Group 4

Expected Travel Time

TUHH Technische Universität Hamburg

03.07.24

Benjamin Ko, Abeeku Ampah, Jorgos Drossinakis

Agenda:

1. Introduction to the ETT
2. Data Preparation and Cleaning
3. Selection of Machine Learning Models
4. Solution
5. Demo
6. SCRUM Evaluation

Introduction to ETT Problem and Tasks

TUHH

Problem: Harbors need to optimize their daily operations

Our Subtasks:

- Clean Automatic Identification System (AIS) Data
  - Felixstowe to Rotterdam
  - Rotterdam to Hamburg
- Feature Selection
- Model Selection
- Integration of decision-making system.

Data Preparation and Cleaning

Overview of AIS Data

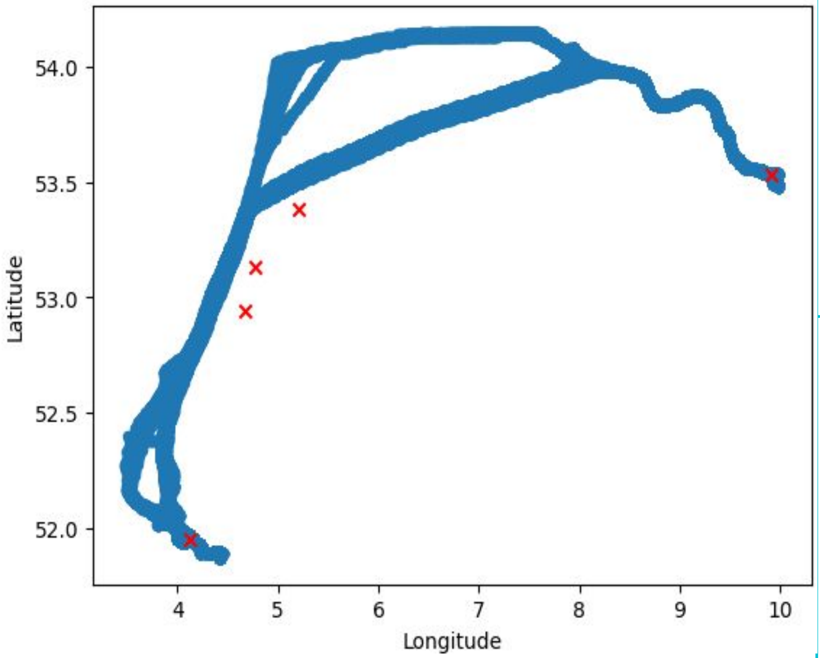
1. Static Data
   a. MMSI
   b. TripID
   c. Length, Draught etc

2. Dynamic Data
   a. Speed over Ground
   b. Course over Ground
   c. Vessel position etc

	TripID	MMSI	StartLatitude	StartLongitude	StartTime	EndLatitude	EndLongitude	EndTime	Sta
0	27811	477829700	51.95	4.03	'2016-01-24 12:50'	53.5	9.93	'2016-01-25 13:00'	RO
1	27811	477829700	51.95	4.03	'2016-01-24 12:50'	53.5	9.93	'2016-01-25 13:00'	RO
2	27811	477829700	51.95	4.03	'2016-01-24 12:50'	53.5	9.93	'2016-01-25 13:00'	RO
3	27811	477829700	51.95	4.03	'2016-01-24 12:50'	53.5	9.93	'2016-01-25 13:00'	RO
4	27811	477829700	51.95	4.03	'2016-01-24 12:50'	53.5	9.93	'2016-01-25 13:00'	RO

Data Preparation and Cleaning

- Checking for Duplicates
- Identifying how the data is distributed
- Handling of missing data
- Cross-checking data validity
- Addition of new features.



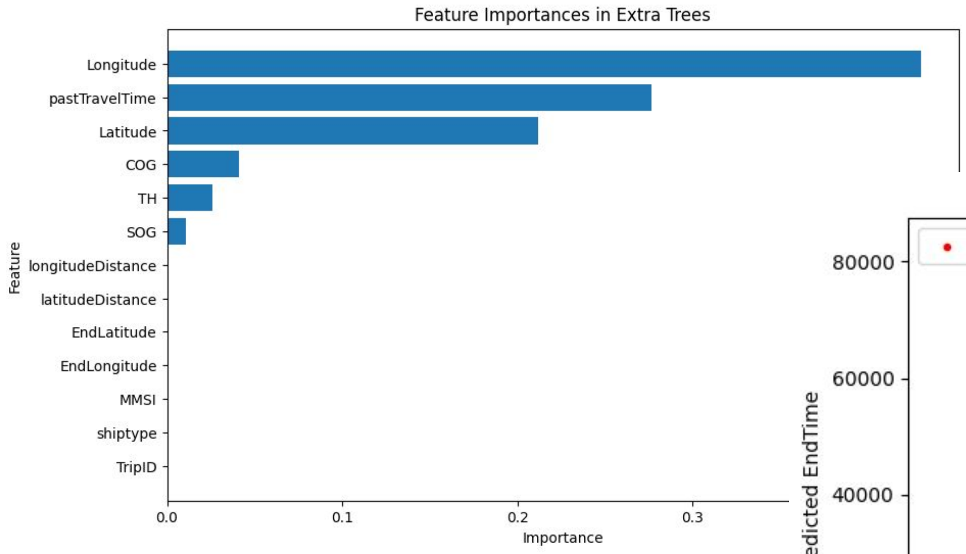
Trip: Rotterdam to Hamburg

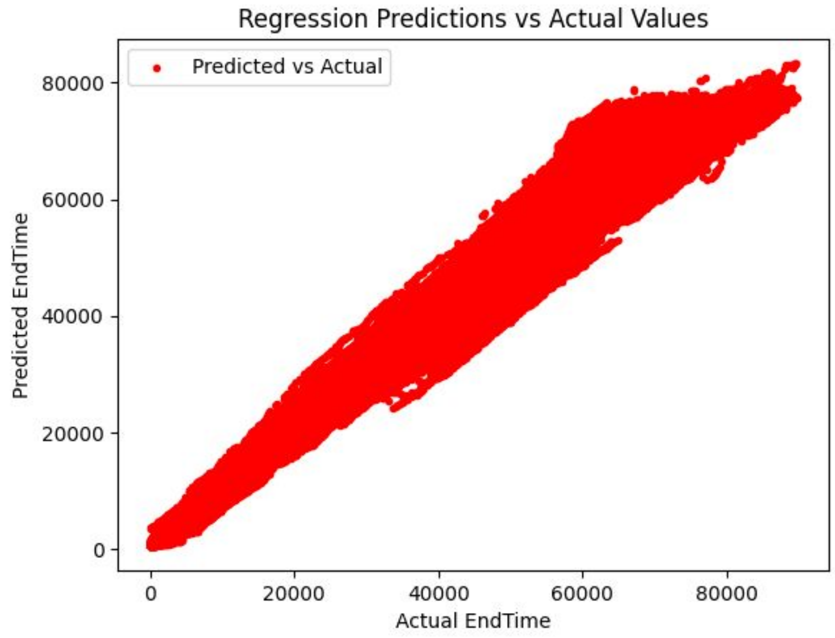
Selection of Machine Learning Methods

The following Machine Learning Methods were tested and experimented on:

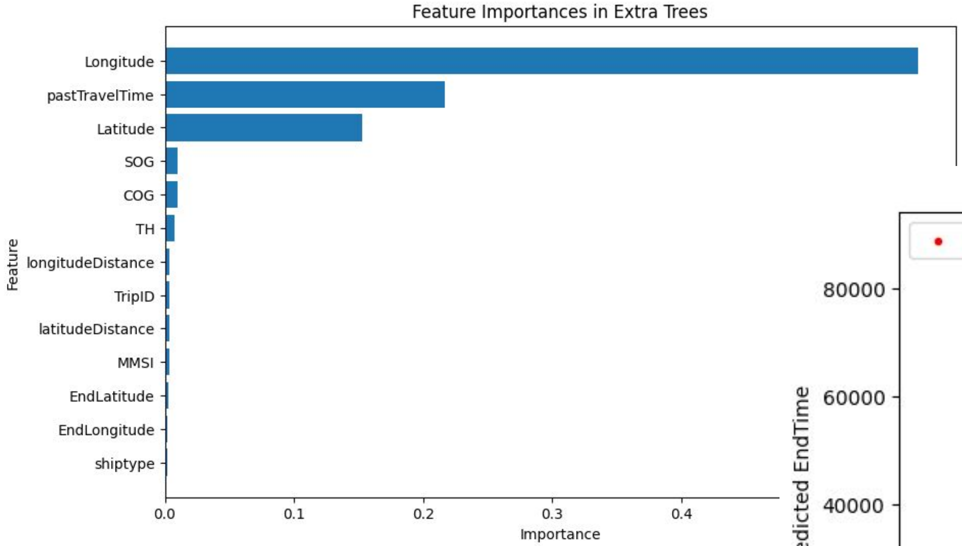
- Extra Trees
- Random Forest
- Adaptive Boosting
- XGBoost
- Neural Networks

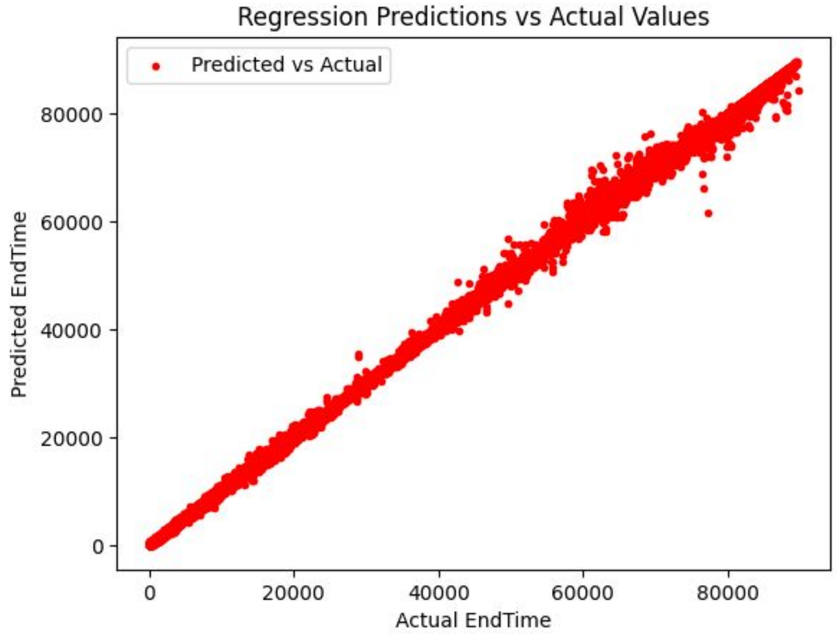
Selection of Machine Learning Methods: Extra Trees



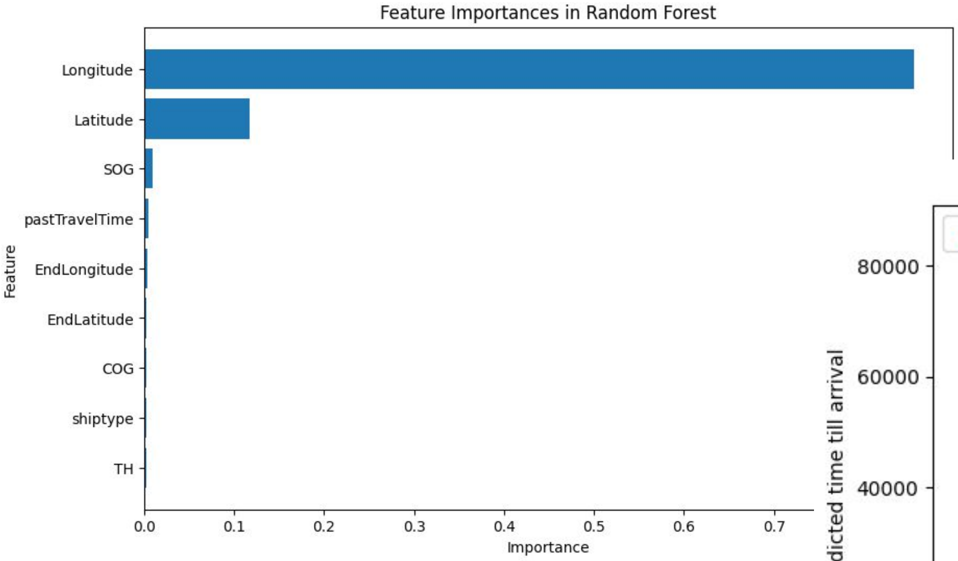


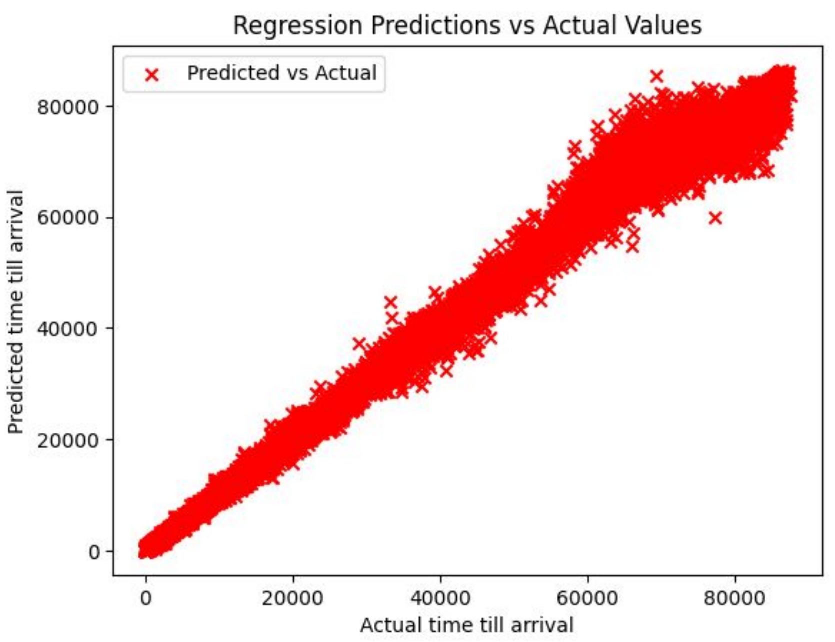
Selection of Machine Learning Methods: Extra Trees



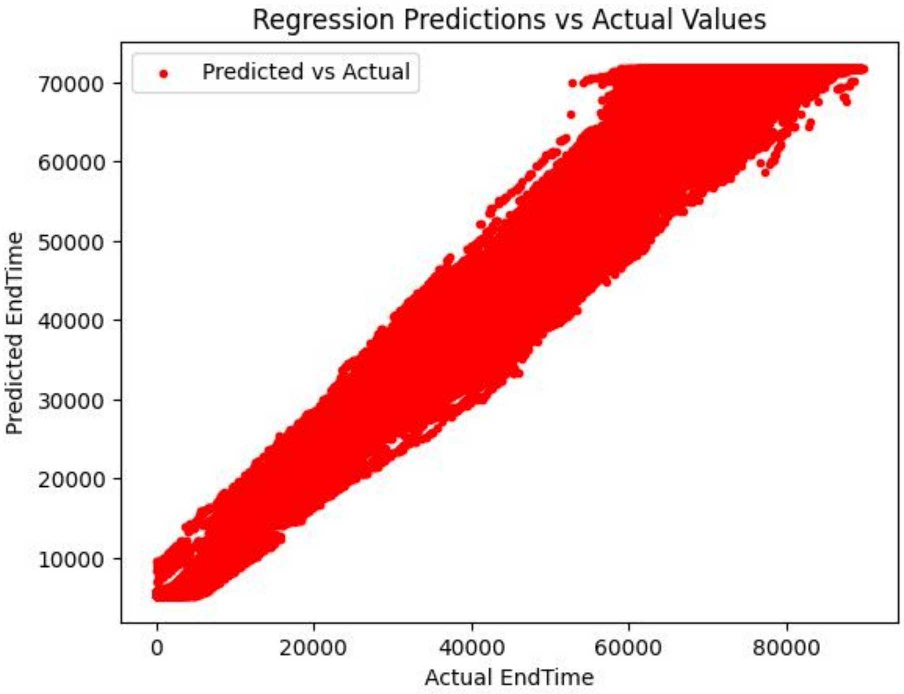


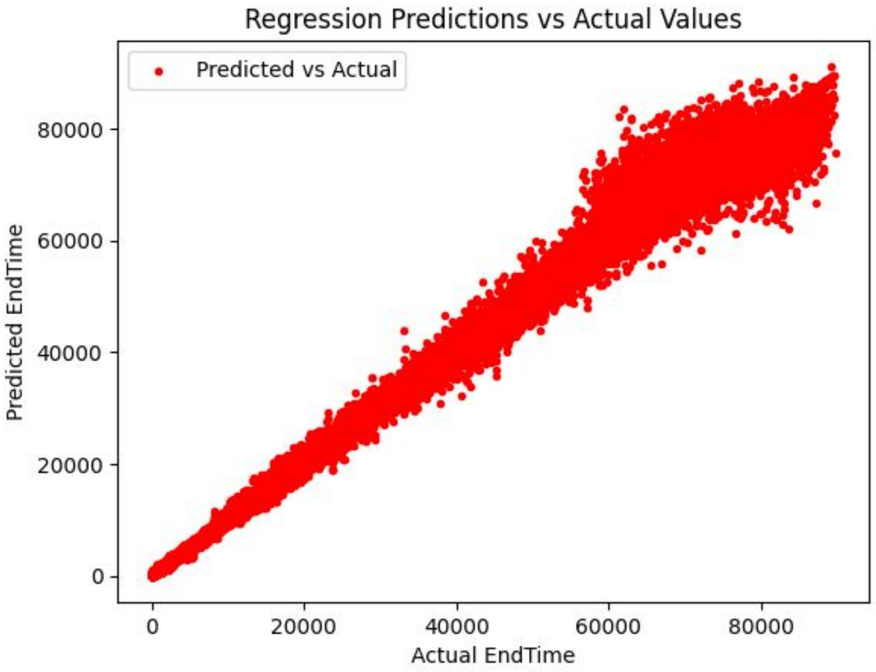
Selection of Machine Learning Methods: Random Forests





Selection of Machine Learning Methods: XGBoost

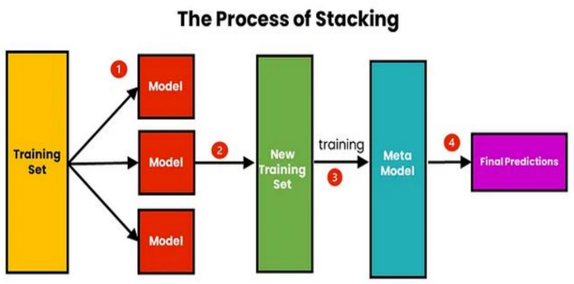




Development of final ETT Prediction (Broker Agent)

Broker Agent is implemented by Stacking method.

- Base models (predictor agents)
- Meta model selection
- Repeated K-Fold cross validation
- Final ETT Prediction with meta model
- Save models with joblib



Solution: Stacking


```
#use given csv data for the model
data = pd.read_csv("../../../data/RotHam_cleaned/rotterdam_hamburg_clean.csv", on_bad_lines="warn")
print('Data read done')

#specify test features
test_features = [ "COG", "TH", "shiptype", "EndLongitude", "EndLatitude", "pastTravelTime"]
print('Specify test features done')

#specify test and training sets
#Random state is used for initializing the internal random number generator, which will decide the splitting of data into train and test indices
y = data["timeTillArrival"]
X = data[["Latitude", "Longitude", "SOG"] + test_features]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print('Spliting data done')
```

Solution: Stacking


```
#Initializing models
model1 = RandomForestRegressor(n_estimators=100, max_depth=8, random_state=42)
model2 = ExtraTreesRegressor(n_estimators=450, min_samples_split=2, min_samples_leaf=2, max_depth=21, random_state=42)
model3 = xgb.XGBRegressor(n_estimators=500, learning_rate=0.05, max_depth=25, min_child_weight=3, subsample=0.8, colsample_bytree=0.8,

# Meta model
meta_model = ExtraTreesRegressor(n_estimators=400, min_samples_split=2, min_samples_leaf=2, max_depth=20, random_state=42)
print('Initializing models done')

#K-fold Cross-Validation: Divide dataset into k equally sized folds (subsets) to reduce variance and better utilize the data
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
print('K-Folds done')

#Arrays to store predictions
x1_train = np.zeros((X_train.shape[0],))
x2_train = np.zeros((X_train.shape[0],))
x3_train = np.zeros((X_train.shape[0],))

x1_test = np.zeros((X_test.shape[0], n_splits))
x2_test = np.zeros((X_test.shape[0], n_splits))
x3_test = np.zeros((X_test.shape[0], n_splits))
print('Initializing Arrays done')
```

Solution: Stacking

•

```
for fold_idx, (train_idx, valid_idx) in enumerate(kf.split(X_train)):
    #Training base models
    model1.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
    model2.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
    model3.fit(X_train.iloc[train_idx], y_train.iloc[train_idx])
    print(f'Fiting base models done for fold {fold_idx + 1}')

    #Making predictions
    x1_train[valid_idx] = model1.predict(X_train.iloc[valid_idx])
    x2_train[valid_idx] = model2.predict(X_train.iloc[valid_idx])
    x3_train[valid_idx] = model3.predict(X_train.iloc[valid_idx])
    print(f'Predictins base models done for fold {fold_idx + 1}')

    #Collecting test set predictions for averaging later
    x1_test[:, fold_idx] = model1.predict(X_test)
    x2_test[:, fold_idx] = model2.predict(X_test)
    x3_test[:, fold_idx] = model3.predict(X_test)
    print(f'Collecting test predictions done for fold {fold_idx + 1}')

#Average the test set predictions
x1_test = x1_test.mean(axis=1)
x2_test = x2_test.mean(axis=1)
x3_test = x3_test.mean(axis=1)
print('Averaging test set predictions done')

#Stack predictions as new feature set for meta model
X_train_meta = np.column_stack((x1_train, x2_train, x3_train))
X_test_meta = np.column_stack((x1_test, x2_test, x3_test))
print('Stack predictions as new features set for meta model done')
```

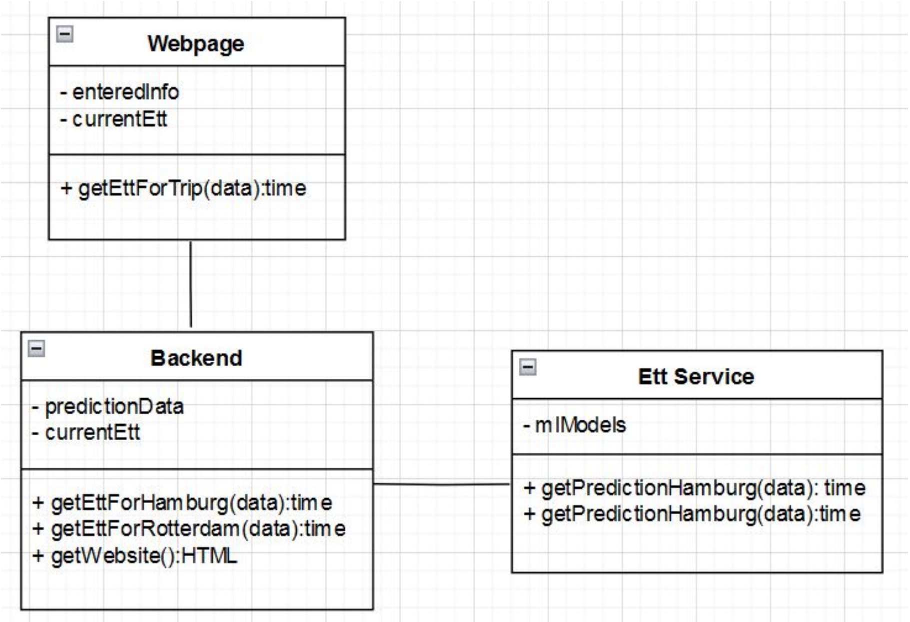
Solution: Stacking

```
#Train meta model
meta_model.fit(X_train_meta, y_train)
print('Train meta model done')

#Make final predictions
final_predictions = meta_model.predict(X_test_meta)
print('Final Prediction done')

#Evaluate the model (Perfect MAE = 0)
#Give out MAE of the prediction set compared to the test set
#MAE in minutes
mse = mean_absolute_error(y_test, final_predictions)
print('Mean absolute Error for Extra Trees: ' , mse/60)
rmse = np.sqrt(mean_squared_error(y_test, final_predictions))
print('Mean squared Error for Extra Trees: ', rmse)

#feature names for stacked features
stacked_features = ['x1_train', 'x2_train', 'x3_train']
importances = meta_model.feature_importances_
#features = X.columns
```

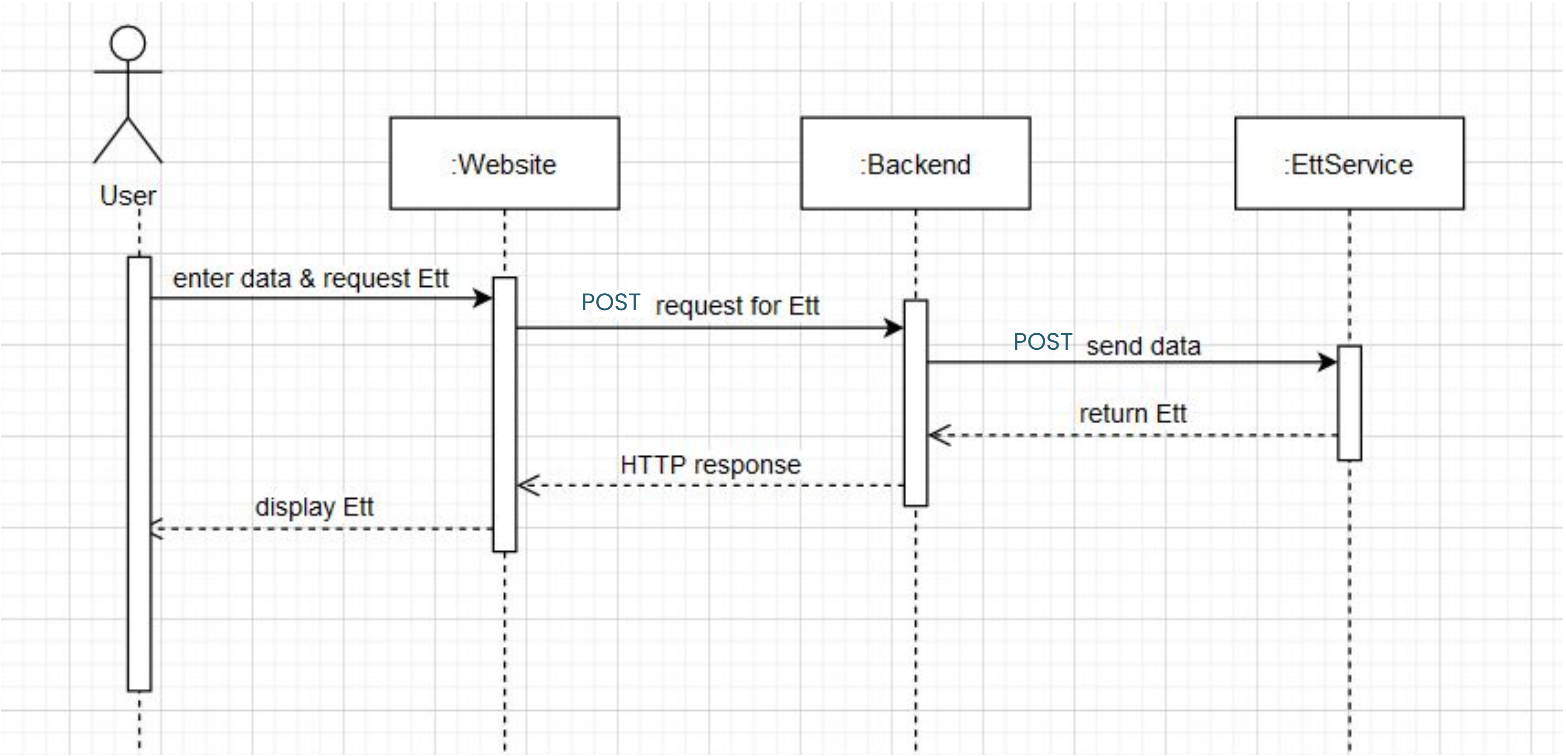
```
#Save the models with Joblib
dump(model1, 'model1.joblib', compress=3)
dump(model2, 'model2.joblib', compress=3)
dump(model3, 'model3.joblib', compress=3)
dump(meta_model, 'meta_model.joblib', compress=3)
```

Solution - UI

- Web based server-client architecture
- Frontend: HTML, CSS(Bootstrap), Javascript
- Backend: Node.js (Express)
- Flask in a docker container
- Joblib to load the models



Solution - Sequence diagram



Demonstration

SCRUM Evaluation

- We planned our time realistically
- Tasks were evenly distributed between us
- Bi-Weekly SCRUM Meetings helpful
- Used Google Sheets for planning

Thank you!

tuhh.de

TUHH
Technische Universität Hamburg