Kaggle Airbnb New User Bookings

Group 9 Final Project Report

Shawn Ericksen
ericksen@uark.edu

Garrett Fulghum
gmfulghu@uark.edu

Wesley Parker
wgparker@uark.edu

ABSTRACT

The Airbnb New User Bookings from Kaggle is a dataset containing different parameters of user information collected anonymously. This means that the data is specific to Airbnb only, like the signup date and signup method.

At the time of writing, 1459 current scores are listed on the Kaggle leaderboard, with the highest being a score of 0.88697. From this, it was decided to compare the results of this project to that of the top 75 percent of the leaderboard.

To compare findings, SVC Classifier scored the highest at 0.8819, ranking at the top 80.9 percentile. XGB Classifier came in as a good second with a score of 0.88155, and its score is in the top 80.2 percentiles.

The average percentile with all three scores was 20.1%, which is a massive success with the classifiers chosen.

The average score of these three is 0.8726, which happens to be closer to the SVC model Classifier. For more information concerning the data and results, please see below.

KEYWORDS & SYMBOLS

Kaggle – Public repository for any dataset imaginable

Pandas – Library to load data into a usable database/data frame

Classifiers – algorithms to train data into a functional database for prediction and learning.

NDCG – Evaluation Metric used for scoring on the Kaggle Airbnb.

Logistic Regression – Classifier that uses logistic regression

XGBoost – Gradient boosting classifier

SVM/SVC – Support Vector Classifier

joblib – A set of tools to provide lightweight pipelining in Python

Intel Extension for Scikit-Learn – The library serves as a patch to scikit-learn, optimizing various models through built-in parallelization and hardware acceleration.

INTRODUCTION & MOTIVATION

At first glance, the goal was to use the Microsoft Malware Prediction dataset as it allowed for more complex algorithms to encode and utilize the data. However, after a week or two of testing, issues became apparent when dealing with the large dataset. Extra Libraries such as Dask and Dask-ML helped to implement out-of-memory data frames. While these memory improvements did help, they proved to be more complex than anticipated, and under the time constraint for this class, it was decided to switch to the Airbnb dataset over the MS Malware.

The Airbnb data consists of the date of the first booking, gender, age, signup method, signup flow, language, possible affiliation, country, and first browser. In addition, a log of users' session data includes action, action_type, action_description, device_type, and secs_elapsed. The goal is to predict in which country a new user will make their first booking.

At the time of this competition, the top teams were considered for an interview for the opportunity to join Airbnb's Data Science and Analytics team. The idea that groups or individuals could work together to create algorithms to help predict new user booking is fascinating. Nowadays, the ability to predict trends or use machine learning to customize ads or shopping preferences has become popular and must-have for all kinds of businesses. While Airbnb is no Amazon or Walmart for advertising, it does have its choices and customization that would benefit from having the ability to learn from the users.

While the idea is simple, the same principles for this Airbnb could also be used to implement more complex algorithms. For example, having a model to determine if a user leaves a review or not a listing could help show

whether the establishment was clean, respectful to its tenants, and popular. This could help further predict why an Airbnb member was perhaps staying there. Was it simply for a concert, traveling for family, or something else?

There are plenty of things that could even be built further from the dataset provided, and it is an excellent experience to start with and is, in part, the motivation behind Airbnb bookings. Advertisements and customized experiences are the future for large companies.

DESIGN AND IMPLEMENTATION

Machine learning algorithms have been around since the 50s with what started as the alpha-beta pruning method. Since then, multiple classifiers and more complex algorithms have been developed and created.

All programming was done using the Jupyter notebook, a web-based interactive development environment for code and data using Python. First, the data was loaded into data frames using pandas from a CSV format. These data frames could then be encoded by normalizing the datatypes. Unlike the practice project, rows with unknown values were left to be included in the encoding as they had some predictive power or were randomly distributed. Since the training data set also had unknown values, removing these values would provide an unacceptable threshold during the model's training.

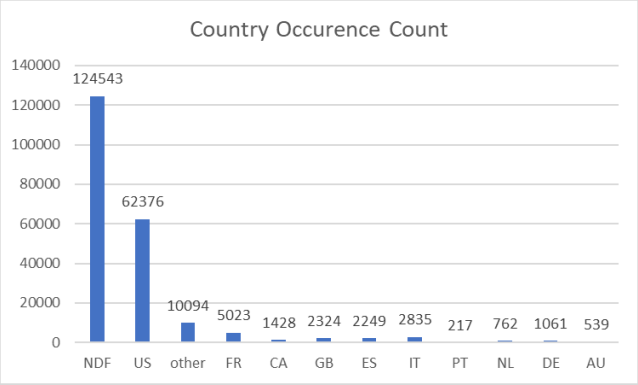

Figure 1: Destination Occurrence Count

As DateTime fields had a very high proportion of NA values, those columns were removed as they would hold low importance or predictive power. Invalid 'age' values were imputed as NA (since it was interpreted that NA values would be randomly distributed). The remaining values were binned into categorical labels for each 5-year range of ages up to 100+. The remaining features` NA values were not removed or attributed to maintaining parity with the unlabeled test_users.csv data.

From sessions.csv, we aggregated rows by user_id and transformed action_type into an array of binary values, indicating for each value whether a user had a row/"action" of a particular action_type. Finally, the transformed action_type collection was merged into the train data. Several models were applied to the dataset to see which model performed the best between themselves and the leaderboard on Kaggle. These models included Logistic Regression, Support Vector Classification, and XGBoost.

Logistic Regression is one of the most popular machine learning algorithms. It is often used to predict the categorical dependent variable using a set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Instead of giving the exact value as 0 or 1, it was utilized to infer probabilistic values which lie between 0 or 1 for each class label to obtain an NDCG score. The logistic regression model is a statistical method often chosen for building machine learning models where the dependent variable is binary to compare it against one or more independent variables.

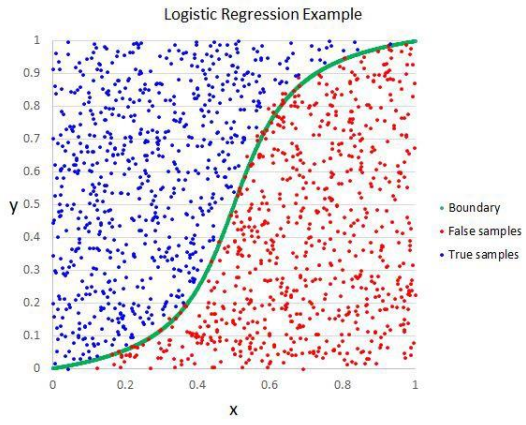

Figure 2: Logistic Regression Example

Support Vector Classification is a more flexible machine-learning method for classification and regression. The main goal of SVMs is to divide the datasets into classes to find a maximum marginal hyperplane, the decision plan

that divides the set of objects having different categories. Since this model can return probability values instead of a single label, it was preferred over doing a simple linear classification such as Linear Support Vector Classification, which in scikit-learn does not support outputting probabilistic values.

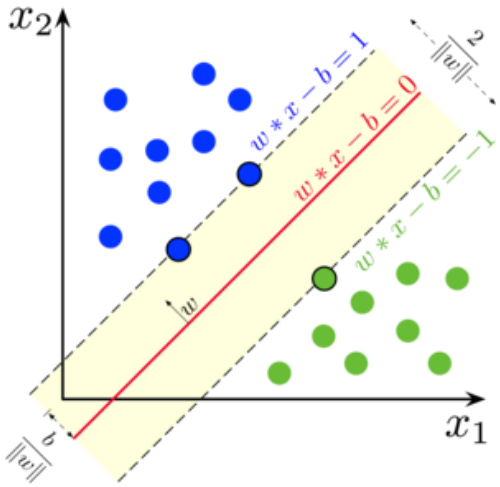

Figure 3: Support Vector Classification Example

XGBoost is a gradient-boosting algorithm that implements several features which differentiate it from other algorithms of its type, including automatic feature selection, advanced penalization of trees, and proportional shrinking of leaf nodes. It is highly efficient, flexible, and portable. Its speed and performance are unparalleled by other algorithms and are the go-to choice when speed is a significant factor for the project. Gradient boosting gives a prediction model in the form of weak prediction models, often decision trees. This method often outperforms random forest algorithms when testing comparable data.

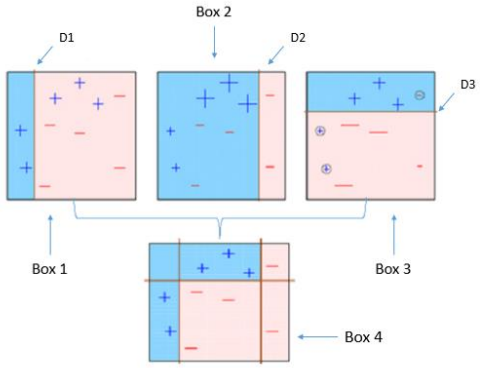

Figure 4: XGBoost Classification Example

The Kaggle Airbnb score evaluation metric used NDCG Normalized discounted cumulative gain where k=5.

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

$$nDCG_k = \frac{DCG_k}{IDCG_k}$$

For each new user in the test data, you can make five ranked predictions on the country of the first booking. Because of this, all classification models decided to use probability over straight Boolean values. As you can see from the equations, these ranked results are cumulative. The probability output lets the model utilize the higher possibilities for its ranked guesses and thus increases the chances of guessing the country of the first user booking.

```python
def ndcg_score(y_true, y_pred, k):
    scores = []
    for true, pred in zip(y_true, y_pred):
        rank = np.argsort(pred)[:-(k+1):-1]
        relevance = np.where(rank == true, 1, 0)
        ndcg_score = np.sum(relevance / np.log2(np.arange(2,
len(relevance) + 2)))
        scores.append(ndcg_score)

    return np.mean(scores)

ndcg_scorer = make_scorer(ndcg_score, needs_proba=True, k=5)
```

Figure 5: NDCG Scoring Funcion

Various tools and techniques were utilized to simplify and accelerate the data mining and machine learning workflow. These methods included pipelines, efficient storing and loading of data, encoders, models, and hardware acceleration tools such as *scikit-learn-intelex*.

Pipelines greatly simplified the creation of models. In the practice project, custom functions were utilized along with sklearn's OneHotEncoder to transform the data and then pass it into a classifier for fitting.

```python
def encode(train_data, test_data, include_continuous=True):
    df = clean(train_data, include_continuous)
    X_train = df.drop(columns=['income'])
    encoder = OneHotEncoder().fit(X_train)
    X_train = encoder.transform(X_train).toarray()
    y_train = df['income'].values
    y_enc = LabelEncoder().fit(y_train)
    y_train = y_enc.transform(y_train)
    df = clean(test_data, include_continuous)
    X_test = df.drop(columns=['income'])
    X_test = encoder.transform(X_test).toarray()
    y_test = y_enc.transform(df['income'].values)

    return X_train, y_train, X_test, y_test, y_enc
```

Figure 6: Sample code from the G9 Practice Project

Pipelines allowed the integration of encoders for the varied columns and classifiers to be serialized into one object, which could be fitted, saved to, and loaded from disk and could transform test data.

```python
clf2 = SVC(random_state=0, probability=True)
pipeline2 = make_pipeline(encoder, clf2)
pipeline2.fit(X_train, y_train)

y_pred = pipeline2.predict_proba(X_test)
ndcg_score(y_test, y_pred, k=5)

joblib.dump(pipeline2, "models/SVC_1.pkl")
```

Figure 7: Sample code from the G9 Final Project

Efficient data handling reduced overhead for experimentation and training. The test and train data from Kaggle, after being processed and merged into a format compatible with machine learning algorithms (which took some time each time the kernel was rebooted), were saved in an efficient hierarchical data format, HDF5.
Models such as Support Vector Classification had long training times above 1 hour (after utilizing scikit-learn-intelex). This and other models, such as those which were final products of resource-intensive GridSearchCV operations, were saved using joblib. These models could be read from disk during the evaluation phase against Kaggle's test data to produce submissions efficiently.

As the Airbnb dataset is relatively large at over 200,000 rows for train_users_2.csv, Support Vector Machine models would generally not be feasible candidates. However, the Intel Extension for Scikit-Learn (scikit-learn-intel OR sklearnex) provides built-in parallelization and hardware acceleration of some sklearn algorithms, particularly for SVC, yielding faster training by a factor of at least 20 and faster inference by a factor of at least 100 on large datasets. [4]

[Subsection(s) on Training, Hyperparameter Tuning/Cross Validation, and Final Evaluation]

Train-Test splitting of labeled data and k-fold cross-validation were used to evaluate models and mitigate overfitting. The implemented NDCG scorer was incorporated into cross-validation metrics in order to orient CV to our success metric.

Hyperparameter Tuning was utilized to refine our models for the highest performance feasible given the preprocessing, hardware, and time available.

Logistic Regression hyperparameter search included: C, solver, max_iter.

XGBoost hyperparameter search included: subsample, min_child_weight, max_depth, gamma, colsample_bytree.

SVC hyperparameter search included: kernel, C (SVC training times limited the practical search space available).

RESULTS

Below are the top classifier scores and ranks reported on Kaggle at the time of writing. Currently, there are 1500 user scores on the private leaderboard. However, as more scores get added and registered or new algorithms are developed, these ranks are expected to be pushed back.

Classifier	Score	Rank out of 1459	Percentile
Logistic Regression	0.87940	314	78.5%
XGB Classifier	0.88155	289	80.2%
SVC Model	0.8819	279	80.9%

Our Logistic Regression model utilized the following parameters:

C	0.008
max_iter	500
solver	lbfgs

Our XGB Classifier model utilized the following parameters:

subsample	1.0
min_child_weight	10
max_depth	5
gamma	1.5
colsample_bytree	0.8
tree_method	hist

Our SVC model utilized the following parameters:

kernel	poly
C	1.0

SVC had the highest score at 0.88190, ranking at the 80.9 percentile. XGB Classifier came in as a good second with a score of 0.88155, and its score is in the 80.2 percentile.

The average percentile with all three scores was 79.9, which is a massive success with the classifiers chosen.
The average score of these three is 0.8810, which happens to be closer to the SVC model Classifier.

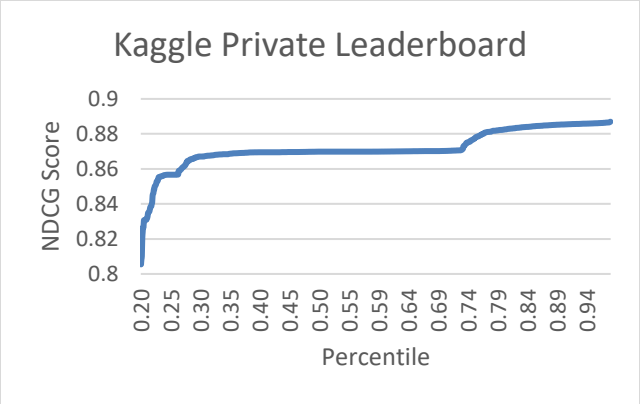

Figure 8: Scores from the Kaggle Private Leaderboard

DISCUSSION

Compared to the leaderboard, the results from this project were overwhelmingly good and above the 75th percentile mark. The highest being the SVC at 0.88175 and 80.9 percentile.

As mentioned earlier, the Airbnb data was not the first selected dataset on which to base this project. Unfortunately, due to the size of the Microsoft malware prediction dataset, it was not viable to obtain results in the time frame for this project. Much research went into optimizing data frames and memory management for encoding and running the classifications. However, issues persisted with memory limitations on devices. Even running the classifications on 128 Gb servers would have resulted in problems and days for it to train.

Airbnb did not have this issue with size and complexity. While it was relatively simple with hardware implementation, it provided the much-needed time and options for testing more complex algorithms and expropriation into different machine learning models. Still, the Microsoft malware prediction code is there, and at some point, I would like to finish that to test large-scale data and objects in the future.

It was not apparent during writing that the XGBoost classifier would outperform the others. But with logistic regression coming in with a close second, only .4 percent separates these two scores. However, this shouldn't be used to determine if one algorithm is better than the other. These results are data-specific, and it could be that the overall XGBClassifier has the upper hand compared to

others; however, more testing with this data would have to be don't to prove this fact.

The clustering of results of our candidate models suggests that the primary factor in overall performance was the ability to understand the data and process it accordingly. Therefore, to improve the results, efforts into further feature engineering would be the most effective at increasing our models` performance.

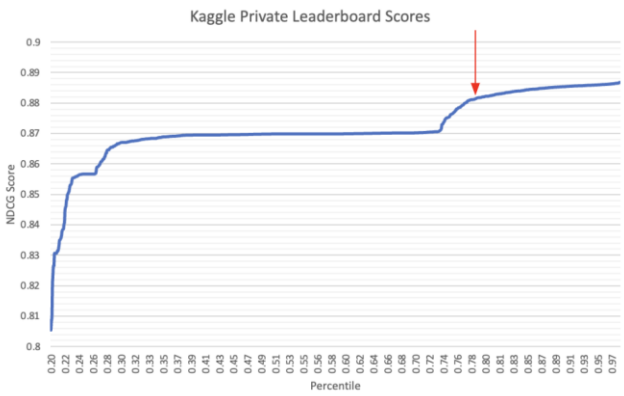

Figure 9: Location of Highest Score

ACKNOWLEDGMENTS

Kaggle is an excellent resource for testing different classifiers on any dataset imaginable. From infection rates of covid to data breaches and internet usage analysis, Kaggle has thousands of datasets that anyone can access. So, whether you want to compete in various competitions for prizes, or just someone who wants to test their knowledge in data algorithms, Kaggle is fantastic in both regards.

Airbnb has graciously provided data regarding customers such that we can implement algorithms to predict first-user bookings. Without the data and small competition, this project and many like it would not have been possible. While this competition was mainly used for finding experienced individuals for job offers, the data is still accessible for us students to see their current rankings and learn from the experience and provided data.

Professor Xintao Wu has been a wealth of information on the dealings of classifiers and data processing. Thanks to his practice project earlier this semester, this final project has been significantly easier to implement with the previous experience we gained from the practice. While the projects themselves are different in terms of data, they both require reading data, pre-processing, training, and sampling allowing the use of reusing the various functions already

Google Vertex for providing $300 worth of credits to use for renting computer clusters for computationally intensive jobs.

REFERENCES

[1] https://scikit-learn.org/0.15/index.html#
[2] https://jupyter.org/
[3] https://arxiv.org/abs/1603.02754
[4] https://github.com/IntelPython/scikit-learn_bench

Airbnb

December 5, 2022

1 Final Project Notebook

Group: 9 Group Members: Shawn Ericksen (ericksen@uark.edu), Garret Fulghum (gmfulghu@uark.edu), Wesley Parker (wgparker@uark.edu)

This practice project focuses on the Airbnb New User Bookings dataset. This can be accessed from: https://www.kaggle.com/competitions/airbnb-recruiting-new-user-bookings/data

1.1 Kaggle Performance Info

Kaggle's scoring for this competition utilizes a Normalized Discounted Cumulative Gain (NDCG) scoring, where up to 5 guesses of destination county (ordered by confidence) are submitted per entry in the test data. A score of 1.0 reflects the first guess being correct, and less points for other scenarios (0.63 for the second guess being correct and so on).

Q1/Q2/Q3 Kaggle scores: 0.85535 / 0.86979 / 0.87062

Notes:

The baseline, which is to always guess NDF-US-OTHER-FR-IT, recieves a NDCG score of 0.85669 on the Private Leaderboard. The NDCG score of the dummy model on the training data is 0.80676.

As for scikit's accuracy score, always guessing NDF results in a score of 0.58347 against the training data.

1.1.1 Usage

Running the second code cell will prompt the user read data from CSV or HDF5 (setting ___no_prompt___ will skip the prompt and use HDF5).

1.1.2 Imports and reading dataset into memory

[1]:
```
from pathlib import Path

import numpy as np
import pandas as pd
import csv

from statistics import mean
from scipy.stats import uniform, loguniform, randint
```

```
from sklearnex import patch_sklearn
patch_sklearn()

from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_transformer, make_column_selector
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split, cross_val_predict,␣
 ↪cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import make_scorer


import ray.tune as tune
from tune_sklearn import TuneGridSearchCV, TuneSearchCV

from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.neural_network import MLPClassifier

from sklearn import set_config
set_config(display='diagram')

import xgboost as xgb

import joblib
from joblib import parallel_backend
```

```
Intel(R) Extension for Scikit-learn* enabled (https://github.com/intel/scikit-
learn-intelex)
```

[2]:
```
%%time
__no_prompt__ = False # If True, skips user input (defaults to HDF).
__no_hdf__ = False # If True, skips prompt and only reads from CSV.

data_folder = Path("airbnb-recruiting-new-user-bookings/") # Replace if using␣
 ↪Kaggle Notebook
hdf_path = data_folder / "data.h5"
use_hdf = not __no_hdf__ and (__no_prompt__ or input("Do you want to use an HDF␣
 ↪file [Y/n]:") != "n") if hdf_path.exists() else False

if use_hdf: # Read from hdf if availible (much faster)
```

```
    df_merge = pd.read_hdf(hdf_path)
else:
    filepath = data_folder / "train_users_2.csv"
    dtypes={'id': 'object', 'date_account_created': 'string',␣
 ↪'timestamp_first_active': 'string', 'date_first_booking': 'string', 'gender':
 ↪ 'category', 'age': 'float64', 'signup_method': 'category', 'signup_flow':␣
 ↪'category', 'language': 'category', 'affiliate_channel': 'category',␣
 ↪'affiliate_provider': 'category', 'first_affiliate_tracked': 'category',␣
 ↪'signup_app': 'category', 'first_device_type': 'category', 'first_browser':␣
 ↪'category', 'country_destination': 'category'}
    parse_dates = ['date_account_created', 'timestamp_first_active',␣
 ↪'date_first_booking']
    cols = list(pd.read_csv(filepath, nrows=1))[1:]
    df = pd.read_csv(filepath, dtype=dtypes, na_values=['-unknown-', '<NA>'],␣
 ↪parse_dates=parse_dates
                     , infer_datetime_format=True)
    df['date_first_booking'] = pd.to_datetime(pd.
 ↪Series(df['date_first_booking'])
                                              , format='%Y-%m-%d',␣
 ↪errors='coerce')

#     df.select_dtypes('datetime64[ns]').fillna(pd.NaT)
    # df = df.set_index('id')
    df['age'] = df['age'].replace(range(2000, 2015), np.nan)
    df['age'] = pd.cut(df['age'], bins = [i*5 for i in range(0, 21)] + [120]
                       , labels=(['%d-%d' % (i*5, i*5+4) for i in range(0, 20)]␣
 ↪+ ['100+']))

#     filepath = data_folder / "age_gender_bkts.csv"
#     dtypes = {}

    filepath = data_folder / "sessions.csv"
    dtypes={'user_id': 'string', 'action': 'category', 'action_type':␣
 ↪'category', 'action_detail': 'category', 'device_type': 'category',␣
 ↪'secs_elapsed': 'float64'}
    df_session = pd.read_csv(filepath, dtype=dtypes, na_values=['-unknown-',␣
 ↪'NDF', '<NA>'])

    S = df_session.groupby(by='user_id', as_index=False).aggregate(lambda x:x.
 ↪tolist())

    id_list = []
    types_list = []
    for row in S.values:
        id_list.append(row[0])
        types_list.append(np.unique(row[2]))
```

```
    rows = []
    for user_id, types in zip(id_list, types_list):
        types = np.delete(types, np.where(types == 'nan'))
        row = [user_id]
        for col in df_session['action_type'].cat.categories:
            if col in types:
                row.append(1)
            else:
                row.append(0)
        rows.append(row)

    t_cols = ['user_id'] + list(df_session['action_type'].cat.categories)
    t_dtypes = {**{'user_id': 'string'}, **{col: 'category' for col in t_cols[1:
↪]}}
    df_action_types = pd.DataFrame(np.asarray(rows), columns=t_cols).
↪astype(t_dtypes)

    df_merge = df.merge(df_action_types, left_on ='id', right_on='user_id').
↪drop(columns=['user_id'])
    df_merge = df_merge.set_index('id')

    # Save to hdf if reading from csv
    df_merge.to_hdf(data_folder / "data.h5", key='df', mode='w', format="table")
```

```
Do you want to use an HDF file [Y/n]:
Wall time: 4.19 s
```

1.1.3 Diagnostics

[3]:
```
print("- - test_users_2.csv - -")
print("Number of lines present: ", len(df_merge))
print("Number of Columns: ", len(df_merge.columns))
```

```
- - test_users_2.csv - -
Number of lines present:  73815
Number of Columns:  24
```

[4]:
```
topCount = 5
print("Top ", topCount, " dataFrames:")
print(df_merge.head(topCount))
```

```
Top  5  dataFrames:
           date_account_created timestamp_first_active date_first_booking  \
id
d1mm9tcy42           2014-01-01    2014-01-01 00:09:36         2014-01-04
yo8nz8bqcq           2014-01-01    2014-01-01 00:15:58                NaT
```

```
4grx6yxeby           2014-01-01    2014-01-01 00:16:39                NaT
ncf87guaf0           2014-01-01    2014-01-01 00:21:46                NaT
4rvqpxoh3h           2014-01-01    2014-01-01 00:26:19         2014-01-02

           gender    age signup_method signup_flow language affiliate_channel  \
id
d1mm9tcy42   MALE  60-64         basic           0       en     sem-non-brand
yo8nz8bqcq    NaN    NaN         basic           0       en            direct
4grx6yxeby    NaN    NaN         basic           0       en         sem-brand
ncf87guaf0    NaN    NaN         basic           0       en            direct
4rvqpxoh3h    NaN    NaN         basic          25       en            direct

           affiliate_provider  … country_destination booking_request click  \
id                             …
d1mm9tcy42             google  …               other               0     1
yo8nz8bqcq             direct  …                 NDF               0     1
4grx6yxeby             google  …                 NDF               1     0
ncf87guaf0             direct  …                 NDF               0     1
4rvqpxoh3h             direct  …                  GB               0     0

           data message_post partner_callback submit view booking_response  \
id
d1mm9tcy42    1            1                0      1    1                0
yo8nz8bqcq    1            0                0      1    1                0
4grx6yxeby    1            1                0      1    1                0
ncf87guaf0    1            0                0      1    1                0
4rvqpxoh3h    0            0                0      0    0                0

           modify
id
d1mm9tcy42      0
yo8nz8bqcq      0
4grx6yxeby      0
ncf87guaf0      0
4rvqpxoh3h      0

[5 rows x 24 columns]
```

[5]:
```
df_merge.memory_usage(deep=True, index=False).sort_values(ascending=False)
```

[5]:
```
date_account_created       590520
date_first_booking         590520
timestamp_first_active     590520
first_browser               79329
language                    75846
age                         75668
affiliate_provider          75521
```

```
signup_flow                75366
country_destination        74827
first_device_type          74737
affiliate_channel          74628
first_affiliate_tracked    74570
signup_app                 74233
signup_method              74113
gender                     74109
booking_request            74039
click                      74039
data                       74039
message_post               74039
partner_callback           74039
submit                     74039
view                       74039
booking_response           74039
modify                     74039
dtype: int64
```

[6]:
```
df_merge.memory_usage(deep=True, index=False).sum()
```

[6]:
```
3340858
```

1.1.4 Preprocessing

[7]:
```
X = df_merge.drop(columns=['country_destination'])
X = X.drop(columns=['date_account_created', 'timestamp_first_active',␣
 ↪'date_first_booking'])
```

[8]:
```
# This block can be commented out when doing prediction on the Kaggle test.csv
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0, shuffle=True)
```

[9]:
```
# numeric_transformer = make_pipeline(
# #     SimpleImputer(strategy='mean'),
# #     StandardScaler()
# )

encoder = make_column_transformer(
#     (numeric_transformer, ['age']),
    (OneHotEncoder(sparse=False, handle_unknown='ignore'),␣
 ↪make_column_selector(dtype_include='category')),
    remainder='passthrough'
)
encoder
```

[9]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('onehotencoder',
                                 OneHotEncoder(handle_unknown='ignore',
                                               sparse=False),
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])
```

[10]:
```
y_enc = LabelEncoder().fit(y_train.values)
y_train = y_enc.transform(y_train)
y_test = y_enc.transform(y_test.values)
```

[11]:
```
joblib.dump(y_enc, Path("models/LabelEncoder.pkl"))
```

[11]:
```
['models\\LabelEncoder.pkl']
```

1.1.5 NDCG Score Implementation

(1, 0, 0, 0, 0) will serve as the true ranking.

The prediction (NDF-US-OTHER-FR-IT) will be used as a dummy model. It will be transformed into an ndarray such that incorrect guesses are transformed to 0 and correct guesses to 1.

A first-rank correct guess generally appears as (1, 0, 0, 0, 0), a second-rank correct guess as (0, 1, 0, 0, 0), and no correct guess as (0, 0, 0, 0, 0).

[12]:
```
def ndcg_score(y_true, y_pred, k):
    scores = []
    for true, pred in zip(y_true, y_pred):
        rank = np.argsort(pred)[:-(k+1):-1]
        relevance = np.where(rank == true, 1, 0)
        dcg_score = np.sum(relevance / np.log2(np.arange(2, len(relevance) +␣
 ↪2)))
        scores.append(dcg_score)

    return np.mean(scores)

ndcg_scorer = make_scorer(ndcg_score, needs_proba=True, k=5)
```

[15]:
```
%%time
# NDCG for the dummy model NDF-US-other-FR-IT as a series of probabilities
y_pred = np.asarray([[0, 0, 0, 0, .0625, 0, 0.03125, .5, 0, 0, .25, .
 ↪125]]*df_merge.shape[0])

ndcg_score(y_enc.transform(df_merge['country_destination'].values), y_pred, k=5)
```

```
Wall time: 1.05 s
```

[15]:
```
0.8202097562438039
```

1.1.6 Logistic Regression

[16]:
```
clf0 = LogisticRegression(C=0.08, max_iter=500)

pipeline0 = make_pipeline(encoder, clf0)
pipeline0
```

[16]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
                                                                 sparse=False),
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])),
                ('logisticregression',
                 LogisticRegression(C=0.08, max_iter=500))])
```

[17]:
```
%%time
# Fit the pipeline on the training data
pipeline0.fit(X_train, y_train)

# Predict probabilities
y_pred = pipeline0.predict_proba(X_test)
```

```
Wall time: 2.14 s
```

[18]:
```
ndcg_score(y_test, y_pred, k=5)
```

[18]:
```
0.8495785851987129
```

[19]:
```
%%time
np.mean(cross_val_score(pipeline0, X_train, y_train, scoring=ndcg_scorer))
```

```
Wall time: 7.89 s
```

[19]:
```
0.8505219978430472
```

LR with GridSearchCV

[ ]:
```
# params = {
#     'logisticregression__C': [0.1, 1.0, 10.0, 100.0],
#     'logisticregression__penalty': ['l2'], # ,l1
#     'logisticregression__solver': ['lbfgs', 'liblinear', 'saga']
# }

# grid_search = GridSearchCV(pipeline0, params, n_jobs=-1, verbose=1, cv=5,␣
 ↪scoring=ndcg_scorer)
```

[ ]:
```
# %%time
# grid_search.fit(X_train, y_train)
# grid_search.best_params_
```

[ ]:
```
# to_drop = ['mean_score_time', 'std_score_time', 'std_fit_time', 'params',␣
 ↪'split0_test_score', 'split1_test_score', 'split2_test_score',␣
 ↪'split3_test_score', 'split4_test_score']
# result = pd.DataFrame(grid_search.cv_results_)
# result = result.sort_values('mean_test_score', axis=0, ascending=False).
 ↪drop(columns=to_drop)
# result = result.rename(columns=lambda x: x[x.find('__')+1:])
# result
```

[ ]:
```
# y_pred = grid_search.predict_proba(X_test)
# ndcg_score(y_test, y_pred, k=5)
```

[ ]:
```
# Pickle model and write to hard drive
# joblib.dump(pipeline0, "models/LogisticRegression.pkl")
```

1.1.7 SVC

[20]:
```
clf2 = SVC(C=1, kernel='poly', random_state=0, probability=True)

pipeline2 = make_pipeline(encoder, clf2)
pipeline2
```

[20]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
                                                                  sparse=False),
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])),
                ('svc',
                 SVC(C=1, kernel='poly', probability=True, random_state=0))])
```

[21]:
```
%%time
# Fit the pipeline on the training data
pipeline2.fit(X_train, y_train)
```

```
Wall time: 7min 35s
```

[21]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
                                                                  sparse=False),
```

```
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])),
                ('svc',
                 SVC(C=1, kernel='poly', probability=True, random_state=0))])
```

[22]:
```
%%time
pipeline2.cross_val_score(X_train, y_train)
```

```
---------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<timed eval> in <module>

AttributeError: 'Pipeline' object has no attribute 'cross_val_score'
```

[ ]:
```
# params = {
#     'svc__C': [0.2, 0.4, 0.6, 0.8, 1]
#     'svc__kernel: ['linear', 'poly', 'rfb', 'sigmoid']'
# }

# grid_search = GridSearchCV(pipeline2, param_grid=params, verbose=1,␣
 ↪cv=5,scoring=ndcg_scorer)
```

[ ]:
```
# %%time
# grid_search.fit(X_train, y_train)
# print(grid_search.best_params_)
# print(grid_search.best_score_)
```

[23]:
```
%%time
# Score the pipeline on the testing data
y_pred = pipeline2.predict_proba(X_test)
ndcg_score(y_test, y_pred, k=5)
```

```
Wall time: 3.33 s
```

[23]:
```
0.8511589301805949
```

[25]:
```
# Pickle model and write to hard drive
joblib.dump(pipeline2, "models/SVC.pkl")
```

[25]:
```
['models/SVC.pkl']
```

1.1.8 XGBoostClassifier

[27]:
```
clf6 =xgb.XGBClassifier(objective='mulit:softprob',
                        tree_method='hist',
                        subsample=1.0,
```

```
                        min_child_weight=10,
                        max_depth=5,
                        gamma=1.5,
                        colsample_bytree=0.8
)

pipeline6 = make_pipeline(encoder, clf6)
pipeline6
```

[27]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])),
                ('xgbclassifier',
                 XGBClassifier(base_score=None, booster=None, callbacks=None,
                               colsample_bylevel=None, c…
                               grow_policy=None, importance_type=None,
                               interaction_constraints=None, learning_rate=None,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=None,
                               max_depth=5, max_leaves=None,
                               min_child_weight=10, missing=nan,
                               monotone_constraints=None, n_estimators=100,
                               n_jobs=None, num_parallel_tree=None,
                               objective='mulit:softprob', predictor=None,
…))])
```

[28]:
```
%%time
# Fit the pipeline on the training data
pipeline6.fit(X_train, y_train)
```

```
Wall time: 1.87 s
```

[28]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False),
<sklearn.compose._column_transformer.make_column_selector object at
0x00000276D8CCD2B0>)])),
                ('xgbclassifier',
                 XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
                               colsample_bylevel=1, c…
                               feature_types=None, gamma=1.5, gpu_id=-1,
```

```
                                    grow_policy='depthwise', importance_type=None,
                                    interaction_constraints='',
                                    learning_rate=0.300000012, max_bin=256,
                                    max_cat_threshold=64, max_cat_to_onehot=4,
                                    max_delta_step=0, max_depth=5, max_leaves=0,
                                    min_child_weight=10, missing=nan,
                                    monotone_constraints='()', n_estimators=100,
                                    n_jobs=0, num_parallel_tree=1,
                                    objective='multi:softprob', predictor='auto',
    …))])
```

[29]:
```
# Score the pipeline on the testing data
y_pred = pipeline6.predict_proba(X_test)
ndcg_score(y_test, y_pred, k=5)
```

[29]:
```
0.8506915820090147
```

[31]:
```
cross_val_score(pipeline6, X_train, y_train, scoring=ndcg_scorer)
```

[31]:
```
array([0.85046852, 0.85378984, 0.85228106, 0.84960463, 0.85171962])
```

[ ]:
```
# search_space = {
    # "xgbclassifier__n_estimators": tune.loguniform(100, 10000),
    # "xgbclassifier__max_depth": [tune.randint(0, 5)],
    # "xgbclassifier__gamma": tune.quniform(0, 5, 0.1),
    # "xgbclassifier__min_child_weight": tune.quniform(1, 20, 1),
    # "xgbclassifier__subsample": tune.quniform(0.25, 0.75, 0.01),
    # "xgbclassifier__colsample_bytree": tune.quniform(0.05, 0.5, 0.01),
    # "xgbclassifier__colsample_bylevel": tune.quniform(0.05, 0.5, 0.01),
    # "xgbclassifier__learning_rate": tune.quniform(0, 10, 0.1),  # powers of 10
# }

# tune_params = [k for k in search_space.keys() if k != 'wandb']

# config['xgbclassifier__n_estimators'] =␣
 ↪int(config['xgbclassifier__n_estimators'])

# tune_search = TuneSearchCV(
    # pipeline6,
    # param_distributions=search_space,
    # n_trials=3,
    # early_stopping=True,
    # scoring=ndcg_scorer,
    # search_optimization="bayesian",
# )
# tune_search
```

[ ]:
```
# params = {
#     "xgbclassifier__learning_rate"    : uniform(0.05, 0.4),
#     "xgbclassifier__max_depth"        : randint(4, 10),
#     "xgbclassifier__min_child_weight" : randint(3, 10),
#     "xgbclassifier__gamma"            : uniform(0, 0.6),
#     "xgbclassifier__colsample_bytree" : uniform(0.3, 0.5),
    # "xgbclassifier__subsample" : uniform(0.8, 1)
# }

# random_search = RandomizedSearchCV(
#     pipeline6,
#     param_distributions=params,
#     n_iter=3000,
#     scoring=ndcg_scorer
# )
```

[ ]:
```
# %%time
# random_search.fit(X_train, y_train)
```

[ ]:
```
# print(random_search.best_params_)
# print(random_search.best_score_)
# y_pred = random_search.predict_proba(X_test)
# print(ndcg_score(y_test, y_pred, k=5))
```

[ ]:
```
# Pickle model and write to hard drive
# joblib.dump(random_search, "models/XGBClassifier.pkl")
```

1.2 Predictions for Kaggle's test_users.csv

[32]:
```
%%time
__no_prompt__ = False # If True, skips user input (defaults to HDF).
__no_hdf__ = False # If True, skips prompt and only reads from CSV.

data_folder = Path("airbnb-recruiting-new-user-bookings/") # Replace if using␣
 ↪Kaggle Notebook
hdf_path = data_folder / "test_data.h5"
use_hdf = not __no_hdf__ and (__no_prompt__ or input("Do you want to use an HDF␣
 ↪file [Y/n]:") != "n") if hdf_path.exists() else False

if use_hdf: # Read from hdf if availible (much faster)
    df_kaggle = pd.read_hdf(hdf_path)
else:
    filepath = data_folder / "test_users.csv"
```

```
    dtypes={'id': 'object', 'date_account_created': 'string',␣
 ↪'timestamp_first_active': 'string', 'date_first_booking': 'string', 'gender':
 ↪ 'category', 'age': 'float64', 'signup_method': 'category', 'signup_flow':␣
 ↪'category', 'language': 'category', 'affiliate_channel': 'category',␣
 ↪'affiliate_provider': 'category', 'first_affiliate_tracked': 'category',␣
 ↪'signup_app': 'category', 'first_device_type': 'category', 'first_browser':␣
 ↪'category', 'country_destination': 'category'}
    parse_dates = ['date_account_created', 'timestamp_first_active',␣
 ↪'date_first_booking']
    cols = list(pd.read_csv(filepath, nrows=1))[1:]
    df = pd.read_csv(filepath, dtype=dtypes, na_values=['-unknown-', '<NA>'],␣
 ↪parse_dates=parse_dates
                     , infer_datetime_format=True)
    df['date_first_booking'] = pd.to_datetime(pd.
 ↪Series(df['date_first_booking'])
                                              , format='%Y-%m-%d',␣
 ↪errors='coerce')

#     df.select_dtypes('datetime64[ns]').fillna(pd.NaT)
    # df = df.set_index('id')
    df['age'] = df['age'].replace(range(2000, 2015), np.nan)
    df['age'] = pd.cut(df['age'], bins = [i*5 for i in range(0, 21)] + [120]
                       , labels=(['%d-%d' % (i*5, i*5+4) for i in range(0, 20)]␣
 ↪+ ['100+']))

#     filepath = data_folder / "age_gender_bkts.csv"
#     dtypes = {}

    filepath = data_folder / "sessions.csv"
    dtypes={'user_id': 'string', 'action': 'category', 'action_type':␣
 ↪'category', 'action_detail': 'category', 'device_type': 'category',␣
 ↪'secs_elapsed': 'float64'}
    df_session = pd.read_csv(filepath, dtype=dtypes, na_values=['-unknown-',␣
 ↪'NDF', '<NA>'])

    S = df_session.groupby(by='user_id', as_index=False).aggregate(lambda x:x.
 ↪tolist())

    id_list = []
    types_list = []
    for row in S.values:
        id_list.append(row[0])
        types_list.append(np.unique(row[2]))

    rows = []
    for user_id, types in zip(id_list, types_list):
```

```
        types = np.delete(types, np.where(types == 'nan'))
        row = [user_id]
        for col in df_session['action_type'].cat.categories:
            if col in types:
                row.append(1)
            else:
                row.append(0)
        rows.append(row)

    t_cols = ['user_id'] + list(df_session['action_type'].cat.categories)
    t_dtypes = {**{'user_id': 'string'}, **{col: 'category' for col in t_cols[1:
↪]}}
    df_action_types = pd.DataFrame(np.asarray(rows), columns=t_cols).
↪astype(t_dtypes)

    df_kaggle = df.merge(df_action_types, left_on ='id', right_on='user_id').
↪drop(columns=['user_id'])
    df_kaggle = df_kaggle.set_index('id')

    # Save to hdf if reading from csv
    df_kaggle.to_hdf(data_folder / "test_data.h5", key='df', mode='w',␣
↪format="table")
```

```
Do you want to use an HDF file [Y/n]:
Wall time: 1.43 s
```

[33]:
```
X_kaggle = df_kaggle.drop(columns=['date_account_created',␣
 ↪'timestamp_first_active', 'date_first_booking'])
id_list = np.asarray(X_kaggle.index)
```

[34]:
```
filename = Path("models/SVC.pkl")
loaded_model = joblib.load(filename)
le_filename = Path("models/LabelEncoder.pkl")
y_enc = joblib.load(le_filename)
```

[35]:
```
%%time
y_kaggle = loaded_model.predict_proba(X_kaggle)
ranks = y_kaggle.argsort()[:, :6:-1]
orders = []
for rank in ranks:
    orders.append(y_enc.inverse_transform(rank))
```

```
Wall time: 9.83 s
```

[36]:
```
with open("submission.csv", "w", newline="") as csvfile:
    csv_writer = csv.writer(csvfile, delimiter=',', quotechar='"')
    csv_writer.writerow(['id', 'country'])
```

```
    for i, order in zip(id_list, orders):
        for country in order:
            row = [i, country]
            csv_writer.writerow(row)
```