Background and Objectives

- A point of interest is a specific physical location, which someone may find interesting. Examples: Restaurants, retail and grocery stores, gas stations etc.
- Telenav provides GPS satellite navigation, local searches, entertainment for automotive navigation.
- Points of interest are provided to Telenav by different vendors. As a result, many provided points of interest are different in title, address, etc., but are duplicates, and vice versa
  POI 1: “Costco”, POI 2: “Costco Gas” FALSE (non-duplicates)
- Objective:
  o Develop and integrate into Telenav data processing pipeline a solution to find
  o Machine Learning model prediction accuracy above 95%
  o Points of interest are provided to Telenav by different vendors. As a result, many provided points of interest are different in title, address, etc., but are duplicates, and vice versa
  o Machine Learning model prediction accuracy above 95%
  o Java API library with a function entrypoint and Command Line Interface for analyzing large POI datasets

Pre-processing and Features

- The overall workflow involves three large steps. The first is data pre-processing to extract features such as title/address similarity, physical distance, category relationships, etc.
- The second is the prediction made by the machine learning model using the extracted features. The third is the hard-coded rulebook that overrules model decisions based on company policy.

Command Line Interface

- One of our supporting deliverables is to implement a command line interface for Telenav to perform offline testing
- CLI request a CSV input file with POI pairs and the given decision of being duplicates
- For each POI pair, CLI triggers a call to the Java API function to determine if the POIs are duplicates
- Outputs the accuracy of the model based on number of decisions made by the model that matches the given decision
- Outputs a text file with inference results for each API call named output.csv

Results

- Overall, we were able to meet and in fact exceed accuracy expectations.
- The final accuracy scores with all of the features and rulebook for the API library are:
  o Overall Accuracy = 98.3%
  o Positive Accuracy = 97.7%
  o Negative Accuracy = 98.7%
  o Model has a true confidence score based on new data, parameters, etc.
  o Improved on difficult edge cases

Machine Learning Research

- Since our training target is categories and the training data has been labeled, we use the classification model to train the data.
- After trying many different models, we finally settled on the Random Forest model and used ROC, AUC to validate and tune hyperparameters and accuracy.
- Once the Python model is trained, we convert it to a Java model to be used.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Total Accuracy Training</th>
<th>Validation</th>
<th>Positive Accuracy Training</th>
<th>Validation</th>
<th>Negative Accuracy Training</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>95.7%</td>
<td>95.6%</td>
<td>83.3%</td>
<td>82.1%</td>
<td>97.6%</td>
<td>97.6%</td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td>97.4%</td>
<td>96.1%</td>
<td>86.5%</td>
<td>82.4%</td>
<td>98.8%</td>
<td>98.1%</td>
<td></td>
</tr>
<tr>
<td>Decision Tree</td>
<td>96.1%</td>
<td>95.8%</td>
<td>84.4%</td>
<td>82.1%</td>
<td>97.9%</td>
<td>97.8%</td>
<td></td>
</tr>
<tr>
<td>Extra Tree</td>
<td>96.8%</td>
<td>96.0%</td>
<td>86.5%</td>
<td>81.5%</td>
<td>98.4%</td>
<td>98.0%</td>
<td></td>
</tr>
<tr>
<td>Bagging</td>
<td>95.2%</td>
<td>95.8%</td>
<td>96.5%</td>
<td>79.7%</td>
<td>99.6%</td>
<td>98.1%</td>
<td></td>
</tr>
<tr>
<td>Bagging + KNeighbors</td>
<td>94.9%</td>
<td>95.5%</td>
<td>89.3%</td>
<td>79.8%</td>
<td>98.5%</td>
<td>97.8%</td>
<td></td>
</tr>
<tr>
<td>Bagging + SVC</td>
<td>95.6%</td>
<td>95.5%</td>
<td>75.7%</td>
<td>77.7%</td>
<td>98.1%</td>
<td>98.1%</td>
<td></td>
</tr>
<tr>
<td>SGD</td>
<td>95.8%</td>
<td>97.7%</td>
<td>81.8%</td>
<td>80.1%</td>
<td>97.9%</td>
<td>97.9%</td>
<td></td>
</tr>
</tbody>
</table>

Java Library Implementation

- The Java API Library is the main deliverable for Telenav as it will serve as the basis for executing deduplication analysis at scale.
- The grouping function kicks off by first analyzing whether the POI objects contain any category pairs that are considered non-pairs by the built rulebook.
- If the rulebook passes, the function performs data normalization to prepare the POI attributes for feature analysis.
- A series of helper functions are called which calculate the feature parameter values used for the Random Forest model.
- The library will then return true or false for the POI comparison if the Random Forest model has a true confidence score of at least 50%.

Future Development Ideas

- Provided Telenav with Python Jupyter Notebook to further train the model based on new data, parameters, etc.
- Add new preprocessing steps to provide more insight to model for inference
- Analyze the text similarity scoring system for non-english native POI titles for improvement on difficult edge cases
- Allow more information from the POI attributes to either rule out or improve positive accuracy