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)
```