competition.

III. **Model documentation and write-up**
You write up answers to our questionnaire, providing important context and documentation so that the beneficiary and the community get the most out of your work.

I. Code submission and result reproducibility

You will need to submit a compressed archive of your code. You don't need to include the raw data we provided, but everything else should be included and organized. If the files are too large to be e-mailed, a Google Drive or Dropbox share (or other comparable method of transferring data) works.

Note: *Please follow these instructions carefully*. The spirit and purpose of the competition (and the reason for offering prizes) is to give our beneficiary organizations the best possible solution *along with working code they can actually use*. In accordance with the competition rules, if we can't get your code working and reproduce your results with a reasonable effort, or if your entry is too disorganized to be practically usable, then your entry may be disqualified!

The overall concept is to **set up this archive as if it were a finished open source project**, with clear instructions, dependencies and requirements identified, and code structured logically with an obvious point of entry. Check out the competition prize winner README template to get started. We also have a data science project template which may be helpful.

At a minimum, your solution must include **an extremely clear README** that details all of the steps necessary to get to your submission from a fresh system with no dependencies (e.g., a brand new Linux, Mac OS X, or Windows installation depending on what environment you choose to develop under) and no other data aside from the raw data you downloaded from us.

This will probably entail the following:

- Necessary tools and requirements (e.g. "You must install Word2Vec 0.1c" or "Install the required Python packages in `requirements.txt`").
    - **All requirements should be clearly documented**, for instance in either a requirements.txt file with versions specified or environment.yml file.
- The series of commands, in order, that would get a reasonably experienced and savvy user from your code to a finished submission.
    - **Ideally, you will have a main point of entry to your code** such as an executable script that runs all steps of the pipeline in a deterministic fashion. A well-constructed Jupyter notebook or R script meets this standard.
    - **The next best thing is a list of specific, manual steps** outlining what to do. For example, "First, open Word2Vec and set these parameters. [...] Take the output file and run the script `src/make_preds.R` with the following parameters [...]". *(The limitations of this approach should be clear to any experienced data scientist!)*
- **Make sure to provide access to all trained model weights necessary to generate predictions from new data samples** without needing to retrain your model from scratch. Note that model weights can be contained in your archive or shared via a cloud storage service. The solution should provide clear instructions to perform inference on a new data point, whether or not it is included in the test set.
- Any other instructions necessary to end up with your winning submission file (or comparable — we understand that certain parts of model fitting are stochastic and won't result in exactly the same parameters every time).

II. Basic information for winner announcement

Please provide your preferred information for use in announcing the winners of the competition.

Top row: Kyler Robison, Daniil Filienko, Yudong Lin, Trevor Tomlin
Bottom row: Anderson Nascimento, Sikha Pentyala, Martine De Cock

- Name (first and last name or first name and last initial): **Daniil Filienko**
- Hometown: Federal Way, WA
- Social handle or URL (optional):
  https://www.linkedin.com/in/daniil-filienko-800160215/

- Name (first and last name or first name and last initial): **Yudong Lin**
- Hometown: Renton, WA
- Social handle or URL (optional):
  https://www.linkedin.com/in/yudonglin99

- Name (first and last name or first name and last initial): **Trevor Tomlin**
- Hometown: Lacey, WA
- Social handle or URL (optional):
  https://www.linkedin.com/in/trevor-tomlin

- Name (first and last name or first name and last initial): **Kyler Robison**
- Hometown: Spanaway, WA
- Social handle or URL (optional):
  https://www.linkedin.com/in/kyler-robison/

- Name (first and last name or first name and last initial): **Sikha Pentyala**
- Hometown: Seattle, WA
- Social handle or URL (optional):
  https://sikhapentyala.github.io/
  https://www.linkedin.com/in/sikhapentyala/

- Name (first and last name or first name and last initial): **Anderson Nascimento**
- Hometown: University Place, WA
- Social handle or URL (optional):
  http://www.anderson-nascimento.org/
  https://www.linkedin.com/in/anderson-nascimento-43a4852

- Name (first and last name or first name and last initial): **Martine De Cock**
- Hometown: Bellevue, WA
- Social handle or URL (optional):
  https://faculty.washington.edu/mdecock/
  https://www.linkedin.com/in/decockmartine/

III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

- 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.

  We are a team of computer science faculty and students at the School of Engineering and Technology, University of Washington Tacoma.

  Dr. Martine De Cock is a Professor with expertise in machine learning. **Dr. Anderson Nascimento** is an Associate Professor with extensive background in information theory and cryptography. Together they lead a research group in privacy-preserving machine learning (PPML). Their group has a proven track record in privacy-preserving machine learning with 1st place positions in the iDASH2019 and 2021 competitions on secure genome analysis, being one of the winners of the 2023 U.S.-U.K. PETs Prize Challenge, a U.S. patent on cryptographically-secure machine learning, and publications in NeurIPS, ICML, PoPETS, IEEE Transactions on Dependable and Secure Computing, and IEEE Transactions on Information Forensics and Security, among others.

  Sikha Pentyala is a 3rd year PhD student in the PPML group. She has published her work on the use of secure multiparty computation, differential Privacy, and federated learning for trustworthy machine learning in ICML, ECIR, and workshops at NeurIPS and AAAI. She acted as the student lead in the PPML group's winning participation in the iDASH2021 and 2023 U.S.-U.K. PETs Prize challenges.

  Kyler Robison, Daniil Filienko, Yudong Lin, and Trevor Tomlin are senior undergraduate students in computer science. As a team, they have been awarded the 2023 Outstanding Undergraduate Research Award by the School of Engineering and Technology, UW Tacoma.

- What motivated you to compete in this challenge?

  The federated learning aspect. While there are many data science competitions, only very few are about privacy-preserving machine learning.

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

  Our solution for Phase 1 is a gradient boosted decision tree approach with a lot of feature engineering. We trained one LightGBM model per airport.

  We started with the suggested minutes until ETD (expected time of departure) feature and gradually built out our feature set, spending much of our effort on writing scripts to extract feature values from the train and test data in a reasonable amount of time. We used the LightGBM library for boosted decision trees because it has absolute error as a built-in objective function and it is much faster for model training than similar tree ensemble based

algorithms. We experimented with training one global model across all airports, as well as with training one local model per airport, and found that the latter resulted in lower MAE, so we adopted that as our final solution for Phase 1.

To process the large data files and train our models, we used Azure Virtual Machines with a substantial amount of memory provided through the UW Azure Cloud Computing Credits for Research program.

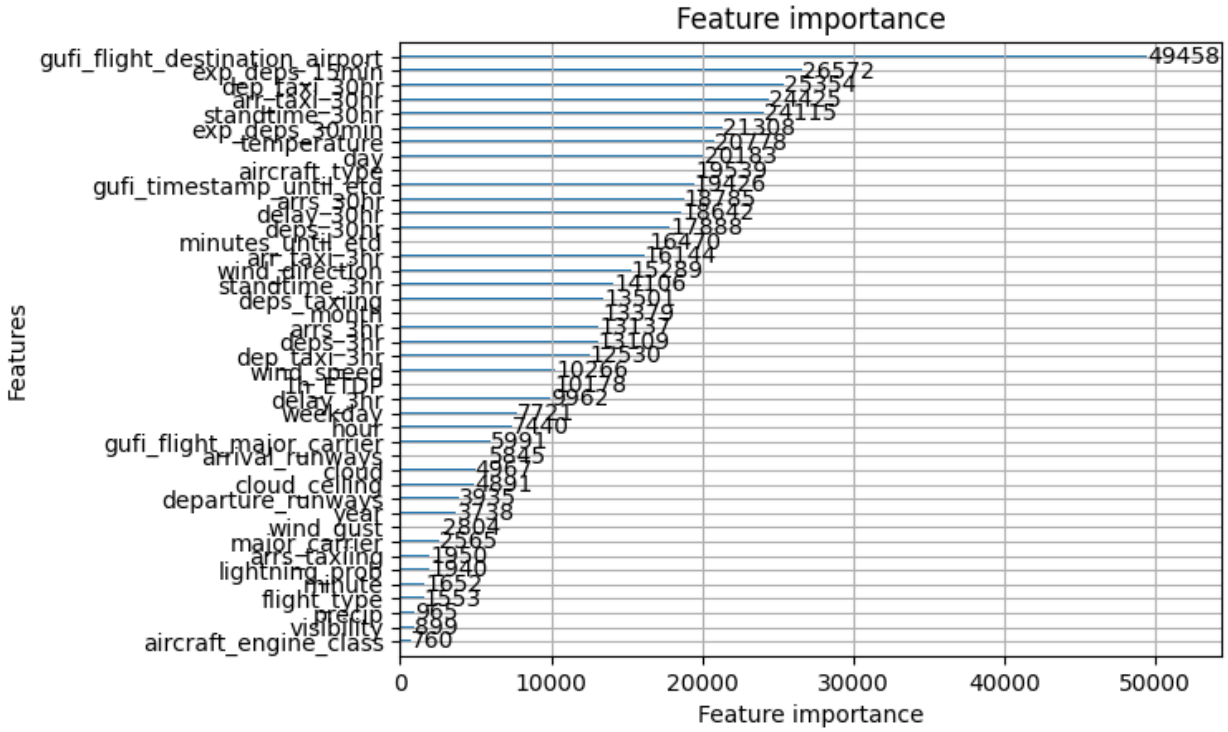
- How did knowing that you might need to federate this model change your approach to this challenge problem?

  We decided to spend most of our effort in Phase 1 on understanding the data and constructing features. We chose to work with gradient boosted decision trees because they are easier to train than neural networks on tabular data (faster and with less need for hyperparameter tuning). We initially assumed that the clients in the federation would be airports, which prompted us to train one model per airport.

  We understood from the beginning that creating a federated version of our approach in a short time span may require us to switch from a tree ensem to a neural network based approach (multilayer perceptron) because most of the literature and state-of-the-art libraries for federated learning are neural network based. The feature set that we have identified during Phase 1 can serve as a starting point, and we welcome suggestions for additional useful features that can further improve the utility.

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

Feature importance graph



The above feature importance graph for airport KMEM illustrates that for the trained LightGBM model for this airport, the top 5 most valuable features, i.e. the features found to be of highest value for determining the pushback time, are:

- gufi_flight_destination_airport: destination airport inferred from flight's GUFI label,
- exp_deps_15min: number of expected departures within the next 15 minutes,
- dep_taxi_30hr: average difference in time between actual departure and actual pushback for the preceding 30 hours at the analyzed airport
- arr_taxi_30hr: average amount of time that arrivals took to taxi from the runway to their gate in the preceding 30 hours
- standtime_30hr: average time between first position and pushback for all arrival flights at this airport in the past 30 hours

MAE results per airport

Apr 14 Submission	MAE
LEADERBOARD	11.1046
SMOKE TEST	6.35
CUMULATIVE	11.1104
KATL	8.5079
KCLT	9.5547
KDEN	12.1308
KDFW	12.4029
KJFK	11.8839
KMEM	17.6493
KMIA	12.4863
KORD	11.6937
KPHX	9.6316
KSEA	7.6478

To obtain the MAE results in the table above, we split all the labeled instances from the prescreened arena into a training dataset and a validation dataset. For the validation datasets, we used all the data with GUFIs that were part of the validation dataset in the open arena; all remaining instances formed our training dataset. We then used the training data to train a so-called “local model” for each airport, and measured the MAE of each local model on the validation data of the airport that it was trained for. In order to obtain the cumulative MAE, we simply concatenated the lists of predicted labels and lists of corresponding true values, and then evaluated the accuracy of the concatenated series.

KMEM stands out as the airport with the highest MAE. We suspect that this may be due to its unique nature of being a busy cargo airport with a high variation in types of flights hosted.

We observed that cumulative MAE results that we obtained on the leaderboard in the prescreened arena were typically slightly better than the MAE obtained with our own evaluation, probably due to the larger amount of data on which the submission models were trained (train + validation datasets were used for training the models for leaderboard submissions).

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

```
for airport in airports:
...
    y_pred = gbm.predict(X_test,num_iteration=gbm.best_iteration_)

    print("Finished training")
    print(f"MAE on {airport} test data: {mean_absolute_error(y_test,
      y_pred):.4f}\n")
    # appending the predictions and test to a single datasets to evaluate
      overall performance
    y_tests = np.concatenate((y_tests, y_test))
    y_preds = np.concatenate((y_preds, y_pred))

    plotImp(gbm,X_test,airport=airport)
```

The code snippet above was part of the evaluation script we used for determining the local and global accuracy of trained models. Each model was trained on training data from a given airport, obtained as a pandas dataframe from a training csv file, in the for-loop, followed by being evaluated on the validation dataset, which we created using GUFIs obtained from the open arena validation dataset file. The resulting predicted labels and the expected [true] labels then would be added to 2 cumulative Series containing the predicted/true labels from previous airports, that would later be compared to produce a weighted average of individual airport's MAEs, with the weights accounting for the ratio of the number of validation cases from a particular airport present in the cumulative validation dataset.

```
unique = defaultdict(set)

Encoder = partial(OrdinalEncoder, handle_unknown="use_encoded_value", unknown_value=-1)
encoders = defaultdict(Encoder)

for airport in airports:
    print(f"Processing Airport {airport}")
    data = pd.read_csv(f"data_apr16/tables/full_tables/{airport}_full.csv").sort_values("timestamp")
    data["precip"] = data["precip"].astype(str)
    data["isdeparture"] = data["isdeparture"].astype(str)

    for column in encoded_columns:
        unique[column].update(data[column].unique())

for column in encoded_columns:
    encoders[column].fit(np.array(list(unique[column])).reshape(-1, 1))

with open("encoders.pickle", "wb") as f:
    pickle.dump(encoders, f)
```

The code snippet above shows how the ordinal encoders were fitted when we were training the model using the entire dataset. We store a dictionary where the key is the column and the value is an encoder that is fitted on that column using all airports. Sets are used to keep track of the unique values that appear across airports. The encoders will use the value -1 when they run into a value in the test set that they have never seen before. The dictionaries are pickled at the end for use in the submission script.

```
def _process_timestamp(now: pd.Timestamp, flights: pd.DataFrame,
     data_tables: dict[str, pd.DataFrame]) -> pd.DataFrame:
    # subset table to only contain flights for the current timestamp
    filtered_table: pd.DataFrame = flights.loc[flights.timestamp ==
     now].reset_index(drop=True)

    # filters the data tables to only include data from past 30 hours,
     this call can be omitted in a submission script
    data_tables = filter_tables(now, data_tables)
    # get the latest ETD for each flight
    latest_etd: pd.DataFrame = data_tables["etd"].groupby("gufi").last()
    # add features
    filtered_table = add_etd(filtered_table, latest_etd)
    filtered_table = add_averages(now, filtered_table, latest_etd,
     data_tables)
    filtered_table = add_traffic(now, filtered_table, latest_etd,
     data_tables)
    filtered_table = add_config(filtered_table, data_tables)
    filtered_table = add_lamp(now, filtered_table, data_tables)

    return filtered_table
```

The code snippet above was part of the table generating algorithm and was valuable because of the highly modular yet efficient structure which we were able to achieve using that algorithm. We were able to utilize the Pandarallel library to improve the overall performance of the code, allowing us to support a final model with more than 30 derived features, including averages that require a high amount of memory and CPU resources to compute. In addition, having a single method responsible for the majority of the per-row operations allowed us to pre-filter the dataset to the previous 30 hrs from the timestamp and pass it to multiple functions, leading to a program that is modular and available for further modification, allowing to later include more derived features.

- Please provide the machine specs and time you used to run your model.
  - CPU (model): Intel Xeon Platinum 8168, 44 cores
  - GPU (model or N/A): N/A
  - Memory (GB): 352 GB
  - OS: x64 Ubuntu Server 20.04
  - Train duration: roughly 4 hours (3 hours for data preprocessing, 1 hour for encoders/models training)
  - Inference duration: ~ 20 minutes to classify 2,042,723 instances.

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

  Our code requires a substantial amount of memory to process the large data files. During development of our solution, we used Virtual Machines on Microsoft Azure (see above) as well as a University of Washington workstation which had similar configurations to the virtual environment on which the submission code was executed. The workstation had 250 GB of RAM and was sufficient for generating the tables with all of the extracted features that we used in our final submission, and for evaluating the submission test cases within a couple of hours. Based on our observations, the scripts need a minimum of 48 GB of RAM to execute, and ideally, you will need 128 GB of RAM to run the scripts without constantly triggering garbage collection which will significantly slow down the process.

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

  No.

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

  We split the data from the prescreened arena in a training dataset and a validation dataset. For the validation dataset we used all the data with GUFI's that were part of the validation dataset in the open arena. We used MAE as our main metric, and, to a lesser extent, RMSE.

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

  - Making a global model, trained on all airports, instead of locally trained models.
  - We had tried to utilize xgboost, but it does not perform as well as lightgbm.
  - Utilizing weather predictions for the next n hours instead of including exclusively current weather from LAMP.
  - Using ETD to predict the number of flights departing within the next n hours.