tryout_1

September 9, 2019

[1]:
```python
import gc
gc.collect()
```

[1]: 62

[2]:
```python
# Competition link:
# https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot
```

[3]:
```python
# Bidder
# bidder_id  Unique identifier of a bidder.
# payment_account  Payment account associated with a bidder. These are␣
↪obfuscated to protect privacy.
# address  Mailing address of a bidder. These are obfuscated to protect privacy.
↪
# outcome  Label of a bidder indicating whether or not it is a robot. Value 1.0␣
↪indicates a robot, where value 0.0 indicates human.

# Bidder: train.csv and tesst.csv

# Bid
# bid_id - unique id for this bid
# bidder_id  Unique identifier of a bidder (same as the bidder_id used in train.
↪csv and test.csv)
# auction  Unique identifier of an auction
# merchandise   The category of the auction site campaign, which means the␣
↪bidder might come to this site by way of searching for "home goods" but␣
↪ended up bidding for "sporting goods" - and that leads to this field being␣
↪"home goods". This categorical field could be a search term, or online␣
↪advertisement.
# device  Phone model of a visitor
# time - Time that the bid is made (transformed to protect privacy).
# country - The country that the IP belongs to
# ip  IP address of a bidder (obfuscated to protect privacy).
# url - url where the bidder was referred from (obfuscated to protect privacy).

# Bids: bids.csv
```

[97]:
```python
# Load libraries

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd

import scipy.stats as sstats

from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import roc_curve, auc, classification_report,␣
 ↪accuracy_score
```

[5]:
```python
# Load datasets

BIDS_CSV_FILEPATH = 'data/bids.csv'
BIDDERS_CSV_TRAIN_FILEPATH = 'data/train.csv'

bids = pd.read_csv(BIDS_CSV_FILEPATH, header=0)
bidders = pd.read_csv(BIDDERS_CSV_TRAIN_FILEPATH, header=0)
```

[6]:
```python
# Overview loaded data

def overview_df( dataset_df ):
    display(dataset_df.sample())
    display(dataset_df.shape)

    dataset_df.info()
```

[7]:
```python
# overview_df( bids )

# overview_df( bidders )
```

[8]:
```python
# Left join the data

bids_bidders = pd.merge( bidders, bids, on='bidder_id', how='left' )

# I won't use bids or bidders by separately anymore - free them

# display( bids_bidders.shape )
# display( bids.shape, bidders.shape )

# del bids
# del bidders
```

[9]:
```python
# overview_df( bids_bidders )
```

[10]:
```python
# Check for NaN values

# display( bids_bidders.isnull().sum() )

# 29 - all these are bidders without bids
bidders_without_bids = bids_bidders[ bids_bidders['bid_id'].isnull() ]
# display(bidders_without_bids)
# display(bidders_without_bids.shape)  # (29, 12)

# Because all of these were human (outcome=0.0) - drop them
bids_bidders = bids_bidders.drop( bidders_without_bids.index )
```

[11]:
```python
# Check for NaNs again: 2701 missing countries

# display( bids_bidders.isnull().sum() )

# Solution for now: drop them
bids_bidders = bids_bidders.dropna()

# No NaNs left
# display( bids_bidders.isnull().sum() )
```

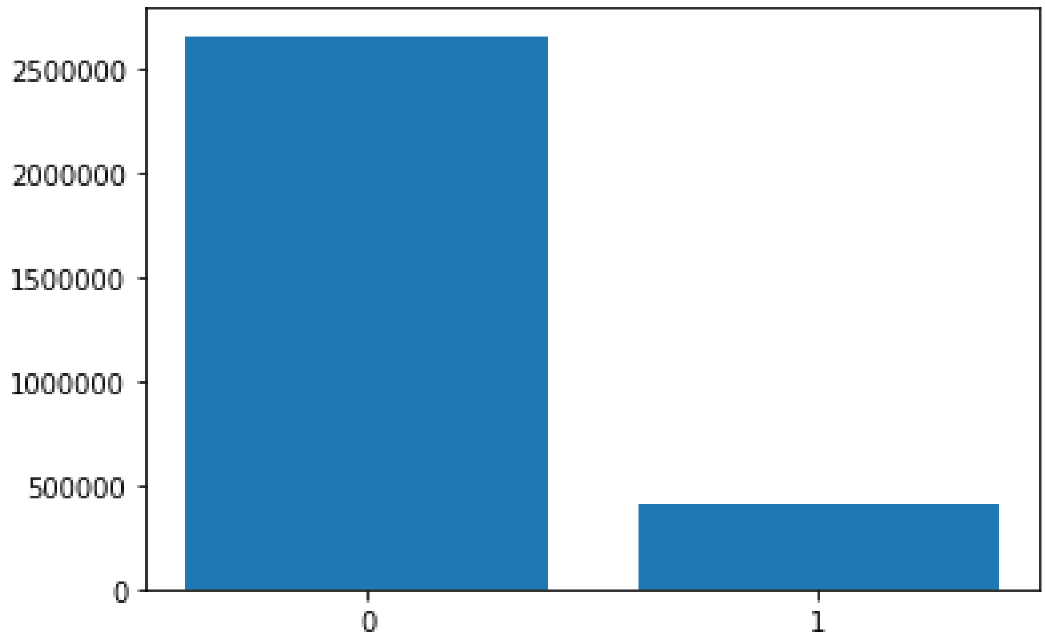
[12]:
```python
# Overview duplicated values

display( bids_bidders.duplicated().sum() )
```

```
0
```

[13]:
```python
# Overview unique values

for col_name in bids_bidders:
    print('{0} - {1}'.format( col_name, len(bids_bidders[col_name].unique()) ))

# (bidder_id, payment_account, address) - all are unique => linearly dependable␣
↪=> drop 2/3 of them later
```

```
bidder_id - 1983
payment_account - 1983
address - 1983
outcome - 2
bid_id - 3068523
auction - 12740
merchandise - 10
device - 5726
time - 742582
country - 198
ip - 1028810
url - 663265
```

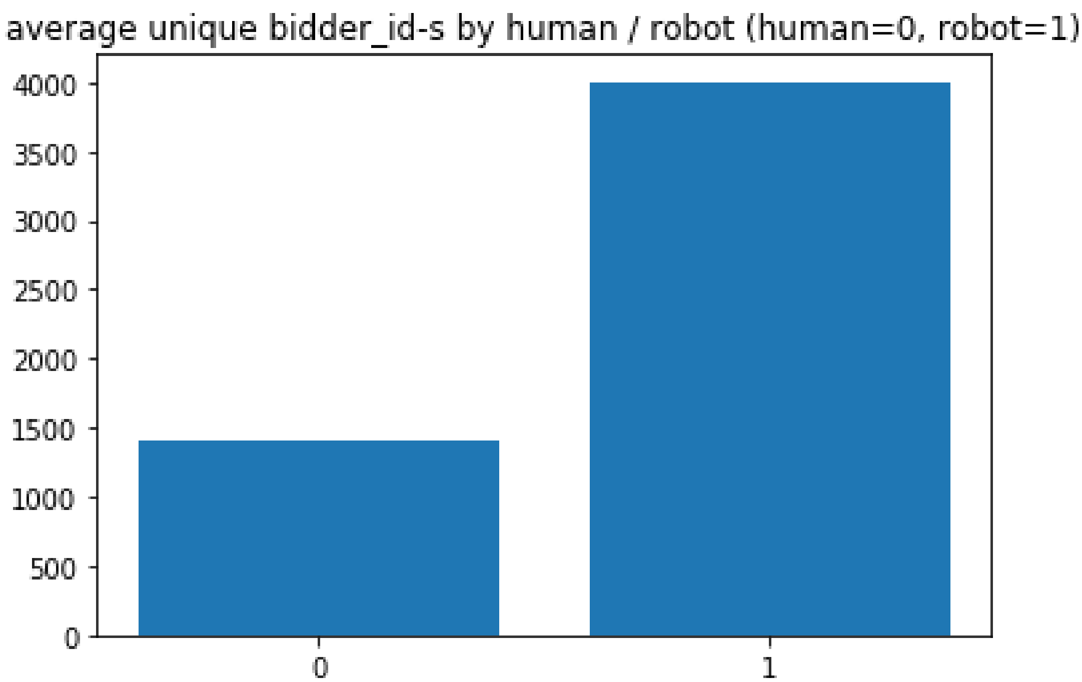
[14]:
```python
# Overview robots vs humans

outcome_freq = bids_bidders['outcome'].value_counts()

plt.bar( outcome_freq.index.values, outcome_freq.values )
plt.xticks( outcome_freq.index.values )
plt.show()

# result - totally not well weighted distribution - imbalanced classes
# using f1/prec/recall/acc would be a mistake
```



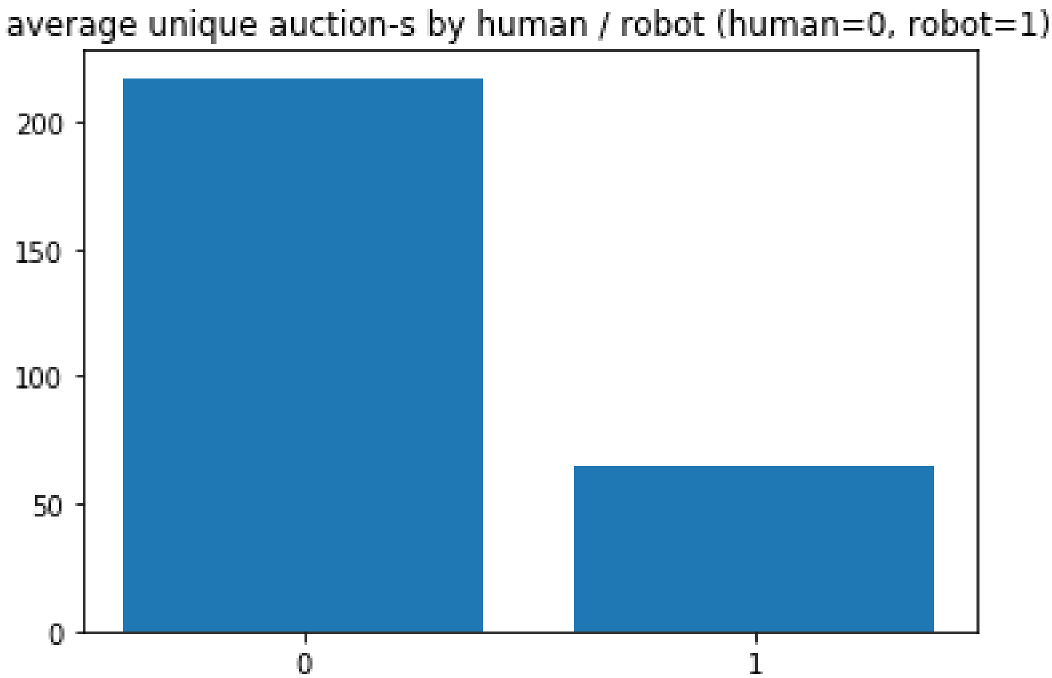
[15]:
```python
# Overview mean cnt for robots / for humans

HUMAN_OUTCOME_VALUE = 0.0
ROBOT_OUTCOME_VALUE = 1.0

bids_bidders_humans = bids_bidders[ bids_bidders['outcome'] ==␣
 ↪HUMAN_OUTCOME_VALUE ]
bids_bidders_robots = bids_bidders[ bids_bidders['outcome'] ==␣
 ↪ROBOT_OUTCOME_VALUE ]


def display_avg_col_value_per_outcome( compare_col_name ):
    x_axis = [0, 1]
    y_axis = [
```

```
        len(bids_bidders_humans) / len(bids_bidders_humans[compare_col_name].
↪unique()),
        len(bids_bidders_robots) / len(bids_bidders_robots[compare_col_name].
↪unique())
    ]
    fig, ax = plt.subplots()
    ax.bar(x_axis, y_axis)
    ax.set_title('average unique {0}-s by human / robot (human=0, robot=1)'.
↪format(compare_col_name))
    ax.set_xticks(x_axis)
    plt.show()
    print('human: {:.2f}; robot: {:.2f}\n---\n'.format(*y_axis))

# del bids_bidders_robots
# del bids_bidders_humans
```

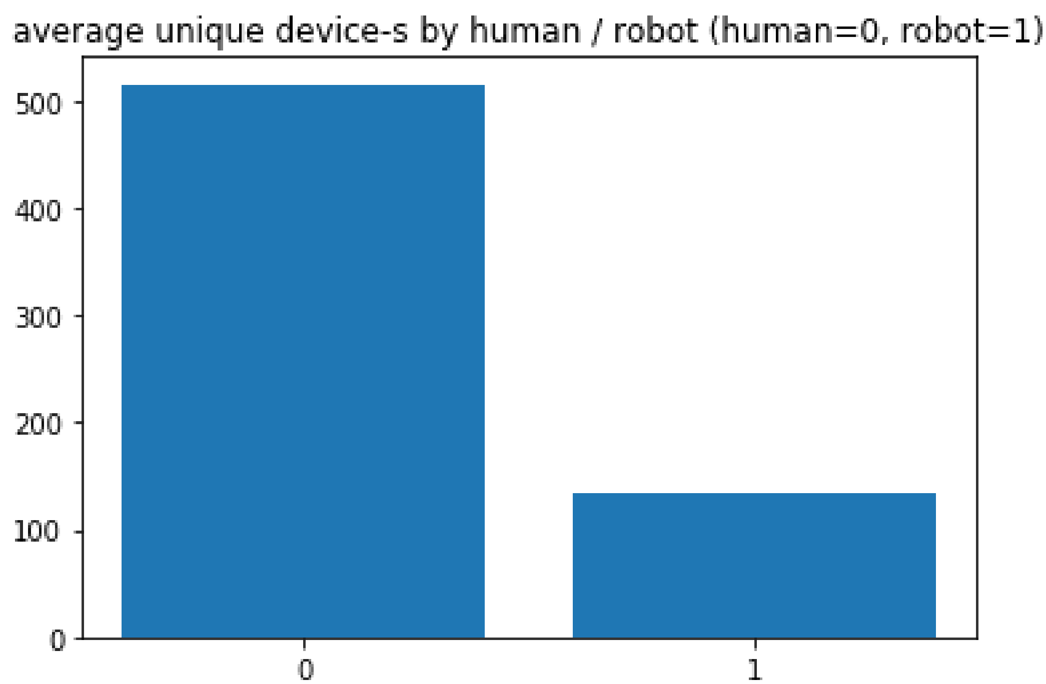
[16]:
```
display_avg_col_value_per_outcome( 'bidder_id' )
display_avg_col_value_per_outcome( 'auction' )
display_avg_col_value_per_outcome( 'device' )
display_avg_col_value_per_outcome( 'ip' )
display_avg_col_value_per_outcome( 'url' )
```



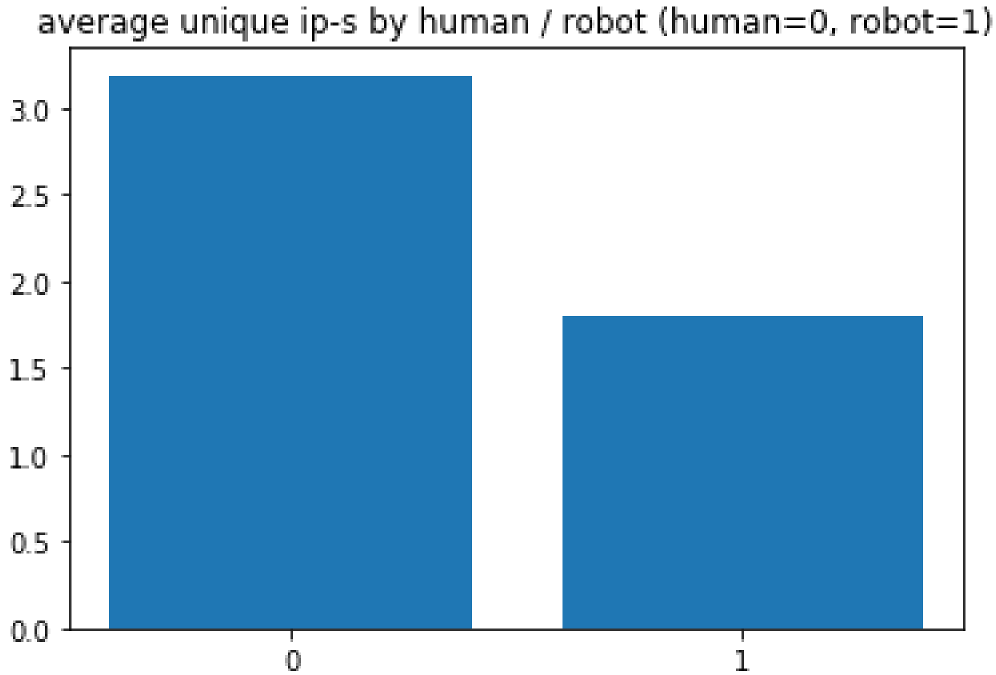

```
human: 1412.96; robot: 4001.49
---
```



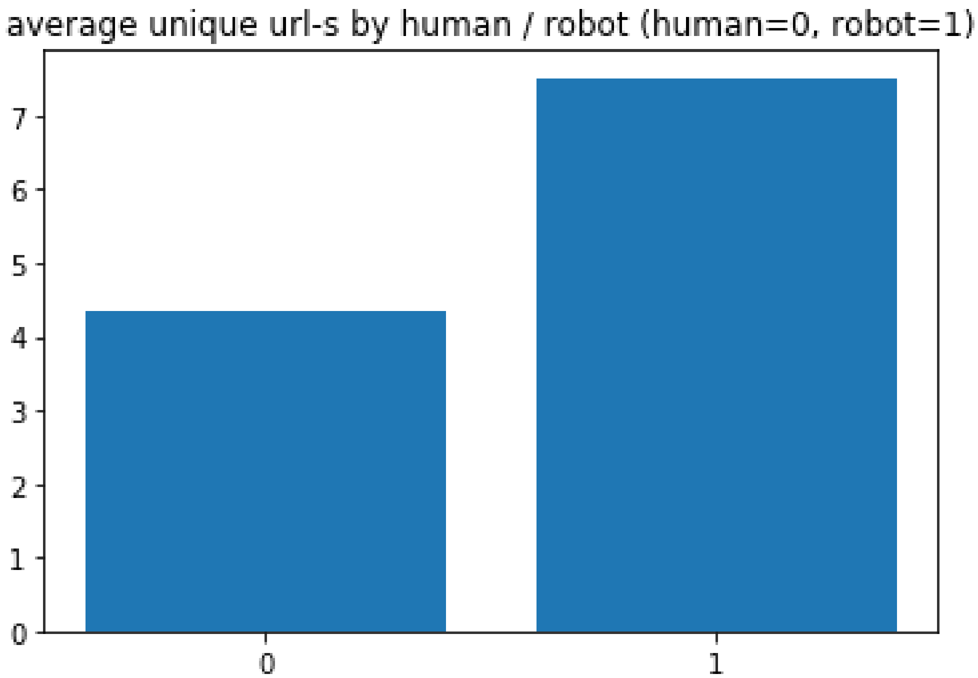

```
human: 217.18; robot: 64.50
---
```



```
human: 515.70; robot: 134.16
---
```




```
human: 3.19; robot: 1.79
---
```




```
human: 4.36; robot: 7.52
---
```

[17]:
```
# Time differences for bids made by every single bidder
bids_bidders = bids_bidders.sort_values(by=['time'])
bids_bidders['tdiff'] = bids_bidders.groupby(
    by='bidder_id')['time'].transform(pd.Series.diff)
```

[18]:
```
# Amount of bids made by single bidder for every single auction
bids_per_auction = bids_bidders.groupby(by=['auction', 'bidder_id']).size()

bids_per_auction = bids_per_auction.to_frame()

# display(bids_per_auction)
```

[19]:
```
# Rate of bidders per country which are bots
bots_per_country = bids_bidders_robots.groupby(by='country').size()
all_bidders_per_country = bids_bidders.groupby(by='country').size()
bots_per_country_rate = bots_per_country / all_bidders_per_country

# display(bots_per_country_rate)

# NaNs == no bidders at all in that country
# display(bots_per_country_rate.isnull().sum())  # 7
```

```
bots_per_country_rate = bots_per_country_rate.fillna(0)

bots_per_country_rate = bots_per_country_rate.to_frame()

# display(bots_per_country_rate)
```

[20]:
```
# Rate of bidders per kind of device which are bots
# Note: all of the devices are 'phones'

bots_per_device = bids_bidders_robots.groupby(by='device').size()
all_bidders_per_device = bids_bidders.groupby(by='device').size()
bots_per_device_rate = bots_per_device / all_bidders_per_device

# display(bots_per_device_rate)

# NaNs == no bidders at all for that type of phone
bots_per_device_rate = bots_per_device_rate.fillna(0)

bots_per_device_rate = bots_per_device_rate.to_frame()

# display(bots_per_device_rate)
```

[21]:
```
# Entropy: avg rate at which information is produced by a stochastic source of␣
 ↪data

def calculate_entropy( pd_series ):
    probabilities = pd_series.value_counts() / pd_series.index.size
    entropy = sstats.entropy( probabilities )
    return entropy

# For each user, what is the mean entropy of urls for each auction?

# ent
print('calculating entropy...')
bidder_auctions = bids_bidders.groupby(by=['auction', 'bidder_id'])
auction_urls_ent = bidder_auctions['url'].apply( calculate_entropy )
# display(auction_urls_ent)

# mean ent
print('calculating entropy 2 (mean by unique bidder)...')
auction_urls_ent = auction_urls_ent.groupby(by='bidder_id').mean().reset_index()
# display(auction_urls_ent)
```

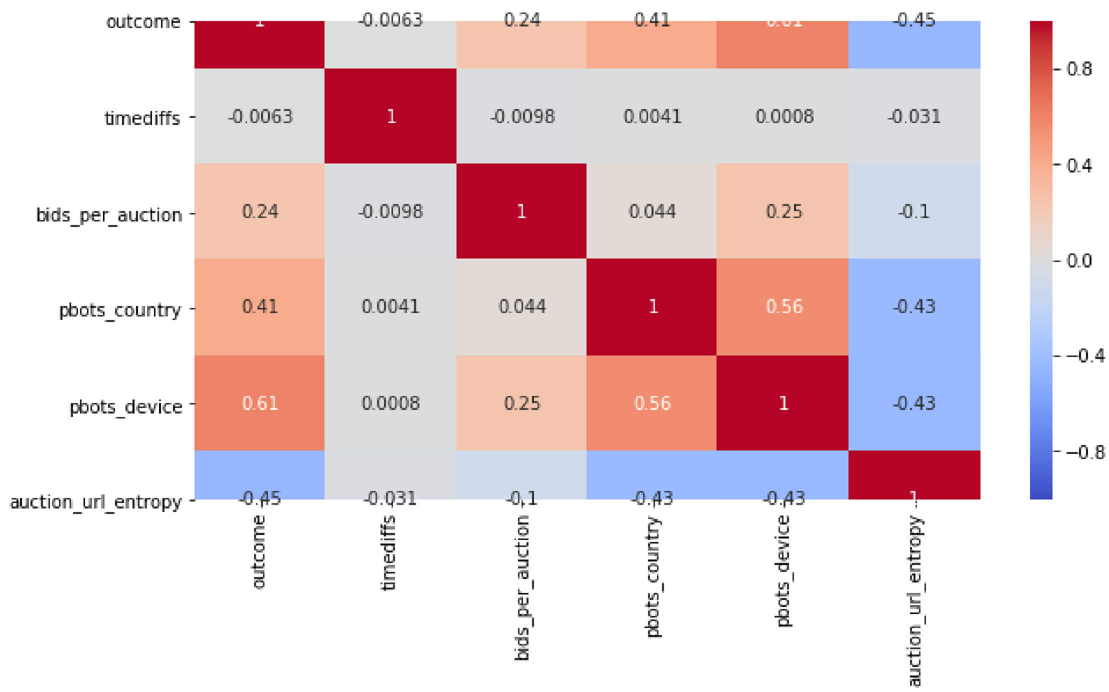
```
calculating entropy...
calculating entropy 2 (mean by unique bidder)...
```

[22]:
```
# Merge new features to bids_bidders df
```

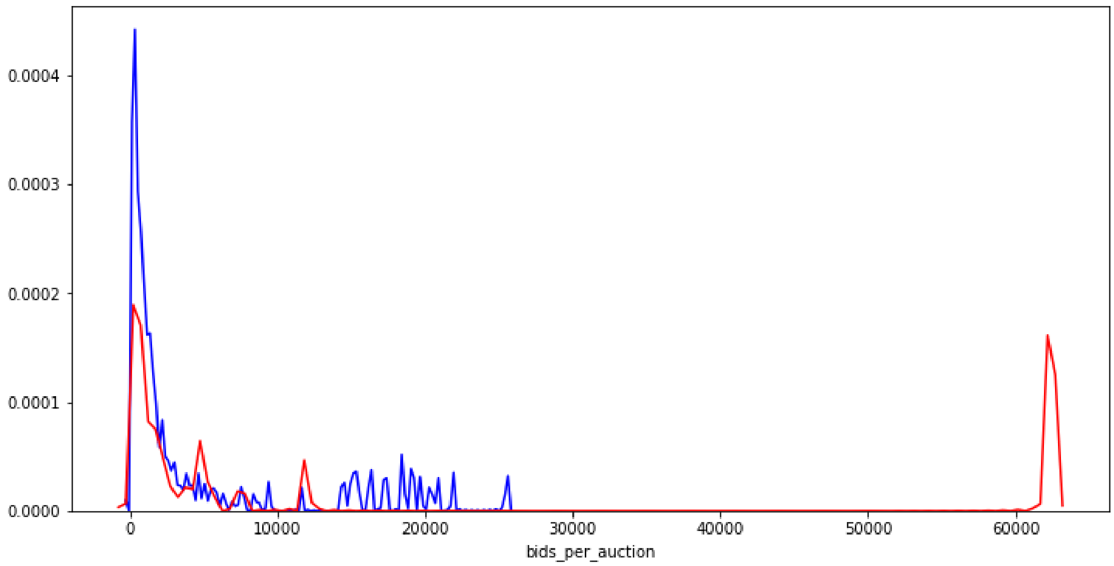
```
bids_bidders = pd.merge( bids_bidders, bids_per_auction, on=['auction',␣
↪'bidder_id'], how='left' )
bids_bidders = pd.merge( bids_bidders, bots_per_country_rate, on=['country'],␣
↪how='left' )
bids_bidders = pd.merge( bids_bidders, bots_per_device_rate, on=['device'],␣
↪how='left' )
bids_bidders = pd.merge( bids_bidders, auction_urls_ent, on=['bidder_id'],␣
↪how='left' )

bids_bidders.columns = [
    'bidder_id', 'payment_account', 'address', 'outcome', 'bid_id', 'auction',␣
↪'merchandise', 'device', 'time', 'country', 'ip', 'url',  # old ones
    'timediffs', 'bids_per_auction', 'pbots_country', 'pbots_device',␣
↪'auction_url_entropy'  # new ones
]
```

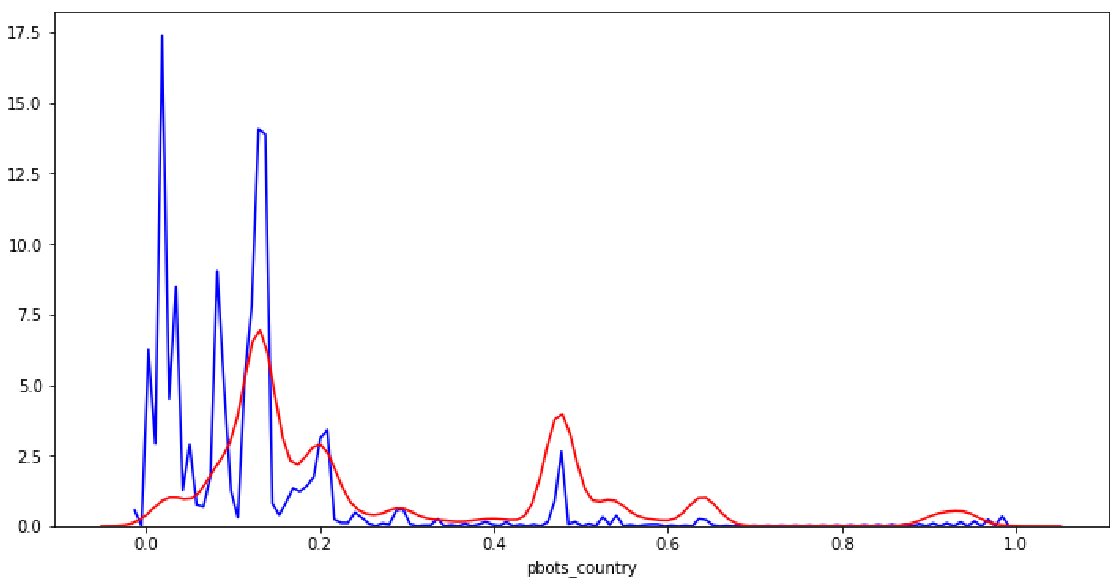
[68]:
```
# Select feature for further use

bids_bidders_newfeat = pd.concat(
    [bids_bidders.iloc[:, 3], bids_bidders.iloc[:, -5:]],
    axis=1, sort=False
)

bids_bidders_newfeat = bids_bidders_newfeat.fillna( 0 )

# Save cleaned df just in case
NEW_FEATURES_BIDS_BIDDERS_DF_FILEPATH = 'data/bids_bidders.csv'
bids_bidders_newfeat.to_csv( NEW_FEATURES_BIDS_BIDDERS_DF_FILEPATH )
```

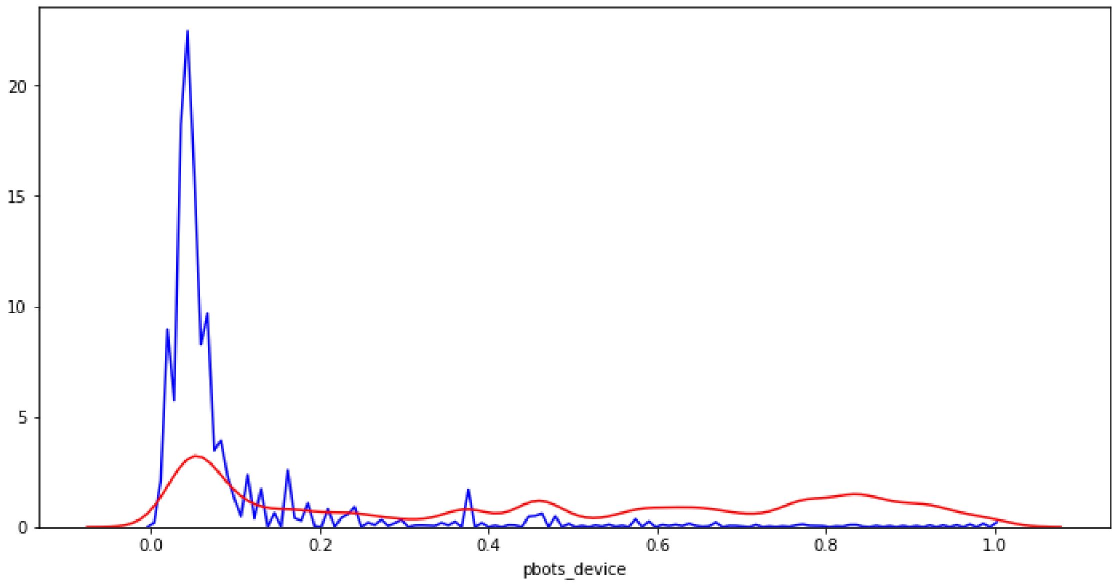
[69]:
```
# Overview Pearson correlation

corr = bids_bidders_newfeat.corr()
fig = plt.figure( figsize=(10, 5) )
sns.heatmap(
    corr,
    vmax=1, vmin=-1,
    cmap='coolwarm',
    annot=True
)
plt.show()
```



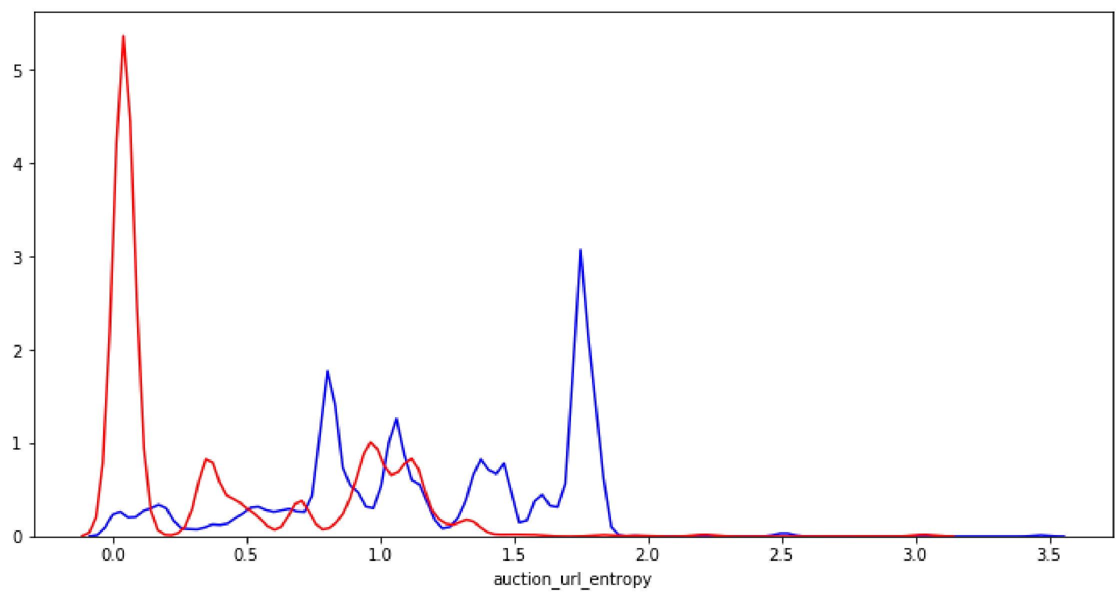
[73]:
```
# Overview distribution of newly created features

def draw_human_vs_bots_col_values( dataset_df, col_name ):
    fig, ax_0 = plt.subplots( 1, 1, figsize=(12, 6) )

    sns.distplot(
        dataset_df[ dataset_df['outcome'] == 0.0 ][col_name],
        hist=False, kde=True, color='blue',
        ax=ax_0
    )
    sns.distplot(
        dataset_df[ dataset_df['outcome'] == 1.0 ][col_name],
        hist=False, kde=True, color='red',
        ax=ax_0
    )

    plt.show()

draw_human_vs_bots_col_values( bids_bidders_newfeat, 'bids_per_auction' )
draw_human_vs_bots_col_values( bids_bidders_newfeat, 'pbots_country' )
draw_human_vs_bots_col_values( bids_bidders_newfeat, 'pbots_device' )
draw_human_vs_bots_col_values( bids_bidders_newfeat, 'auction_url_entropy' )
draw_human_vs_bots_col_values( bids_bidders_newfeat, 'timediffs' )
```


0.0004
0.0003
0.0002
0.0001
0.0000
0
10000
20000
30000
40000
50000
60000
bids_per_auction

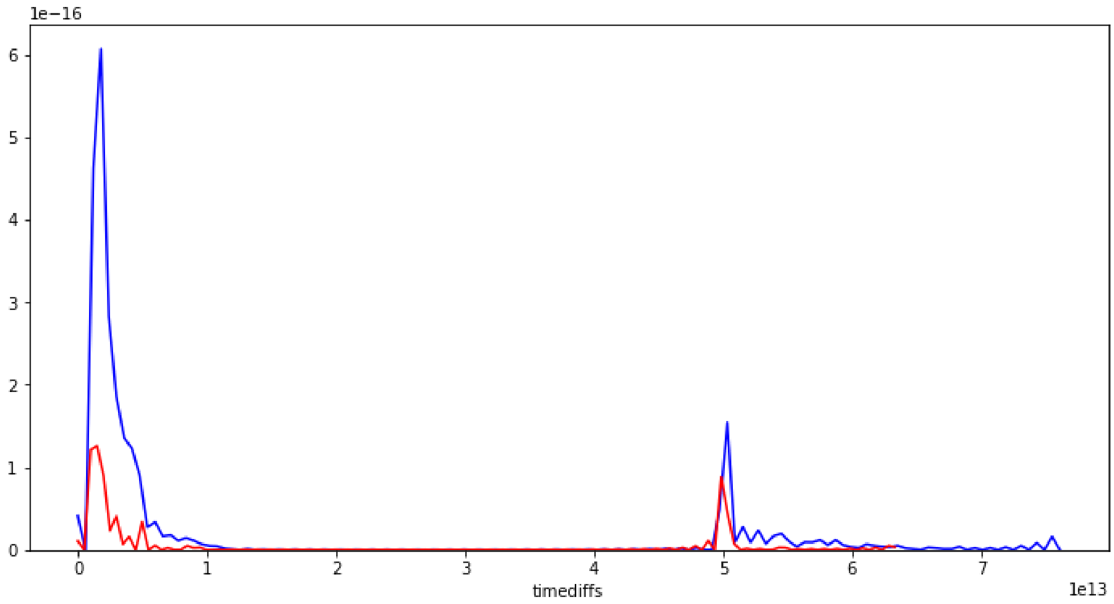

17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
pbots_country



20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0
pbots_device



5
4
3
2
1
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
auction_url_entropy



[85]:
```
# src: crystalxs

# Assume all data is human - what is the model accuracy?

all_humans_cnt = len(bids_bidders_newfeat[bids_bidders_newfeat['outcome'] == 0.
 ↪0])
all_cnt = all_humans_cnt + len(␣
 ↪bids_bidders_newfeat[bids_bidders_newfeat['outcome'] == 1] )
display(all_humans_cnt / all_cnt)
```

0.8656835878368844

[80]:
```
# Build a baseline model

X_bids_bidders_newfeat = bids_bidders_newfeat

bid_train, bid_test = train_test_split( X_bids_bidders_newfeat )
```

[83]:
```
bots_train = bid_train.loc[bid_train.outcome == 1]
human_train = bid_train.loc[bid_train.outcome == 0]
human_sample = human_train.sample(n=len(bots_train))
bid_train_balance = pd.concat([bots_train, human_sample])

y_train = bid_train_balance['outcome']
X_train = bid_train_balance.iloc[:, -5:]
y_test = bid_test['outcome']
X_test = bid_test.iloc[:, -5:]
```

[93]:
```python
# Try out Sklearn.GradientBoostingClf

gbc_model = GradientBoostingClassifier(
    n_estimators=25,
    max_depth=5,
    max_leaf_nodes=10,
    max_features='sqrt'
)

gbc_model.fit(X_train, y_train)

# score

display( accuracy_score( gbc_model.predict(X_test), y_test ) )
```

```
0.9028405839419864
```

```
0.9028405839419864
```

[90]:
```python
# Check out feature importances

display(X_train.columns)
display(gbc_model.feature_importances_)
```

```
Index(['timediffs', 'bids_per_auction', 'pbots_country', 'pbots_device',
       'auction_url_entropy'],
      dtype='object')
```

```
array([0.02811246, 0.0784251 , 0.11837607, 0.31911007, 0.45597631])
```

[101]:
```python
# Precision/Recall/F1/Support

print(
    classification_report( y_test, gbc_model.predict(X_test) )
)
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.90      0.94    663806
         1.0       0.59      0.90      0.72    103325

    accuracy                           0.90    767131
   macro avg       0.79      0.90      0.83    767131
weighted avg       0.93      0.90      0.91    767131
```

[102]:
```python
# AUC

gbc_probab = gbc_model.predict_proba( X_test )[:, 1]
fpr_gb, tpr_gb, _gb = roc_curve(y_test, gbc_probab)
roc_gb_auc = auc(fpr_gb, tpr_gb)

plt.figure( figsize=(5, 5) )
plt.plot(
    fpr_gb, tpr_gb,
    label='GB ROC curve (area = {0:.2f})'.format(roc_gb_auc)
)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.005])
plt.xlabel('FP Rate')
plt.ylabel('TP Rate')
plt.title('ROC')
plt.legend()
plt.show()
```