Predicting Illegal dumpsites

This notebook will contain multiple model tests.

1. First test will be a similar one to what I have done before. This time, I will have more data (and from more continents) as uploaded here. The non-dumpsite data in this dataset has been created with the Foursquare API.
2. The second test will be a more significant one. It will be based off on more carefully created random points around the dumpsites, and extra variables such as number of venues and average distance to venues within a certain radius. This dataset lives here.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

```
"""
required libraries imported
"""

! pip install category_encoders --quiet

import category_encoders as ce

import os, io, sys, random, time, pprint, math, csv
import numpy as np
from numpy import save, load
from itertools import chain
from pylab import rcParams
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import one_hot
from tensorflow.keras.callbacks import LambdaCallback, Callback, ModelCheckpoint, EarlyStopping, LearningRateScheduler
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.models import load_model
from tensorflow.keras import Model, Sequential
from tensorflow.keras import backend, regularizers
from tensorflow.keras.layers import Activation, Reshape, Dense, Embedding, Dropout, Input, BatchNormalization, concatenate, Flatten, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam, RMSprop, SGD, Adadelta, Adagrad, Adamax
from tensorflow_addons.optimizers import AdamW
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.keras.utils import to_categorical, plot_model
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, RepeatedStratifiedKFold
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
```

```
     |████████████████████████████████| 81kB 2.5MB/s

/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

ABANDONED Test 1 - not a lot of data

First test on previously available data. We are not using this test 1 anymore because the dataset was very old with very few variables and examples.

If you want to look at test 1 code (why?), then contact Ram.

Data Preparation

```
#@title ## Loading Data
# training data
final = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_dumpSites.csv')
print(f"final has shape {final.shape}")

final_random = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_randomPoints_1km.csv')
print(f"final_random has shape {final_random.shape}")

# testing data
final_test = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_dumpSites_Test.csv')
print(f"final_test has shape {final_test.shape}")

final_random_test = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_randomPoints_1km_Test.csv')
print(f"final_random_test has shape {final_random_test.shape}")
```

```
final has shape (44881, 20)
final_random has shape (44545, 20)
final_test has shape (11217, 20)
final_random_test has shape (11133, 20)
```

```
(list(final.columns) == list(final_random.columns) == list(final_test.columns) == list(final_random_test.columns))

# all columns are same, no need to rename anything
```

```
True
```

Exploring the data

```
final.head()
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1st
0	57837	-26.196481	28.050331	Africa	25.118444	12718.73	0.840314	332	24	706.25	Fast Rest
1	57831	-28.466159	28.855491	Africa	16.073944	1704.99	0.714321	1500	0	3000.00	Non
2	57830	-28.514575	28.820170	Africa	33.621466	3842.15	0.494658	496	4	620.00	Auto Wor
3	57829	-29.947538	30.915913	Africa	32.739744	11928.59	0.327302	61	1	61.00	Sho
4	57817	-26.376726	27.300703	Africa	6.813308	2.17	0.493056	1500	0	3000.00	Non

```
final_random.head()
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues
0	57836	-26.174340	27.985381	Africa	48.082264	4436.40	0.596904	420	5	607.600000
1	57831	-28.473954	28.859991	Africa	150.000000	2167.54	0.636909	1500	0	3000.000000
2	57830	-28.506780	28.815669	Africa	53.892441	3341.90	0.561918	634	3	757.666667
3	57828	-26.247996	27.703924	Africa	21.060757	5.65	0.499462	1500	0	3000.000000
4	57813	-26.022699	28.006938	Africa	10.986695	3039.87	0.360769	317	41	744.243902

```
final_test.head()
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1st
0	57836	-26.182135	27.989881	Africa	44.673641	3553.60	0.317897	341	5	654.2	Bar
1	57828	-26.243496	27.696129	Africa	1.472705	314.45	0.923777	1500	0	3000.0	Non
2	57809	-26.241132	27.815810	Africa	9.143493	20422.59	0.343147	411	5	557.4	Fish Sho
3	57764	-25.495246	31.179170	Africa	15.193524	1.18	0.499033	1500	0	3000.0	Non
4	57755	-26.230469	27.846345	Africa	2.803380	8480.55	0.315502	650	5	784.0	Pub

```
final_random_test.head()
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues
0	57837	-26.191981	28.042536	Africa	9.128174	45583.50	0.662325	258	44	687.136364
1	57829	-29.938537	30.915913	Africa	2.136312	7893.65	0.622818	1500	0	3000.000000
2	57817	-26.385727	27.300703	Africa	150.000000	2.17	0.644543	1500	0	3000.000000
3	57777	-33.582348	18.468104	Africa	150.000000	9.18	0.440028	704	1	704.000000
4	57604	-33.905952	25.598207	Africa	6.882133	301.96	0.978314	375	4	642.000000

Categorical Variable Preparation

There are two categorical variables

1. Continent

2. Nearest Venue Categories (5) and most frequent venue categories (5).

Since the cardinality of the continent is less than 15, a one-hot encoding would be good.

The Venue categories variables will be transformed as per the different tests in the following sections.

```
final.shape, final.isnull().sum()
```

```
((44881, 20), TrashOutID                    0
 Lat                           0
 Lon                           0
 Continent                    40
 Distance to Road              0
 Population Density          120
 Population gradient         120
 Distance to Nearest Venue     0
 Number of Venues              0
 Avg Dist to Venues            0
 1stMostFreq                   0
 2ndMostFreq                   0
 3rdMostFreq                   0
 4thMostFreq                   0
 5thMostFreq                   0
 1stClosest                    0
 2ndClosest                    0
 3rdClosest                    0
 4thClosest                    0
 5thClosest                    0
 dtype: int64)
```

```
final_random.shape, final_random.isnull().sum()
```

```
((44545, 20), TrashOutID                    0
 Lat                           0
 Lon                           0
 Continent                    40
 Distance to Road              0
 Population Density          314
 Population gradient         314
 Distance to Nearest Venue     0
 Number of Venues              0
 Avg Dist to Venues            0
 1stMostFreq                   0
 2ndMostFreq                   0
 3rdMostFreq                   0
 4thMostFreq                   0
 5thMostFreq                   0
 1stClosest                    0
 2ndClosest                    0
 3rdClosest                    0
 4thClosest                    0
 5thClosest                    0
 dtype: int64)
```

```
set(final['Continent']), set(final_random['Continent'])
```

```
({'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America',
  nan},
 {'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America',
  nan})
```

```
#@markdown ## Removing NaN values
"""
removing null values from Continent variables
"""

final = final[final['Continent'].notna()]
final_random = final_random[final_random['Continent'].notna()]

final_test = final_test[final_test['Continent'].notna()]
final_random_test = final_random_test[final_random_test['Continent'].notna()]
```

```
final['Continent'].isnull().any(), final_random['Continent'].isnull().any(), final_test['Continent'].isnull().any(), final_random_test['Continent'].isnull().any()

# great, null values removed from Continent of both training and test sets
```

```
(False, False, False, False)
```

```
set(final['Continent']), set(final_random['Continent']), set(final_test['Continent']), set(final_random_test['Continent'])
```

```
({'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America'},
 {'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America'},
 {'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America'},
 {'Africa',
  'Asia',
  'Australia',
  'Europe',
  'North America',
  'Oceania',
  'South America'})
```

```
print("Oceania examples in training set: ", len(final[final['Continent'] == 'Oceania']), len(final_random[final_random['Continent'] == 'Oceania']))
print("Oceania examples in test set: ", len(final_test[final_test['Continent'] == 'Oceania']), len(final_random_test[final_random_test['Continent'] == 'Oceania']))
```

```
Oceania examples in training set:  9 9
Oceania examples in test set:  2 2
```

```
#@markdown ## Converting continent to one-hot encoding
"""
making the continent into one-hot encoding for both datasets
"""

train_dumpsite_dummies = pd.get_dummies(final['Continent'])
final = pd.concat([final.drop('Continent', axis=1), train_dumpsite_dummies], axis=1)

train_nondumpsite_dummies = pd.get_dummies(final_random['Continent'])
final_random = pd.concat([final_random.drop('Continent', axis=1), train_nondumpsite_dummies], axis=1)

test_dumpsite_dummies = pd.get_dummies(final_test['Continent'])
final_test = pd.concat([final_test.drop('Continent', axis=1), test_dumpsite_dummies], axis=1)

test_nondumpsite_dummies = pd.get_dummies(final_random_test['Continent'])
final_random_test = pd.concat([final_random_test.drop('Continent', axis=1), test_nondumpsite_dummies], axis=1)
```

```
#@markdown ## Adding target labels
"""
adding target labels
"""

if 'target' not in final.columns + final_random.columns:
    final['target'] = 1
    final_random['target'] = 0

if 'target' not in final_test.columns + final_random_test.columns:
    final_test['target'] = 1
    final_random_test['target'] = 0
```

Test 2 - using venue encodings

This second test will be a more significant one. It will be based off on more carefully created random points around the dumpsites, and extra variables such as number of venues, average distance to venues, 5 nearest venue categories, 5 most frequent venue categories, distance to nearest venue, and a categorial label for distance to roads. The dataset for the extra variables lives here.

Test 2.0 - Leave One Out Encoding on venue categories (all 10 variables).

```
# combined data of dumpsites and non-dumpsites in train and test sets

data = pd.concat([final, final_random], axis=0)

data_test = pd.concat([final_test, final_random_test], axis=0)
```

```
"""
Applying an encoding onto the 10 different columns (#thMostFreq and #thClosest)
 categorical variables
"""

columns = ['1stMostFreq', '2ndMostFreq', '3rdMostFreq', '4thMostFreq', '5thMostFreq', '1stClosest', '2ndClosest', '3rdClosest', '4thClosest', '5thClosest']

for col in columns:
  # training
  leaveoneoutenc = ce.LeaveOneOutEncoder(return_df=True)
  leaveoneoutenc.fit(X=data[col], y=data['target'])

  data[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data[col])
  data.drop([col], axis=1, inplace=True)

  #testing
  data_test[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data_test[col])
  data_test.drop([col], axis=1, inplace=True)
  data_test.head()

data.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)
data_test.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)

data.head()
```

```
"""
removing any null values in other variables
"""

print(data.shape, data_test.shape)

data.dropna(inplace=True)
data_test.dropna(inplace=True)

data.shape, data_test.shape
```

```
((88916, 24), (22222, 24))
```

Building a Model (leave one out encoding)

```
"""
training and testing separation
"""

X = data.drop(['target'], axis=1).to_numpy()
Y = data['target'].to_numpy()

test_x = data_test.drop(['target'], axis=1).to_numpy()
test_y = data_test['target'].to_numpy()

using_kfold = True

if not using_kfold:
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                      stratify=Y,
                                                      random_state=111)
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")

else:
  X_train = X
  Y_train = Y
  print(f"X_train is {X_train.shape}", f"Y_train is {Y_train.shape}")
```

```
X_train is (88916, 23) Y_train is (88916,)
```

```
"""
normalizing the continuous variables
"""

scaler = StandardScaler()

scaler.fit(X_train[:, :6])

X_train[:, :6] = scaler.transform(X_train[:, :6])

if not using_kfold:
  X_test[:, :6] = scaler.transform(X_test[:, :6])
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")
else:
  X_train.shape
```

```
(88916, 23)
```

```
"""
building the model
"""

x_input = Input(shape=(X_train.shape[-1],), name='input_layer')

x = Dense(units=32, activation='relu', name=f'dense_1',
          kernel_regularizer=regularizers.l2(1e-4),
          bias_regularizer=regularizers.l2(1e-4),
          activity_regularizer=regularizers.l2(1e-4))(x_input)

# x = Dropout(rate=0.2, name=f'dropout_1')(x)

for i, unit in enumerate([32, 32], start=2):
    x = Dense(units=unit, activation='relu', name=f'dense_{i}',
              kernel_regularizer=regularizers.l2(1e-4),
              bias_regularizer=regularizers.l2(1e-4),
              activity_regularizer=regularizers.l2(1e-4))(x)

    # x = BatchNormalization(name=f'batchnorm_{i}')(x)
    # x = Dropout(rate=0.2, name=f'dropout_{i}')(x)

out = Dense(units=1, activation='sigmoid', name='output_layer')(x)

model = Model(inputs=x_input, outputs=out, name='Dumpsite_Prediction_Model')

# saving the model graph and seeing the architecture
plot_model(model, show_shapes=True, to_file='/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/leaveoneoutencoding/nn_graph.png')

model.summary()
```

```
Model: "Dumpsite_Prediction_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 23)]              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                768
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
output_layer (Dense)         (None, 1)                 33
=================================================================
Total params: 2,913
Trainable params: 2,913
Non-trainable params: 0
_________________________________________________________________
```

Using K-Fold for better learning

```
"""
using the sklearn library to split the training data
into folds of training and validation splits
"""

n_split=10

fold = StratifiedShuffleSplit(n_splits=n_split, test_size=0.15,
                              random_state=111)

train_acc=[]
train_loss=[]
val_acc=[]
val_loss=[]
hists = []

fold.get_n_splits(X_train, Y_train)
```

```
10
```

Training for LeaveOneOut encoding

```python
"""
compiling and fitting
"""

filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/frequencyencoding/curr_model.h5'

checkpoint = ModelCheckpoint(filepath=filepath_curr, monitor='val_accuracy', save_best_only=True)

opts = Adam(learning_rate=0.001, epsilon=1e-8, decay=0.0001)

# model will begin training with previously trained weights
try:
    model.load_weights(filepath_curr)
    print('Same architecture as before.\n\n')
except:
    print('Model architecture has been changed. No weights loaded.\n\n')

i = 1 # counter for print statement

if not using_kfold:
  model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])
  history = model.fit(
      X_train,
      Y_train,
      batch_size=128,
      epochs=75,
      validation_data=(X_test, Y_test),
      callbacks=[checkpoint],
      verbose=2
  )
else:
  for train_index, val_index in fold.split(X_train, Y_train):

      train_x, val_x = X_train[train_index], X_train[val_index]
      train_y, val_y = Y_train[train_index], Y_train[val_index]

      model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])

      history = model.fit(
          train_x,
          train_y,
          batch_size=128,
          epochs=75,
          validation_data=(val_x, val_y),
          callbacks=[checkpoint],
          verbose=0
      )

      curr_train_acc = round(model.evaluate(train_x, train_y, verbose=0)[1]*100, 2)
      curr_val_acc = round(model.evaluate(val_x, val_y, verbose=0)[1]*100, 2)

      curr_train_loss = model.evaluate(train_x, train_y, verbose=0)[0]
      curr_val_loss = model.evaluate(val_x, val_y, verbose=0)[0]

      print("-"*75)
      print(f"Fold {i}: train acc = {curr_train_acc}%")
      print(f"Fold {i}: val acc = {curr_val_acc}%")
      print("-"*75 + "\n\n")

      train_acc.append(curr_train_acc)
      val_acc.append(curr_val_acc)
      train_loss.append(curr_train_loss)
      val_loss.append(curr_val_loss)

      i += 1
```

```
Same architecture as before.


---------------------------------------------------------------------------
Fold 1: train acc = 71.33%
Fold 1: val acc = 70.66%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 2: train acc = 71.46%
Fold 2: val acc = 70.03%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 3: train acc = 71.56%
Fold 3: val acc = 70.75%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 4: train acc = 71.53%
Fold 4: val acc = 71.48%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 5: train acc = 71.56%
Fold 5: val acc = 71.19%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 6: train acc = 71.62%
Fold 6: val acc = 71.32%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 7: train acc = 71.6%
Fold 7: val acc = 71.46%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 8: train acc = 71.64%
Fold 8: val acc = 71.39%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 9: train acc = 71.64%
Fold 9: val acc = 71.83%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 10: train acc = 71.63%
Fold 10: val acc = 71.52%
---------------------------------------------------------------------------
```

Evaluating average accuracy and loss after K-Fold

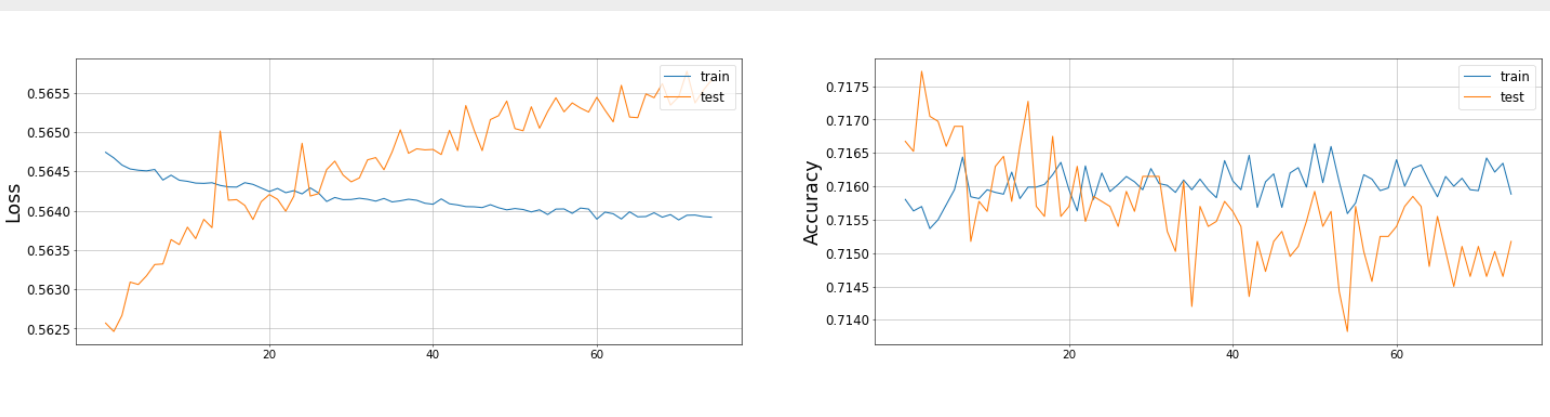
```python
if using_kfold:
  for i, v in enumerate(train_acc, start=0):
    print('-'*50)
    print(f'Fold {i+1} - Train Loss: {train_loss[i]} - Train Accuracy: {train_acc[i]}%')
    print(f'Fold {i+1} - Validation Loss: {val_loss[i]} - Validation Accuracy: {val_acc[i]}%')

  print('-'*50)

  print('\n\nAverage scores for all folds:\n\n')
  # training
  print(f'Train Accuracy: {round(np.mean(train_acc), 2)} +- {round(np.std(train_acc), 2)}')
  print(f'Train Loss: {round(np.mean(train_loss), 3)}')
  # validation
  print(f'\n\nTest Accuracy: {round(np.mean(val_acc), 2)} +- {round(np.std(val_acc), 2)}')
  print(f'Test Loss: {round(np.mean(val_loss), 3)}')
```

```
--------------------------------------------------
Fold 1 - Train Loss: 0.5680950880050659 - Train Accuracy: 71.33%
Fold 1 - Validation Loss: 0.5826188325881958 - Validation Accuracy: 70.66%
--------------------------------------------------
Fold 2 - Train Loss: 0.5660271048545837 - Train Accuracy: 71.46%
Fold 2 - Validation Loss: 0.5841373205184937 - Validation Accuracy: 70.03%
--------------------------------------------------
Fold 3 - Train Loss: 0.5650194883346558 - Train Accuracy: 71.56%
Fold 3 - Validation Loss: 0.5743318200111389 - Validation Accuracy: 70.75%
--------------------------------------------------
Fold 4 - Train Loss: 0.5657946467399597 - Train Accuracy: 71.53%
Fold 4 - Validation Loss: 0.5652322769165039 - Validation Accuracy: 71.48%
--------------------------------------------------
Fold 5 - Train Loss: 0.5649651885032654 - Train Accuracy: 71.56%
Fold 5 - Validation Loss: 0.5673887729644775 - Validation Accuracy: 71.19%
--------------------------------------------------
Fold 6 - Train Loss: 0.5642725825309753 - Train Accuracy: 71.62%
Fold 6 - Validation Loss: 0.5678231120109558 - Validation Accuracy: 71.32%
--------------------------------------------------
Fold 7 - Train Loss: 0.5639496445655823 - Train Accuracy: 71.6%
Fold 7 - Validation Loss: 0.5674647092819214 - Validation Accuracy: 71.46%
--------------------------------------------------
Fold 8 - Train Loss: 0.5635033845901489 - Train Accuracy: 71.64%
Fold 8 - Validation Loss: 0.5686129331588745 - Validation Accuracy: 71.39%
--------------------------------------------------
Fold 9 - Train Loss: 0.5639879703521729 - Train Accuracy: 71.64%
Fold 9 - Validation Loss: 0.5646975040435791 - Validation Accuracy: 71.83%
--------------------------------------------------
Fold 10 - Train Loss: 0.563536524772644 - Train Accuracy: 71.63%
Fold 10 - Validation Loss: 0.5656611323356628 - Validation Accuracy: 71.52%
--------------------------------------------------


Average scores for all folds:


Train Accuracy: 71.56 +- 0.09
Train Loss: 0.565


Test Accuracy: 71.16 +- 0.5
Test Loss: 0.571
```

```python
"""
this code takes care of saving the new model only if its accuracy is better than that of the last model
"""

filepath = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/frequencyencoding/best_model.h5'
filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/frequencyencoding/curr_model.h5'


if os.path.exists(filepath):
  prev_model = load_model(filepath)
  curr_model = load_model(filepath_curr)

  prev_acc = prev_model.evaluate(test_x, test_y, verbose=0)[1]
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]

  if curr_acc > prev_acc:
    print("There was a previous model saved.\n")
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    curr_model.save(filepath)
    print('\nNew model is saved.')
  else:
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    print('Old model is kept.')

else: # if this is the first time saving the model
  model.save(filepath)
  curr_model = load_model(filepath)
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]
  print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
  print('\nFirst time model is saved.')
```

```
Previous test accuracy: 52.32%
Current test accuracy: 51.08%
Old model is kept.
```

```python
"""
looking at the train/test loss and accuracy over the epochs
"""

width_in_inches = 35
height_in_inches = 7
dots_per_inch = 50

plt.figure(
    figsize=(width_in_inches, height_in_inches), dpi=dots_per_inch)

# PICK THE FOLD
fold = 5

# plot loss during training
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['loss'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(loc='upper right', fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/frequencyencoding/train.png')

# plot accuracy during training
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['accuracy'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/frequencyencoding/test.png')

plt.show()
```




```python
"""
printing the confusion matrix
"""

preds = load_model(filepath).predict(test_x).squeeze()

assert preds.shape == test_y.shape, 'The shapes of the two tensors are unequal'

print(confusion_matrix(test_y, np.round(preds)))

print(classification_report(test_y, np.round(preds)))
```

```
[[7174 3799]
 [6733 4385]]
              precision    recall  f1-score   support

           0       0.52      0.65      0.58     10973
           1       0.54      0.39      0.45     11118

    accuracy                           0.52     22091
   macro avg       0.53      0.52      0.52     22091
weighted avg       0.53      0.52      0.52     22091
```

Test 2.2 - Mean Encoding on venue categories (all 10 variables).

```python
datam = pd.concat([final, final_random], axis=0)

datam_test = pd.concat([final_test, final_random_test], axis=0)
```

```python
# Helper Class for Testing out different encodings


class MultiColumnEncoder:
    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit_transform(self,X):
        '''
        Transforms columns of X specified in self.columns using
        given encoder. If no columns specified, transforms all
        columns in X.
        '''
        mean = X['target'].mean()
        if self.columns is not None:
            for col in self.columns:
                agg= X.groupby(col)['target'].agg(['count','mean'])
                counts=agg['count']
                means=agg['mean']
                weight=100
                smooth = (counts*means +weight *mean) / (counts + weight)
                X.loc[:,col+'_encoding'] = X[col].map(smooth)
        else:
            for col in X.columns:
                agg= X.groupby(col)['target'].agg(['count','mean'])
                counts=agg['count']
                means=agg['mean']
                weight=100
                smooth = (counts*means +weight *mean) / (counts + weight)
                X.loc[:,col+'_encoding'] = X[col].map(smooth)
        X.drop(self.columns,axis=1,inplace=True)
        return X

columns =['1stMostFreq','2ndMostFreq','3rdMostFreq','4thMostFreq','5thMostFreq','1stClosest','2ndClosest','3rdClosest',
          '4thClosest','5thClosest']
mc_class =MultiColumnEncoder(columns)
```

```python
#Applying mean encoding to venue category categorical variables


datam = mc_class.fit_transform(datam)
datam.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)

datam_test = mc_class.fit_transform(datam_test)
datam_test.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)

datam.dropna(inplace=True)
datam_test.dropna(inplace=True)

datam.head()
```

	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	Africa	Asia	Australia	Europe	North America	Ocear
0	25.118444	12718.73	0.840314	332	24	706.25	1	0	0	0	0	0
1	16.073944	1704.99	0.714321	1500	0	3000.00	1	0	0	0	0	0
2	33.621466	3842.15	0.494658	496	4	620.00	1	0	0	0	0	0
3	32.739744	11928.59	0.327302	61	1	61.00	1	0	0	0	0	0
4	6.813308	2.17	0.493056	1500	0	3000.00	1	0	0	0	0	0

Building a Model(mean Encoding)

```python
"""
training and testing separation
"""

X = datam.drop(['target'], axis=1).to_numpy()
Y = datam['target'].to_numpy()

test_x = datam_test.drop(['target'], axis=1).to_numpy()
test_y = datam_test['target'].to_numpy()

using_kfold = True

if not using_kfold:
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                      stratify=Y,
                                                      random_state=111)
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")

else:
  X_train = X
  Y_train = Y
  print(f"X_train is {X_train.shape}", f"Y_train is {Y_train.shape}")
```

```
X_train is (88916, 23) Y_train is (88916,)
```

```python
"""
normalizing the continuous variables
"""

scaler = StandardScaler()

scaler.fit(X_train[:, :6])

X_train[:, :6] = scaler.transform(X_train[:, :6])

if not using_kfold:
  X_test[:, :6] = scaler.transform(X_test[:, :6])
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")
else:
  X_train.shape
```

```
(88916, 23)
```

```python
"""
building the model
"""

x_input = Input(shape=(X_train.shape[-1],), name='input_layer')

x = Dense(units=32, activation='relu', name=f'dense_1',
          bias_regularizer=regularizers.l2(1e-2),
          activity_regularizer=regularizers.l2(1e-3))(x_input)

# x = Dropout(rate=0.5, name=f'dropout_1')(x)

for i, unit in enumerate([32, 32], start=2):
    # x = Dropout(rate=0.5, name=f'dropout_{i}')(x)
    # x = BatchNormalization(name=f'batchnorm_{i}')(x)
    x = Dense(units=unit, activation='relu', name=f'dense_{i}',
              bias_regularizer=regularizers.l2(1e-2),
              activity_regularizer=regularizers.l2(1e-3))(x)

out = Dense(units=1, activation='sigmoid', name='output_layer')(x)

model = Model(inputs=x_input, outputs=out, name='Dumpsite_Prediction_Model')

# saving the model graph and seeing the architecture
plot_model(model, show_shapes=True, to_file='/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/nn_graph.png')

model.summary()
```

```
Model: "Dumpsite_Prediction_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 23)]              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                768
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
output_layer (Dense)         (None, 1)                 33
=================================================================
Total params: 2,913
Trainable params: 2,913
Non-trainable params: 0
_________________________________________________________________
```

Using K-Fold for better learning

```python
"""
using the sklearn library to split the training data
into folds of training and validation splits
"""

n_split=2

fold = StratifiedShuffleSplit(n_splits=n_split, test_size=0.15,
                              random_state=111)

train_acc=[]
train_loss=[]
val_acc=[]
val_loss=[]

fold.get_n_splits(X_train, Y_train)
```

```
2
```

Training for Mean encoding

```python
"""
compiling and fitting
"""

filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/meanencoding/curr_model.h5'

checkpoint = ModelCheckpoint(filepath=filepath_curr, monitor='val_accuracy', save_best_only=True)

opts = Adam(learning_rate=0.001, epsilon=1e-8, decay=0.0001)

# model will begin training with previously trained weights
try:
    model.load_weights(filepath_curr)
    print('Same architecture as before.\n\n')
except:
    print('Model architecture has been changed. No weights loaded.\n\n')

i = 1 # counter for print statement

if not using_kfold:
  model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])
  history = model.fit(
      X_train,
      Y_train,
      batch_size=128,
      epochs=75,
      validation_data=(X_test, Y_test),
      callbacks=[checkpoint],
      verbose=2
  )
else:
  for train_index, val_index in fold.split(X_train, Y_train):

      train_x, val_x = X_train[train_index], X_train[val_index]
      train_y, val_y = Y_train[train_index], Y_train[val_index]

      model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])

      history = model.fit(
          train_x,
          train_y,
          batch_size=128,
          epochs=75,
          validation_data=(val_x, val_y),
          callbacks=[checkpoint],
          verbose=0
      )

      curr_train_acc = round(model.evaluate(train_x, train_y, verbose=0)[1]*100, 2)
      curr_val_acc = round(model.evaluate(val_x, val_y, verbose=0)[1]*100, 2)

      curr_train_loss = model.evaluate(train_x, train_y, verbose=0)[0]
      curr_val_loss = model.evaluate(val_x, val_y, verbose=0)[0]

      print("-"*75)
      print(f"Fold {i}: train acc = {curr_train_acc}%")
      print(f"Fold {i}: val acc = {curr_val_acc}%")
      print("-"*75 + "\n\n")

      train_acc.append(curr_train_acc)
      val_acc.append(curr_val_acc)
      train_loss.append(curr_train_loss)
      val_loss.append(curr_val_loss)

      i += 1
```

```
Model architecture has been changed. No weights loaded.


---------------------------------------------------------------------------
Fold 1: train acc = 74.97%
Fold 1: val acc = 74.82%
---------------------------------------------------------------------------


---------------------------------------------------------------------------
Fold 2: train acc = 75.53%
Fold 2: val acc = 74.6%
---------------------------------------------------------------------------
```

Evaluating average accuracy and loss after K-Fold

```python
if using_kfold:
  for i, v in enumerate(train_acc, start=0):
    print('-'*50)
    print(f'Fold {i+1} - Train Loss: {train_loss[i]} - Train Accuracy: {train_acc[i]}%')
    print(f'Fold {i+1} - Validation Loss: {val_loss[i]} - Validation Accuracy: {val_acc[i]}%')

  print('-'*50)

  print('\n\nAverage scores for all folds:\n\n')
  # training
  print(f'Train Accuracy: {round(np.mean(train_acc), 2)} +- {round(np.std(train_acc), 2)}')
  print(f'Train Loss: {round(np.mean(train_loss), 3)}')
  # validation
  print(f'\n\nTest Accuracy: {round(np.mean(val_acc), 2)} +- {round(np.std(val_acc), 2)}')
  print(f'Test Loss: {round(np.mean(val_loss), 3)}')
```

```
--------------------------------------------------
Fold 1 - Train Loss: 0.511714518070221 - Train Accuracy: 74.97%
Fold 1 - Validation Loss: 0.5154419541358948 - Validation Accuracy: 74.82%
--------------------------------------------------
Fold 2 - Train Loss: 0.5039599537849426 - Train Accuracy: 75.53%
Fold 2 - Validation Loss: 0.5196113586425781 - Validation Accuracy: 74.6%
--------------------------------------------------


Average scores for all folds:


Train Accuracy: 75.25 +- 0.28
Train Loss: 0.508


Test Accuracy: 74.71 +- 0.11
Test Loss: 0.518
```

```python
"""
this code takes care of saving the new model only if its accuracy is better than that of the last model
"""

filepath = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/best_model.h5'
filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/curr_model.h5'


if os.path.exists(filepath):
  prev_model = load_model(filepath)
  curr_model = load_model(filepath_curr)

  prev_acc = prev_model.evaluate(test_x, test_y, verbose=0)[1]
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]

  if curr_acc > prev_acc:
    print("There was a previous model saved.\n")
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    curr_model.save(filepath)
    print('\nNew model is saved.')
  else:
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    print('Old model is kept.')

else: # if this is the first time saving the model
  model.save(filepath)
  curr_model = load_model(filepath)
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]
  print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
  print('\nFirst time model is saved.')
```

```
Previous test accuracy: 49.37%
Current test accuracy: 48.94%
Old model is kept.
```

```python
"""
printing the confusion matrix
"""

preds = load_model(filepath).predict(test_x).squeeze()

assert preds.shape == test_y.shape, 'The shapes of the two tensors are unequal'

print(confusion_matrix(test_y, np.round(preds)))

print(classification_report(test_y, np.round(preds)))
```

```
[[2409 8631]
 [2621 8561]]
              precision    recall  f1-score   support

           0       0.48      0.22      0.30     11040
           1       0.50      0.77      0.60     11182

    accuracy                           0.49     22222
   macro avg       0.49      0.49      0.45     22222
weighted avg       0.49      0.49      0.45     22222
```

```python
"""
looking at the train/test loss and accuracy over the epochs
"""

width_in_inches = 35
height_in_inches = 7
dots_per_inch = 50

plt.figure(
    figsize=(width_in_inches, height_in_inches), dpi=dots_per_inch)

# PICK THE FOLD
fold = 5

# plot loss during training
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['loss'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(loc='upper right', fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/train.png')

# plot accuracy during training
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['accuracy'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/test.png')

plt.show()
```

ABANDONED Test 3 - distance to roads ordinal encoding

ABANDONED - the performance accuracy of this variation was far below other models. The continuity of the distance to roads variable is a critically important variable for a good model.

```python
# training data
final = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_dumpSites.csv')
print(f"final has shape {final.shape}")

final_random = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_randomPoints_1km.csv')
print(f"final_random has shape {final_random.shape}")

# testing data
final_test = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_dumpSites_Test.csv')
print(f"final_test has shape {final_test.shape}")

final_random_test = pd.read_csv('/content/drive/My Drive/Official folder for Trashout/Task_1/T1_Datasets/Final Datasets/Complete_randomPoints_1km_Test.csv')
print(f"final_random_test has shape {final_random_test.shape}")
```

```
final has shape (44881, 20)
final_random has shape (44545, 20)
final_test has shape (11217, 20)
final_random_test has shape (11133, 20)
```

Generate Distance to Road Categories

Create 'roadDstBin' column based on following distance categories:

- 0 --> 0 <= x < 15,
- 1 --> 15 <= x < 70,
- 2 --> 70 <= x

Working on 4 datastes:

- final
- final_random
- final_test
- final_random_test

```python
final.loc[final['Distance to Road']<15,'RoadDstBin'] = 0
final.loc[(final['Distance to Road']>=15) & (final['Distance to Road']<70),'RoadDstBin'] = 1
final.loc[final['Distance to Road']>70,'RoadDstBin'] = 2

final
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues
0	57837	-26.196481	28.050331	Africa	25.118444	12718.73	0.840314	332	24	706.25
1	57831	-28.466159	28.855491	Africa	16.073944	1704.99	0.714321	1500	0	3000.00
2	57830	-28.514575	28.820170	Africa	33.621466	3842.15	0.494658	496	4	620.00
3	57829	-29.947538	30.915913	Africa	32.739744	11928.59	0.327302	61	1	61.00
4	57817	-26.376726	27.300703	Africa	6.813308	2.17	0.493056	1500	0	3000.00
...	...	...	...	...	...	...	...	...	...	...
44876	3271	-35.500999	-71.702774	South America	1.132428	117.51	0.000000	1500	0	3000.00
44877	2315	4.535037	-74.116149	South America	15.014083	6356.60	0.000000	332	5	536.40
44878	2280	4.534641	-74.115919	South America	22.447410	6356.60	0.000000	319	5	524.00
44879	1840	-25.524732	-49.107813	South America	9.966370	1088.07	0.772548	581	4	695.00
44880	1839	-25.524935	-49.107255	South America	3.935516	1088.07	0.772548	548	4	729.50

44881 rows × 21 columns

```python
final_random.loc[final_random['Distance to Road']<15,'RoadDstBin'] = 0
final_random.loc[(final_random['Distance to Road']>=15) & (final_random['Distance to Road']<70),'RoadDstBin'] = 1
final_random.loc[final_random['Distance to Road']>70,'RoadDstBin'] = 2

final_random
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dis Ven
0	57836	-26.174340	27.985381	Africa	48.082264	4436.40	0.596904	420	5	607.600
1	57831	-28.473954	28.859991	Africa	150.000000	2167.54	0.636909	1500	0	3000.00
2	57830	-28.506780	28.815669	Africa	53.892441	3341.90	0.561918	634	3	757.666
3	57828	-26.247996	27.703924	Africa	21.060757	5.65	0.499462	1500	0	3000.00
4	57813	-26.022699	28.006938	Africa	10.986695	3039.87	0.360769	317	41	744.243
...	...	...	...	...	...	...	...	...	...	...
44540	4702	-15.485500	-43.458295	South America	150.000000	1.17	0.000000	1500	0	3000.00
44541	4662	-30.006400	-51.185839	South America	49.525021	8626.26	0.608415	104	90	577.666
44542	3693	-1.452680	-48.513001	South America	12.322144	0.00	0.500000	565	20	904.100
44543	2280	4.530140	-74.123714	South America	52.151224	6356.60	0.000000	519	5	669.800
44544	1839	-25.524935	-49.098254	South America	13.926652	3002.78	0.630431	580	6	810.833

44545 rows × 21 columns

```python
final_test.loc[final_test['Distance to Road']<15,'RoadDstBin'] = 0
final_test.loc[(final_test['Distance to Road']>=15) & (final_test['Distance to Road']<70),'RoadDstBin'] = 1
final_test.loc[final_test['Distance to Road']>70,'RoadDstBin'] = 2

final_test
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist Venu
0	57836	-26.182135	27.989881	Africa	44.673641	3553.60	0.317897	341	5	654.20000
1	57828	-26.243496	27.696129	Africa	1.472705	314.45	0.923777	1500	0	3000.0000
2	57809	-26.241132	27.815810	Africa	9.143493	20422.59	0.343147	411	5	557.40000
3	57764	-25.495246	31.179170	Africa	15.193524	1.18	0.499033	1500	0	3000.0000
4	57755	-26.230469	27.846345	Africa	2.803380	8480.55	0.315502	650	5	784.00000
...	...	...	...	...	...	...	...	...	...	...
11212	10889	7.141387	-73.131985	South America	6.937212	4447.68	0.000000	361	2	649.50000
11213	10243	-33.042535	-71.552862	South America	6.086927	2473.31	0.015364	259	6	764.00000
11214	8894	-33.598332	-70.683921	South America	8.446978	2616.26	0.178640	267	6	812.83333
11215	8389	-16.523762	-68.112491	South America	4.406457	374.20	0.114832	253	12	744.08333
11216	4662	-30.006400	-51.194840	South America	16.070194	11048.73	0.499542	12	100	693.86000

11217 rows × 21 columns

```python
final_random_test.loc[final_random_test['Distance to Road']<15,'RoadDstBin'] = 0
final_random_test.loc[(final_random_test['Distance to Road']>=15) & (final_random_test['Distance to Road']<70),'RoadDstBin'] = 1
final_random_test.loc[final_random_test['Distance to Road']>70,'RoadDstBin'] = 2

final_random_test
```

	TrashOutID	Lat	Lon	Continent	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dis Ven
0	57837	-26.191981	28.042536	Africa	9.128174	45583.50	0.662325	258	44	687.1363
1	57829	-29.938537	30.915913	Africa	2.136312	7893.65	0.622818	1500	0	3000.000
2	57817	-26.385727	27.300703	Africa	150.000000	2.17	0.644543	1500	0	3000.000
3	57777	-33.582348	18.468104	Africa	150.000000	9.18	0.440028	704	1	704.0000
4	57604	-33.905952	25.598207	Africa	6.882133	301.96	0.978314	375	4	642.0000
...	...	...	...	...	...	...	...	...	...	...
11128	8389	-16.519261	-68.104696	South America	11.882514	374.20	0.000000	187	8	791.3750
11129	8247	-34.627745	-58.361550	South America	5.852983	10372.39	0.183187	268	71	787.6478
11130	3271	-35.505499	-71.694979	South America	150.000000	117.51	0.000000	996	1	996.0000
11131	2315	4.539538	-74.108354	South America	4.292919	6356.60	0.000000	149	4	576.2500
11132	1840	-25.524732	-49.116814	South America	11.870446	168.55	0.589477	193	5	683.8000

11133 rows × 21 columns

```python
# Drop the original Distance to Road column in each dataset
final = final.drop('Distance to Road',axis=1)
final_random = final_random.drop('Distance to Road',axis=1)
final_test = final_test.drop('Distance to Road',axis=1)
final_random_test = final_random_test.drop('Distance to Road',axis=1)
```

```python
"""
removing null values from Continent variables
"""

final = final[final['Continent'].notna()]
final_random = final_random[final_random['Continent'].notna()]

final_test = final_test[final_test['Continent'].notna()]
final_random_test = final_random_test[final_random_test['Continent'].notna()]
```

```python
final['Continent'].isnull().any(), final_random['Continent'].isnull().any(), final_test['Continent'].isnull().any(), final_random_test['Continent'].isnull().any()

# great, null values removed from Continent of all four datasets
```

```
(False, False, False, False)
```

```python
final['RoadDstBin'].isnull().any(), final_random['RoadDstBin'].isnull().any(), final_test['RoadDstBin'].isnull().any(), final_random_test['RoadDstBin'].isnull().any()

# great, null values removed from RoadDstBin of all four datasets
```

```
(False, False, False, False)
```

```python
# do we need this / what does this do?
set(final['RoadDstBin']), set(final_random['RoadDstBin']), set(final_test['RoadDstBin']), set(final_random_test['RoadDstBin'])
```

```
({0.0, 1.0, 2.0}, {0.0, 1.0, 2.0}, {0.0, 1.0, 2.0}, {0.0, 1.0, 2.0})
```

```python
len(set(final['RoadDstBin'])), len(set(final_random['RoadDstBin'])), len(set(final_test['RoadDstBin'])), len(set(final_random_test['RoadDstBin']))
```

```
(3, 3, 3, 3)
```

```python
"""
adding target labels
"""

if 'target' not in final.columns + final_random.columns:
    final['target'] = 1
    final_random['target'] = 0

if 'target' not in final_test.columns + final_random_test.columns:
    final_test['target'] = 1
    final_random_test['target'] = 0
```

```python
final
```

	TrashOutID	Lat	Lon	Continent	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1stMostFr
0	57837	-26.196481	28.050331	Africa	12718.73	0.840314	332	24	706.25	Fast Food Restaurant
1	57831	-28.466159	28.855491	Africa	1704.99	0.714321	1500	0	3000.00	None
2	57830	-28.514575	28.820170	Africa	3842.15	0.494658	496	4	620.00	Auto Workshop
3	57829	-29.947538	30.915913	Africa	11928.59	0.327302	61	1	61.00	Shopping M
4	57817	-26.376726	27.300703	Africa	2.17	0.493056	1500	0	3000.00	None
...	...	...	...	...	...	...	...	...	...	...
44876	3271	-35.500999	-71.702774	South America	117.51	0.000000	1500	0	3000.00	None
44877	2315	4.535037	-74.116149	South America	6356.60	0.000000	332	5	536.40	Park
44878	2280	4.534641	-74.115919	South America	6356.60	0.000000	319	5	524.00	Park
44879	1840	-25.524732	-49.107813	South America	1088.07	0.772548	581	4	695.00	Soccer Fiel
44880	1839	-25.524935	-49.107255	South America	1088.07	0.772548	548	4	729.50	Soccer Fiel

44841 rows × 21 columns

```python
"""
making the continent into one-hot encoding for both datasets
"""

train_dumpsite_dummies = pd.get_dummies(final['Continent'])
final = pd.concat([final.drop('Continent', axis=1), train_dumpsite_dummies], axis=1)

train_nondumpsite_dummies = pd.get_dummies(final_random['Continent'])
final_random = pd.concat([final_random.drop('Continent', axis=1), train_nondumpsite_dummies], axis=1)

test_dumpsite_dummies = pd.get_dummies(final_test['Continent'])
final_test = pd.concat([final_test.drop('Continent', axis=1), test_dumpsite_dummies], axis=1)

test_nondumpsite_dummies = pd.get_dummies(final_random_test['Continent'])
final_random_test = pd.concat([final_random_test.drop('Continent', axis=1), test_nondumpsite_dummies], axis=1)
```

Using LeaveOneOut encoding on the venue categories.

```python
# combined data of dumpsites and non-dumpsites in train and test sets

data = pd.concat([final, final_random], axis=0)

data_test = pd.concat([final_test, final_random_test], axis=0)
```

```python
"""
Applying an encoding onto the 10 different columns (#thMostFreq and #thClosest)
 categorical variables
"""

columns = ['1stMostFreq', '2ndMostFreq', '3rdMostFreq', '4thMostFreq', '5thMostFreq', '1stClosest', '2ndClosest', '3rdClosest', '4thClosest', '5thClosest']

for col in columns:
  # training
  leaveoneoutenc = ce.LeaveOneOutEncoder(return_df=True)
  leaveoneoutenc.fit(X=data[col], y=data['target'])

  data[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data[col])
  data.drop([col], axis=1, inplace=True)

  #testing
  data_test[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data_test[col])
  data_test.drop([col], axis=1, inplace=True)
  data_test.head()

data.drop('TrashOutID', axis=1, inplace=True)
data_test.drop('TrashOutID', axis=1, inplace=True)
```

```python
"""
removing any null values in other variables
"""

data.dropna(inplace=True)
data_test.dropna(inplace=True)

data.shape, data_test.shape
```

```
((88916, 26), (22222, 26))
```

Building a Model (leave one out encoding)

```python
"""
training and testing separation
"""

X = data.drop(['target'], axis=1).to_numpy()
Y = data['target'].to_numpy()

test_x = data_test.drop(['target'], axis=1).to_numpy()
test_y = data_test['target'].to_numpy()

using_kfold = True

if not using_kfold:
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                      stratify=Y,
                                                      random_state=111)
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")

else:
  X_train = X
  Y_train = Y
  print(f"X_train is {X_train.shape}", f"Y_train is {Y_train.shape}")
```

```
('X_train is (71132, 25)', 'X_test is (17784, 25)')
```

```python
"""
normalizing the continuous variables
"""

scaler = StandardScaler()

scaler.fit(X_train[:, :6])

X_train[:, :6] = scaler.transform(X_train[:, :6])

if not using_kfold:
  X_test[:, :6] = scaler.transform(X_test[:, :6])
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")
else:
  X_train.shape
```

```
('X_train is (71132, 25)', 'X_test is (17784, 25)')
```

```python
"""
building the model
"""

x_input = Input(shape=(X_train.shape[-1],), name='continuous_input')

x = Dense(units=32, activation='relu', name=f'dense_1',
          kernel_regularizer=regularizers.l2(1e-4))(x_input)

# x = Dropout(rate=0.2, name=f'dropout_1')(x)

for i, unit in enumerate([32, 32], start=2):
    x = Dense(units=unit, activation='relu', name=f'dense_{i}',
              kernel_regularizer=regularizers.l2(1e-4))(x)

    # x = BatchNormalization(name=f'batchnorm_{i}')(x)
    # x = Dropout(rate=0.2, name=f'dropout_{i}')(x)

out = Dense(units=1, activation='sigmoid', name='output_layer')(x)

model = Model(inputs=x_input, outputs=out, name='Dumpsite_Prediction_Model')

# saving the model graph and seeing the architecture
plot_model(model, show_shapes=True, to_file='/content/drive/My Drive/Official folder for Trashout/Task_1/models/simone/final_model/leaveoneoutencoding/nn_graph.png')

model.summary()
```

```
Model: "Dumpsite_Prediction_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
continuous_input (InputLayer [(None, 25)]              0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                832
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
output_layer (Dense)         (None, 1)                 33
=================================================================
Total params: 2,977
Trainable params: 2,977
Non-trainable params: 0
_________________________________________________________________
```

Using K-Fold for better learning

```python
"""
using the sklearn library to split the training data
into folds of training and validation splits
"""

n_split=10

fold = StratifiedShuffleSplit(n_splits=n_split, test_size=0.15, random_state=111)

train_acc=[]
train_loss=[]
val_acc=[]
val_loss=[]

fold.get_n_splits(X_train, Y_train)
```

```
10
```

Training for Leaveoneout encoding

```python
"""
compiling and fitting
"""

filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/simone/final_model/leaveoneoutencoding/curr_model.h5'

checkpoint = ModelCheckpoint(filepath=filepath_curr, monitor='val_accuracy', save_best_only=True)

opts = Adam(learning_rate=0.001, epsilon=1e-8, decay=0.0001)

# model will begin training with previously trained weights
try:
    model.load_weights(filepath_curr)
    print('Same architecture as before.\n\n')
except:
    print('Model architecture has been changed. No weights loaded.\n\n')

i = 1 # counter for print statement

if not using_kfold:
  model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])
  history = model.fit(
      X_train,
      Y_train,
      batch_size=128,
      epochs=75,
      validation_data=(X_test, Y_test),
      callbacks=[checkpoint],
      verbose=2
  )
else:
  for train_index, val_index in fold.split(X_train, Y_train):

      train_x, val_x = X_train[train_index], X_train[val_index]
      train_y, val_y = Y_train[train_index], Y_train[val_index]

      model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'])

      history = model.fit(
          train_x,
          train_y,
          batch_size=128,
          epochs=75,
          validation_data=(val_x, val_y),
          callbacks=[checkpoint],
          verbose=0
      )

      curr_train_acc = round(model.evaluate(train_x, train_y, verbose=0)[1]*100, 2)
      curr_val_acc = round(model.evaluate(val_x, val_y, verbose=0)[1]*100, 2)

      curr_train_loss = model.evaluate(train_x, train_y, verbose=0)[0]
      curr_val_loss = model.evaluate(val_x, val_y, verbose=0)[0]

      print("-"*75)
      print(f"Fold {i}: train acc = {curr_train_acc}%")
      print(f"Fold {i}: val acc = {curr_val_acc}%")
      print("-"*75 + "\n\n")

      train_acc.append(curr_train_acc)
      val_acc.append(curr_val_acc)
      train_loss.append(curr_train_loss)
      val_loss.append(curr_val_loss)

      i += 1
```

```
Same architecture as before.



---------------------------------------------------------------------------
Fold 1: train acc = 77.71%
Fold 1: val acc = 76.48%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 2: train acc = 77.95%
Fold 2: val acc = 76.69%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 3: train acc = 77.94%
Fold 3: val acc = 77.21%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 4: train acc = 78.24%
Fold 4: val acc = 76.63%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 5: train acc = 78.26%
Fold 5: val acc = 76.91%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 6: train acc = 78.25%
Fold 6: val acc = 77.45%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 7: train acc = 78.39%
Fold 7: val acc = 77.31%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 8: train acc = 78.29%
Fold 8: val acc = 77.07%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 9: train acc = 78.33%
Fold 9: val acc = 77.68%
---------------------------------------------------------------------------



---------------------------------------------------------------------------
Fold 10: train acc = 78.37%
Fold 10: val acc = 77.48%
---------------------------------------------------------------------------
```

Evaluating average accuracy and loss after K-Fold

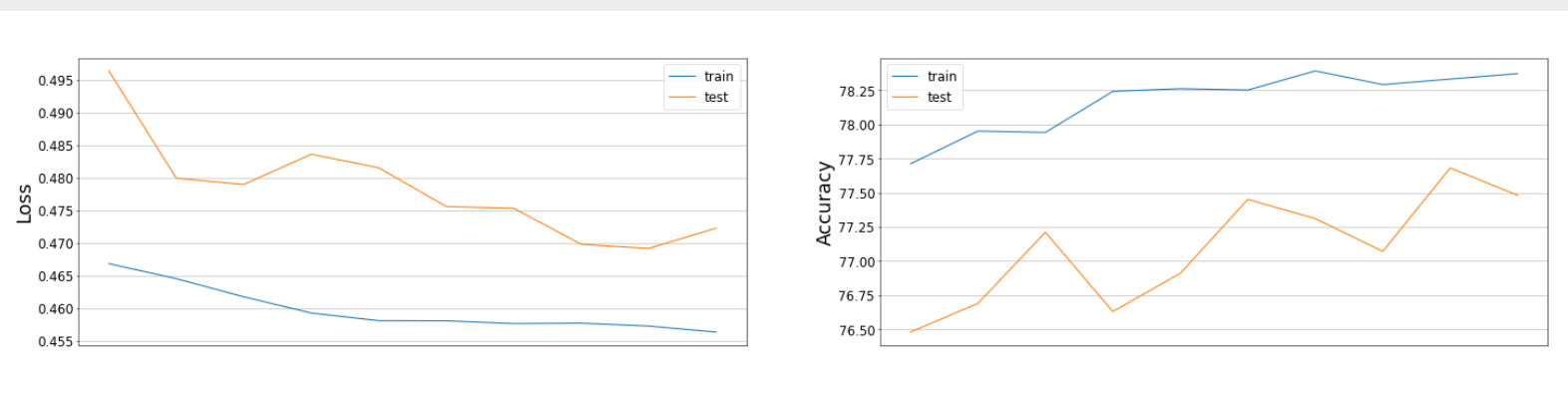
```python
if using_kfold:
  for i, v in enumerate(train_acc, start=0):
    print('-'*50)
    print(f'Fold {i+1} - Train Loss: {train_loss[i]} - Train Accuracy: {train_acc[i]}%')
    print(f'Fold {i+1} - Validation Loss: {val_loss[i]} - Validation Accuracy: {val_acc[i]}%')

  print('-'*50)

  print('\n\nAverage scores for all folds:\n\n')
  # training
  print(f'Train Accuracy: {round(np.mean(train_acc), 2)} +- {round(np.std(train_acc), 2)}')
  print(f'Train Loss: {round(np.mean(train_loss), 3)}')
  # validation
  print(f'\n\nTest Accuracy: {round(np.mean(val_acc), 2)} +- {round(np.std(val_acc), 2)}')
  print(f'Test Loss: {round(np.mean(val_loss), 3)}')
```

```
--------------------------------------------------
Fold 1 - Train Loss: 0.46685877442359924 - Train Accuracy: 77.71%
Fold 1 - Validation Loss: 0.49640798568725586 - Validation Accuracy: 76.48%
--------------------------------------------------
Fold 2 - Train Loss: 0.4645579159259796 - Train Accuracy: 77.95%
Fold 2 - Validation Loss: 0.47994858026504517 - Validation Accuracy: 76.69%
--------------------------------------------------
Fold 3 - Train Loss: 0.46177178621292114 - Train Accuracy: 77.94%
Fold 3 - Validation Loss: 0.47899767756462097 - Validation Accuracy: 77.21%
--------------------------------------------------
Fold 4 - Train Loss: 0.4592919945716858 - Train Accuracy: 78.24%
Fold 4 - Validation Loss: 0.48363110423088074 - Validation Accuracy: 76.63%
--------------------------------------------------
Fold 5 - Train Loss: 0.45811691880226135 - Train Accuracy: 78.26%
Fold 5 - Validation Loss: 0.4815528988838196 - Validation Accuracy: 76.91%
--------------------------------------------------
Fold 6 - Train Loss: 0.4580909311771393 - Train Accuracy: 78.25%
Fold 6 - Validation Loss: 0.475614070892334 - Validation Accuracy: 77.45%
--------------------------------------------------
Fold 7 - Train Loss: 0.45766544342041016 - Train Accuracy: 78.39%
Fold 7 - Validation Loss: 0.4753448963165283 - Validation Accuracy: 77.31%
--------------------------------------------------
Fold 8 - Train Loss: 0.457732617855072 - Train Accuracy: 78.29%
Fold 8 - Validation Loss: 0.4698292911052704 - Validation Accuracy: 77.07%
--------------------------------------------------
Fold 9 - Train Loss: 0.45728957653045654 - Train Accuracy: 78.33%
Fold 9 - Validation Loss: 0.4691846966743469 - Validation Accuracy: 77.68%
--------------------------------------------------
Fold 10 - Train Loss: 0.4563648998737335 - Train Accuracy: 78.37%
Fold 10 - Validation Loss: 0.47230854630470276 - Validation Accuracy: 77.48%
--------------------------------------------------


Average scores for all folds:


Train Accuracy: 78.17 +- 0.21
Train Loss: 0.46


Test Accuracy: 77.09 +- 0.38
Test Loss: 0.478
```

```python
"""
this code takes care of saving the new model only if its accuracy is better than that of the last model
"""

filepath = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/simone/final_model/leaveoneoutencoding/best_model.h5'
filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/simone/final_model/leaveoneoutencoding/curr_model.h5'


if os.path.exists(filepath):
  prev_model = load_model(filepath)
  curr_model = load_model(filepath_curr)

  prev_acc = prev_model.evaluate(test_x, test_y, verbose=0)[1]
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]

  if curr_acc > prev_acc:
    print("There was a previous model saved.\n")
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    curr_model.save(filepath)
    print('\nNew model is saved.')
  else:
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    print('Old model is kept.')

else: # if this is the first time saving the model
  model.save(filepath)
  curr_model = load_model(filepath)
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]
  print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
  print('\nFirst time model is saved.')
```

```
There was a previous model saved.

Previous test accuracy: 49.01%
Current test accuracy: 49.99%

New model is saved.
```

```python
"""
looking at the train/test loss and accuracy over the epochs
"""

width_in_inches = 35
height_in_inches = 7
dots_per_inch = 50

plt.figure(
    figsize=(width_in_inches, height_in_inches), dpi=dots_per_inch)

# PICK THE FOLD
fold = 5

# plot loss during training
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['loss'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(loc='upper right', fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/train.png')

# plot accuracy during training
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['accuracy'])+1, 20), fontsize=15, color='black')
plt.yticks(fontsize=17, color='black')
plt.legend(fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/final_model/meanencoding/test.png')

plt.show()
```




```python
"""
printing the confusion matrix
"""

preds = load_model(filepath).predict(test_x).squeeze()

assert preds.shape == test_y.shape, 'The shapes of the two tensors are unequal'

print(confusion_matrix(test_y, np.round(preds)))

print(classification_report(test_y, np.round(preds)))
```

```
[[  172 10868]
 [  246 10936]]
              precision    recall  f1-score   support

           0       0.41      0.02      0.03     11040
           1       0.50      0.98      0.66     11182

    accuracy                           0.50     22222
   macro avg       0.46      0.50      0.35     22222
weighted avg       0.46      0.50      0.35     22222
```

Test 4 - Data variation between dumpsites and non-dumpsites

Could be structured so as to change the 50/50 ratio between dumpsites and non-dumpsites dataset. Two possible variabtions to get started with

1. More dumpsite data than non-dumpsite data - **Reasoning could be that our model should be good at detecting dumpsites, hence it needs to learn what they look like**.
2. Less dumpsite than than non-dumpsite data - **More closely related to what the data would probably look like in the real world.**

Both variations need to be tested. However, **first variation is more sensible on a preliminary logical approach (opinion) and should be given priority to tune**.

```python
# training data
final_bal = final.copy()

final_random_bal = final_random.copy()

# testing data
final_test_bal = final_test.copy()

final_random_test_bal = final_random_test.copy()
```

```python
final_bal.shape, final_random_bal.shape, final_test_bal.shape, final_random_test_bal.shape
```

```
((44841, 27), (44505, 27), (11207, 27), (11123, 27))
```

```python
final_bal.head()
```

	TrashOutID	Lat	Lon	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1stMostFreq	2
0	57837	-26.196481	28.050331	25.118444	12718.73	0.840314	332	24	706.25	Fast Food Restaurant	F F
1	57831	-28.466159	28.855491	16.073944	1704.99	0.714321	1500	0	3000.00	None	N
2	57830	-28.514575	28.820170	33.621466	3842.15	0.494658	496	4	620.00	Auto Workshop	E
3	57829	-29.947538	30.915913	32.739744	11928.59	0.327302	61	1	61.00	Shopping Mall	N
4	57817	-26.376726	27.300703	6.813308	2.17	0.493056	1500	0	3000.00	None	N

Reduce the size of the training random dumpsites dataset

```python
final_random_bal = final_random_bal.sample(frac = 1.0)
final_bal = final_bal.sample(frac = 1.0)

print(final_bal.shape, final_random_bal.shape)
final_test_bal.shape, final_random_test_bal.shape
```

```
(44841, 27) (44505, 27)


((11207, 27), (11123, 27))
```

```python
final_bal.head()
```

	TrashOutID	Lat	Lon	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1stMost
20081	30320	41.832613	19.661985	1.673061	94.16	0.000903	1500	0	3000.000000	None
39086	743	48.246214	17.284035	1.990003	235.77	0.292847	625	4	782.250000	Slovak Restaurar
12632	42040	41.600443	19.668775	2.326226	257.21	0.000000	1500	0	3000.000000	None
10349	45693	41.210574	-8.562719	11.930083	9742.34	0.447761	245	29	671.000000	Bakery
35943	5556	48.146982	17.076084	1.412669	3655.20	0.066986	216	28	629.285714	Restaurar

```python
class Myobj:
    def __init__(self):
        self.user = user
        self.add = 'really '
        self.i = 1

    def dostuff(self):
        self.user = input("Do you want to use this program(y/n): ")
        while self.user.lower() != 'y':
            self.user = input(f"Are you {self.add * self.i}sure that you do not want to use this program? ")
            self.i += 1
        print("\nIt's not like this does something lol")

binod = Myobj().dostuff()
```

```python
# combined data of dumpsites and non-dumpsites in train and test sets

data_bal = pd.concat([final_bal, final_random_bal], axis=0)

data_test_bal = pd.concat([final_test_bal, final_random_test_bal], axis=0)
```

```python
data_bal.drop(['3rdMostFreq', '4thMostFreq', '5thMostFreq',
               '3rdClosest', '4thClosest', '5thClosest'], axis=1, inplace=True)
data_test_bal.drop(['3rdMostFreq', '4thMostFreq', '5thMostFreq',
               '3rdClosest', '4thClosest', '5thClosest'], axis=1, inplace=True)
```

```python
data_bal.head()
```

	TrashOutID	Lat	Lon	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	1stMost
20081	30320	41.832613	19.661985	1.673061	94.16	0.000903	1500	0	3000.000000	None
39086	743	48.246214	17.284035	1.990003	235.77	0.292847	625	4	782.250000	Slovak Restaurar
12632	42040	41.600443	19.668775	2.326226	257.21	0.000000	1500	0	3000.000000	None
10349	45693	41.210574	-8.562719	11.930083	9742.34	0.447761	245	29	671.000000	Bakery
35943	5556	48.146982	17.076084	1.412669	3655.20	0.066986	216	28	629.285714	Restaurar

Encoding for venue categories

```python
#@title # Transform the Venue categories by picking an encoding below
"""
Applying an encoding onto the 10 different columns (#thMostFreq and #thClosest)
 categorical variables
"""

encoding = "LeaveOneOut" #@param ["LeaveOneOut", "Frequency", "Mean"]

columns = ['1stMostFreq', '2ndMostFreq',
           '1stClosest', '2ndClosest']

if encoding.lower() == 'leaveoneout':
    for col in columns:
        # training
        leaveoneoutenc = ce.LeaveOneOutEncoder(return_df=True)
        leaveoneoutenc.fit(X=data_bal[col], y=data_bal['target'])

        data_bal[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data_bal[col])
        data_bal.drop([col], axis=1, inplace=True)

        #testing
        data_test_bal[col + 'leaveoneout_enc'] = leaveoneoutenc.transform(data_test_bal[col])
        data_test_bal.drop([col], axis=1, inplace=True)
        data_test_bal.head()

    data_bal.drop(['TrashOutID', 'Lat', 'Lon'], axis=1, inplace=True)
    data_test_bal.drop(['TrashOutID', 'Lat', 'Lon'], axis=1, inplace=True)

elif encoding.lower() == 'frequency':
    for col in columns:
        freq = data_bal.groupby(col).size() / len(data_bal)
        # training
        data_bal[col + 'frequency_enc'] = data_bal[col].map(freq)
        data_bal.drop([col], axis=1, inplace=True)

        #testing
        data_test_bal[col + 'frequency_enc'] = data_test_bal[col].map(freq)
        data_test_bal.drop([col], axis=1, inplace=True)

    data_bal.drop(['TrashOutID', 'Lat', 'Lon'], axis=1, inplace=True)
    data_test_bal.drop(['TrashOutID', 'Lat', 'Lon'], axis=1, inplace=True)

elif encoding.lower() == 'mean':
        class MultiColumnEncoder:
            def __init__(self, columns=None):
                self.columns = columns # array of column names to encode

            def fit_transform(self, X, test_set):
                mean = X['target'].mean()
                if self.columns is not None:
                    for col in self.columns:
                        agg= X.groupby(col)['target'].agg(['count','mean'])
                        counts=agg['count']
                        means=agg['mean']
                        weight=100
                        smooth = (counts*means +weight *mean) / (counts + weight)

                        X.loc[:, col+'_mean'] = X[col].map(smooth)
                        test_set.loc[:, col+'_mean'] = test_set[col].map(smooth)
                X.drop(self.columns,axis=1,inplace=True)
                test_set.drop(self.columns,axis=1,inplace=True)
                return X
        mc_class = MultiColumnEncoder(columns)
        data_bal = mc_class.fit_transform(data_bal, data_test_bal)
        data_bal.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)
        data_test_bal.drop(['Lat', 'Lon', 'TrashOutID'], axis=1, inplace=True)
```

```python
data_bal.head()
```

	Distance to Road	Population Density	Population gradient	Distance to Nearest Venue	Number of Venues	Avg Dist to Venues	Africa	Asia	Australia	Europe	North America
20081	1.673061	94.16	0.000903	1500	0	3000.000000	0	0	0	1	0
39086	1.990003	235.77	0.292847	625	4	782.250000	0	0	0	1	0
12632	2.326226	257.21	0.000000	1500	0	3000.000000	0	0	0	1	0
10349	11.930083	9742.34	0.447761	245	29	671.000000	0	0	0	1	0
35943	1.412669	3655.20	0.066986	216	28	629.285714	0	0	0	1	0

```python
#@markdown ## Removing NaN values
"""
removing any null values in other variables
"""

print(data_bal.shape, data_test_bal.shape)

data_bal.dropna(inplace=True)
data_test_bal.dropna(inplace=True)

data_bal.shape, data_test_bal.shape
```

```
(89346, 18) (22330, 18)


((88916, 18), (22222, 18))
```

Build the model

```python
"""
training and testing separation
"""

X = data_bal.drop(['target'], axis=1).to_numpy()
Y = data_bal['target'].to_numpy()

test_x = data_test_bal.drop(['target'], axis=1).to_numpy()
test_y = data_test_bal['target'].to_numpy()

using_kfold = False

if not using_kfold:
  X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,
                                                      stratify=Y,
                                                      random_state=111)
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")

else:
  X_train = X
  Y_train = Y
  print(f"X_train is {X_train.shape}", f"Y_train is {Y_train.shape}")
```

```
X_train is (71132, 17) X_test is (17784, 17)
```

```python
"""
normalizing the continuous variables
"""

scaler = StandardScaler()

scaler.fit(X_train[:, :6])

X_train[:, :6] = scaler.transform(X_train[:, :6])

if not using_kfold:
  X_test[:, :6] = scaler.transform(X_test[:, :6])
  print(f"X_train is {X_train.shape}", f"X_test is {X_test.shape}")
else:
  X_train.shape
```

```
X_train is (71132, 17) X_test is (17784, 17)
```

```python
"""
building the model
"""

x_input = Input(shape=(X_train.shape[-1],), name='continuous_input')

x = Dense(units=16, activation='relu', name=f'dense_1',
          kernel_regularizer=regularizers.l2(1e-3),
          activity_regularizer=regularizers.l2(1e-3))(x_input)

# x = BatchNormalization(name=f'batchnorm_1')(x)
x = Dropout(rate=0.5, name=f'dropout_1')(x)

for i, unit in enumerate([16], start=2):
    x = Dense(units=unit, activation='relu', name=f'dense_{i}',
              kernel_regularizer=regularizers.l2(1e-3),
              activity_regularizer=regularizers.l2(1e-3))(x)

    # x = BatchNormalization(name=f'batchnorm_{i}')(x)
    x = Dropout(rate=0.5, name=f'dropout_{i}')(x)

out = Dense(units=1, activation='sigmoid', name='output_layer')(x)

model = Model(inputs=x_input, outputs=out, name='Dumpsite_Prediction_Model')

# saving the model graph and seeing the architecture
plot_model(model, show_shapes=True, to_file='/content/drive/My Drive/Official fo
lder for Trashout/Task_1/models/pedro/final_model/nn_graph.png')

model.summary()
```

```
Model: "Dumpsite_Prediction_Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
continuous_input (InputLayer [(None, 17)]              0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                288
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
output_layer (Dense)         (None, 1)                 17
=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
_________________________________________________________________
```

Using K-Fold for better learning

```python
"""
using the sklearn library to split the training data
into folds of training and validation splits
"""

n_split=10

fold = StratifiedShuffleSplit(n_splits=n_split, test_size=0.15,
                              random_state=111)

train_acc=[]
train_loss=[]
val_acc=[]
val_loss=[]

fold.get_n_splits(X_train, Y_train)
```

```
10
```

Training

```python
"""
compiling and fitting
"""

filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/mod
els/pedro/final_model/curr_model.h5'

checkpoint = ModelCheckpoint(filepath=filepath_curr, monitor='val_accuracy', sav
e_best_only=True)

opts = Adam(learning_rate=0.001, epsilon=1e-8, decay=0.000005)

# model will begin training with previously trained weights
try:
    model.load_weights(filepath_curr)
    print('Same architecture as before.\n\n')
except:
    print('Model architecture has been changed. No weights loaded.\n\n')

i = 1 # counter for print statement

if not using_kfold:
  model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accuracy'
])
  history = model.fit(
      X_train,
      Y_train,
      batch_size=128,
      epochs=100,
      validation_data=(X_test, Y_test),
      callbacks=[checkpoint],
      verbose=2
  )
else:
  for train_index, val_index in fold.split(X_train, Y_train):

      train_x, val_x = X_train[train_index], X_train[val_index]
      train_y, val_y = Y_train[train_index], Y_train[val_index]

      model.compile(optimizer=opts, loss='binary_crossentropy', metrics=['accura
cy'])

      history = model.fit(
          train_x,
          train_y,
          batch_size=128,
          epochs=75,
          validation_data=(val_x, val_y),
          callbacks=[checkpoint],
          verbose=0
      )

      curr_train_acc = round(model.evaluate(train_x, train_y, verbose=0)[1]*100,
 2)
      curr_val_acc = round(model.evaluate(val_x, val_y, verbose=0)[1]*100, 2)

      curr_train_loss = model.evaluate(train_x, train_y, verbose=0)[0]
      curr_val_loss = model.evaluate(val_x, val_y, verbose=0)[0]

      print("-"*75)
      print(f"Fold {i}: train acc = {curr_train_acc}%")
      print(f"Fold {i}: val acc = {curr_val_acc}%")
      print("-"*75 + "\n\n")

      train_acc.append(curr_train_acc)
      val_acc.append(curr_val_acc)
      train_loss.append(curr_train_loss)
      val_loss.append(curr_val_loss)

      i += 1
```

```
Model architecture has been changed. No weights loaded.


Epoch 1/100
556/556 - 1s - loss: 0.7037 - accuracy: 0.5820 - val_loss: 0.6508 - val_accurac
y: 0.6606
Epoch 2/100
556/556 - 1s - loss: 0.6541 - accuracy: 0.6488 - val_loss: 0.6292 - val_accurac
y: 0.6709
Epoch 3/100
556/556 - 1s - loss: 0.6382 - accuracy: 0.6655 - val_loss: 0.6100 - val_accurac
y: 0.6771
Epoch 4/100
556/556 - 1s - loss: 0.6244 - accuracy: 0.6757 - val_loss: 0.5975 - val_accurac
y: 0.6894
Epoch 5/100
556/556 - 1s - loss: 0.6173 - accuracy: 0.6810 - val_loss: 0.5905 - val_accurac
y: 0.7005
Epoch 6/100
556/556 - 1s - loss: 0.6094 - accuracy: 0.6889 - val_loss: 0.5849 - val_accurac
y: 0.7062
Epoch 7/100
556/556 - 1s - loss: 0.6062 - accuracy: 0.6917 - val_loss: 0.5796 - val_accurac
y: 0.7104
Epoch 8/100
556/556 - 1s - loss: 0.6021 - accuracy: 0.6980 - val_loss: 0.5762 - val_accurac
y: 0.7174
Epoch 9/100
556/556 - 1s - loss: 0.6003 - accuracy: 0.6982 - val_loss: 0.5731 - val_accurac
y: 0.7151
Epoch 10/100
556/556 - 1s - loss: 0.5984 - accuracy: 0.6997 - val_loss: 0.5725 - val_accurac
y: 0.7142
Epoch 11/100
556/556 - 1s - loss: 0.5960 - accuracy: 0.7015 - val_loss: 0.5700 - val_accurac
y: 0.7171
Epoch 12/100
556/556 - 1s - loss: 0.5947 - accuracy: 0.7019 - val_loss: 0.5697 - val_accurac
y: 0.7157
Epoch 13/100
556/556 - 1s - loss: 0.5940 - accuracy: 0.7021 - val_loss: 0.5685 - val_accurac
y: 0.7169
Epoch 14/100
556/556 - 1s - loss: 0.5929 - accuracy: 0.7036 - val_loss: 0.5676 - val_accurac
y: 0.7150
Epoch 15/100
556/556 - 1s - loss: 0.5940 - accuracy: 0.7028 - val_loss: 0.5691 - val_accurac
y: 0.7185
Epoch 16/100
556/556 - 1s - loss: 0.5915 - accuracy: 0.7053 - val_loss: 0.5663 - val_accurac
y: 0.7187
Epoch 17/100
556/556 - 1s - loss: 0.5926 - accuracy: 0.7044 - val_loss: 0.5669 - val_accurac
y: 0.7192
Epoch 18/100
556/556 - 1s - loss: 0.5914 - accuracy: 0.7039 - val_loss: 0.5667 - val_accurac
y: 0.7166
Epoch 19/100
556/556 - 1s - loss: 0.5912 - accuracy: 0.7056 - val_loss: 0.5648 - val_accurac
y: 0.7207
Epoch 20/100
556/556 - 1s - loss: 0.5895 - accuracy: 0.7059 - val_loss: 0.5649 - val_accurac
y: 0.7200
Epoch 21/100
556/556 - 1s - loss: 0.5893 - accuracy: 0.7068 - val_loss: 0.5647 - val_accurac
y: 0.7192
Epoch 22/100
556/556 - 1s - loss: 0.5906 - accuracy: 0.7045 - val_loss: 0.5634 - val_accurac
y: 0.7220
Epoch 23/100
556/556 - 1s - loss: 0.5905 - accuracy: 0.7042 - val_loss: 0.5633 - val_accurac
y: 0.7225
Epoch 24/100
556/556 - 1s - loss: 0.5912 - accuracy: 0.7041 - val_loss: 0.5631 - val_accurac
y: 0.7213
Epoch 25/100
556/556 - 1s - loss: 0.5901 - accuracy: 0.7053 - val_loss: 0.5635 - val_accurac
y: 0.7205
Epoch 26/100
556/556 - 1s - loss: 0.5881 - accuracy: 0.7057 - val_loss: 0.5650 - val_accurac
y: 0.7185
Epoch 27/100
556/556 - 1s - loss: 0.5892 - accuracy: 0.7055 - val_loss: 0.5649 - val_accurac
y: 0.7227
Epoch 28/100
556/556 - 1s - loss: 0.5894 - accuracy: 0.7075 - val_loss: 0.5655 - val_accurac
y: 0.7156
Epoch 29/100
556/556 - 1s - loss: 0.5885 - accuracy: 0.7072 - val_loss: 0.5610 - val_accurac
y: 0.7211
Epoch 30/100
556/556 - 1s - loss: 0.5876 - accuracy: 0.7063 - val_loss: 0.5644 - val_accurac
y: 0.7226
Epoch 31/100
556/556 - 1s - loss: 0.5867 - accuracy: 0.7063 - val_loss: 0.5611 - val_accurac
y: 0.7206
Epoch 32/100
556/556 - 1s - loss: 0.5879 - accuracy: 0.7047 - val_loss: 0.5628 - val_accurac
y: 0.7222
Epoch 33/100
556/556 - 1s - loss: 0.5871 - accuracy: 0.7061 - val_loss: 0.5617 - val_accurac
y: 0.7207
Epoch 34/100
556/556 - 1s - loss: 0.5885 - accuracy: 0.7066 - val_loss: 0.5612 - val_accurac
y: 0.7233
Epoch 35/100
556/556 - 1s - loss: 0.5865 - accuracy: 0.7071 - val_loss: 0.5600 - val_accurac
y: 0.7241
Epoch 36/100
556/556 - 1s - loss: 0.5872 - accuracy: 0.7069 - val_loss: 0.5616 - val_accurac
y: 0.7190
Epoch 37/100
556/556 - 1s - loss: 0.5875 - accuracy: 0.7063 - val_loss: 0.5606 - val_accurac
y: 0.7206
Epoch 38/100
556/556 - 1s - loss: 0.5880 - accuracy: 0.7044 - val_loss: 0.5614 - val_accurac
y: 0.7195
Epoch 39/100
556/556 - 1s - loss: 0.5875 - accuracy: 0.7073 - val_loss: 0.5635 - val_accurac
y: 0.7190
Epoch 40/100
556/556 - 1s - loss: 0.5879 - accuracy: 0.7056 - val_loss: 0.5614 - val_accurac
y: 0.7158
Epoch 41/100
556/556 - 1s - loss: 0.5862 - accuracy: 0.7068 - val_loss: 0.5603 - val_accurac
y: 0.7187
Epoch 42/100
556/556 - 1s - loss: 0.5863 - accuracy: 0.7073 - val_loss: 0.5604 - val_accurac
y: 0.7203
Epoch 43/100
556/556 - 1s - loss: 0.5859 - accuracy: 0.7076 - val_loss: 0.5632 - val_accurac
y: 0.7199
Epoch 44/100
556/556 - 1s - loss: 0.5868 - accuracy: 0.7055 - val_loss: 0.5609 - val_accurac
y: 0.7192
Epoch 45/100
556/556 - 1s - loss: 0.5852 - accuracy: 0.7078 - val_loss: 0.5618 - val_accurac
y: 0.7182
Epoch 46/100
556/556 - 1s - loss: 0.5880 - accuracy: 0.7063 - val_loss: 0.5608 - val_accurac
y: 0.7183
Epoch 47/100
556/556 - 1s - loss: 0.5862 - accuracy: 0.7057 - val_loss: 0.5623 - val_accurac
y: 0.7224
Epoch 48/100
556/556 - 1s - loss: 0.5855 - accuracy: 0.7074 - val_loss: 0.5634 - val_accurac
y: 0.7181
Epoch 49/100
556/556 - 1s - loss: 0.5861 - accuracy: 0.7074 - val_loss: 0.5605 - val_accurac
y: 0.7191
Epoch 50/100
556/556 - 1s - loss: 0.5854 - accuracy: 0.7085 - val_loss: 0.5590 - val_accurac
y: 0.7211
Epoch 51/100
556/556 - 1s - loss: 0.5850 - accuracy: 0.7080 - val_loss: 0.5598 - val_accurac
y: 0.7219
Epoch 52/100
556/556 - 1s - loss: 0.5862 - accuracy: 0.7078 - val_loss: 0.5595 - val_accurac
y: 0.7191
Epoch 53/100
556/556 - 1s - loss: 0.5864 - accuracy: 0.7066 - val_loss: 0.5569 - val_accurac
y: 0.7234
Epoch 54/100
556/556 - 1s - loss: 0.5857 - accuracy: 0.7069 - val_loss: 0.5584 - val_accurac
y: 0.7245
Epoch 55/100
556/556 - 1s - loss: 0.5860 - accuracy: 0.7053 - val_loss: 0.5594 - val_accurac
y: 0.7190
Epoch 56/100
556/556 - 1s - loss: 0.5859 - accuracy: 0.7073 - val_loss: 0.5599 - val_accurac
y: 0.7182
Epoch 57/100
556/556 - 1s - loss: 0.5853 - accuracy: 0.7073 - val_loss: 0.5621 - val_accurac
y: 0.7187
Epoch 58/100
556/556 - 1s - loss: 0.5857 - accuracy: 0.7065 - val_loss: 0.5595 - val_accurac
y: 0.7184
Epoch 59/100
556/556 - 1s - loss: 0.5851 - accuracy: 0.7083 - val_loss: 0.5598 - val_accurac
y: 0.7215
Epoch 60/100
556/556 - 1s - loss: 0.5865 - accuracy: 0.7072 - val_loss: 0.5603 - val_accurac
y: 0.7208
Epoch 61/100
556/556 - 1s - loss: 0.5844 - accuracy: 0.7074 - val_loss: 0.5589 - val_accurac
y: 0.7199
Epoch 62/100
556/556 - 1s - loss: 0.5849 - accuracy: 0.7065 - val_loss: 0.5587 - val_accurac
y: 0.7203
Epoch 63/100
556/556 - 1s - loss: 0.5853 - accuracy: 0.7087 - val_loss: 0.5594 - val_accurac
y: 0.7187
Epoch 64/100
556/556 - 1s - loss: 0.5848 - accuracy: 0.7076 - val_loss: 0.5604 - val_accurac
y: 0.7167
Epoch 65/100
556/556 - 1s - loss: 0.5863 - accuracy: 0.7068 - val_loss: 0.5585 - val_accurac
y: 0.7223
Epoch 66/100
556/556 - 1s - loss: 0.5863 - accuracy: 0.7069 - val_loss: 0.5593 - val_accurac
y: 0.7198
Epoch 67/100
556/556 - 1s - loss: 0.5854 - accuracy: 0.7078 - val_loss: 0.5588 - val_accurac
y: 0.7200
Epoch 68/100
556/556 - 1s - loss: 0.5848 - accuracy: 0.7077 - val_loss: 0.5599 - val_accurac
y: 0.7209
Epoch 69/100
556/556 - 1s - loss: 0.5842 - accuracy: 0.7086 - val_loss: 0.5607 - val_accurac
y: 0.7193
Epoch 70/100
556/556 - 1s - loss: 0.5840 - accuracy: 0.7078 - val_loss: 0.5563 - val_accurac
y: 0.7184
Epoch 71/100
556/556 - 1s - loss: 0.5843 - accuracy: 0.7083 - val_loss: 0.5614 - val_accurac
y: 0.7179
Epoch 72/100
556/556 - 1s - loss: 0.5857 - accuracy: 0.7068 - val_loss: 0.5598 - val_accurac
y: 0.7191
Epoch 73/100
556/556 - 1s - loss: 0.5835 - accuracy: 0.7071 - val_loss: 0.5566 - val_accurac
y: 0.7210
Epoch 74/100
556/556 - 1s - loss: 0.5853 - accuracy: 0.7062 - val_loss: 0.5579 - val_accurac
y: 0.7200
Epoch 75/100
556/556 - 1s - loss: 0.5850 - accuracy: 0.7069 - val_loss: 0.5586 - val_accurac
y: 0.7224
Epoch 76/100
556/556 - 1s - loss: 0.5839 - accuracy: 0.7090 - val_loss: 0.5602 - val_accurac
y: 0.7199
Epoch 77/100
556/556 - 1s - loss: 0.5851 - accuracy: 0.7069 - val_loss: 0.5617 - val_accurac
y: 0.7210
Epoch 78/100
556/556 - 1s - loss: 0.5868 - accuracy: 0.7055 - val_loss: 0.5660 - val_accurac
y: 0.7141
Epoch 79/100
556/556 - 1s - loss: 0.5850 - accuracy: 0.7073 - val_loss: 0.5584 - val_accurac
y: 0.7207
Epoch 80/100
556/556 - 1s - loss: 0.5856 - accuracy: 0.7069 - val_loss: 0.5590 - val_accurac
y: 0.7197
Epoch 81/100
556/556 - 1s - loss: 0.5842 - accuracy: 0.7082 - val_loss: 0.5613 - val_accurac
y: 0.7195
Epoch 82/100
556/556 - 1s - loss: 0.5866 - accuracy: 0.7073 - val_loss: 0.5605 - val_accurac
y: 0.7200
Epoch 83/100
556/556 - 1s - loss: 0.5831 - accuracy: 0.7072 - val_loss: 0.5568 - val_accurac
y: 0.7226
Epoch 84/100
556/556 - 1s - loss: 0.5852 - accuracy: 0.7078 - val_loss: 0.5585 - val_accurac
y: 0.7183
Epoch 85/100
556/556 - 1s - loss: 0.5832 - accuracy: 0.7089 - val_loss: 0.5576 - val_accurac
y: 0.7226
Epoch 86/100
556/556 - 1s - loss: 0.5840 - accuracy: 0.7072 - val_loss: 0.5593 - val_accurac
y: 0.7179
Epoch 87/100
556/556 - 1s - loss: 0.5839 - accuracy: 0.7097 - val_loss: 0.5586 - val_accurac
y: 0.7167
Epoch 88/100
556/556 - 1s - loss: 0.5845 - accuracy: 0.7082 - val_loss: 0.5600 - val_accurac
y: 0.7191
Epoch 89/100
556/556 - 1s - loss: 0.5847 - accuracy: 0.7071 - val_loss: 0.5582 - val_accurac
y: 0.7211
Epoch 90/100
556/556 - 1s - loss: 0.5850 - accuracy: 0.7067 - val_loss: 0.5582 - val_accurac
y: 0.7198
Epoch 91/100
556/556 - 1s - loss: 0.5838 - accuracy: 0.7081 - val_loss: 0.5595 - val_accurac
y: 0.7165
Epoch 92/100
556/556 - 1s - loss: 0.5838 - accuracy: 0.7096 - val_loss: 0.5570 - val_accurac
y: 0.7195
Epoch 93/100
556/556 - 1s - loss: 0.5851 - accuracy: 0.7076 - val_loss: 0.5574 - val_accurac
y: 0.7224
Epoch 94/100
556/556 - 1s - loss: 0.5829 - accuracy: 0.7066 - val_loss: 0.5561 - val_accurac
y: 0.7219
Epoch 95/100
556/556 - 1s - loss: 0.5845 - accuracy: 0.7067 - val_loss: 0.5598 - val_accurac
y: 0.7221
Epoch 96/100
556/556 - 1s - loss: 0.5832 - accuracy: 0.7087 - val_loss: 0.5555 - val_accurac
y: 0.7226
Epoch 97/100
556/556 - 1s - loss: 0.5847 - accuracy: 0.7089 - val_loss: 0.5575 - val_accurac
y: 0.7190
Epoch 98/100
556/556 - 1s - loss: 0.5832 - accuracy: 0.7092 - val_loss: 0.5557 - val_accurac
y: 0.7221
Epoch 99/100
556/556 - 1s - loss: 0.5825 - accuracy: 0.7080 - val_loss: 0.5597 - val_accurac
y: 0.7211
Epoch 100/100
556/556 - 1s - loss: 0.5841 - accuracy: 0.7066 - val_loss: 0.5593 - val_accurac
y: 0.7207
```

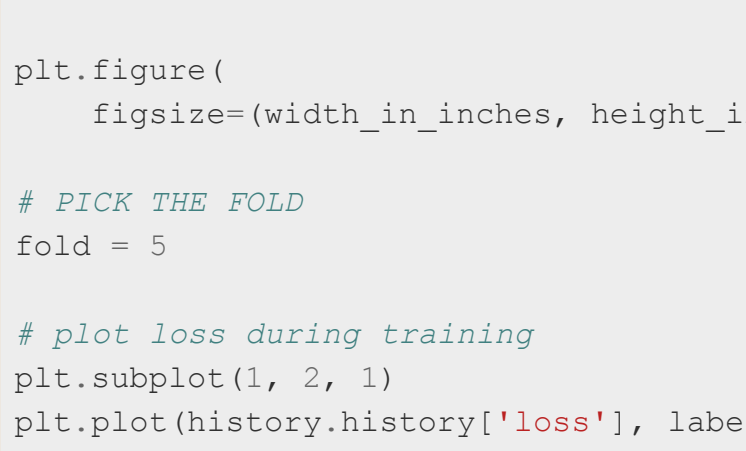
Evaluating average accuracy and loss after K-Fold

```python
if using_kfold:
  for i, v in enumerate(train_acc, start=0):
    print('-'*50)
    print(f'Fold {i+1} - Train Loss: {train_loss[i]} - Train Accuracy: {train_ac
c[i]}%')
    print(f'Fold {i+1} - Validation Loss: {val_loss[i]} - Validation Accuracy:
{val_acc[i]}%')

  print('-'*50)

  print('\n\nAverage scores for all folds:\n\n')
  # training
  print(f'Train Accuracy: {round(np.mean(train_acc), 2)} +- {round(np.std(train_
acc), 2)}')
  print(f'Train Loss: {round(np.mean(train_loss), 3)}')
  # validation
  print(f'\n\nTest Accuracy: {round(np.mean(val_acc), 2)} +- {round(np.std(val_a
cc), 2)}')
  print(f'Test Loss: {round(np.mean(val_loss), 3)}')
```

```python
"""
this code takes care of saving the new model only if its accuracy is better than
that of the last model
"""

filepath = '/content/drive/My Drive/Official folder for Trashout/Task_1/models/p
edro/final_model/best_model.h5'
filepath_curr = '/content/drive/My Drive/Official folder for Trashout/Task_1/mod
els/pedro/final_model/curr_model.h5'


if os.path.exists(filepath):
  prev_model = load_model(filepath)
  curr_model = load_model(filepath_curr)

  prev_acc = prev_model.evaluate(test_x, test_y, verbose=0)[1]
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]

  if curr_acc > prev_acc:
    print("There was a previous model saved.\n")
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    curr_model.save(filepath)
    print('\nNew model is saved.')
  else:
    print(f"Previous test accuracy: {round(prev_acc*100, 2)}%")
    print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
    print('Old model is kept.')

else: # if this is the first time saving the model
  model.save(filepath)
  curr_model = load_model(filepath)
  curr_acc = curr_model.evaluate(test_x, test_y, verbose=0)[1]
  print(f"Current test accuracy: {round(curr_acc*100, 2)}%")
  print('\nFirst time model is saved.')
```

```
Current test accuracy: 54.39%

First time model is saved.
```

```python
"""
looking at the train/test loss and accuracy over the epochs
"""

width_in_inches = 35
height_in_inches = 7
dots_per_inch = 50

plt.figure(
    figsize=(width_in_inches, height_in_inches), dpi=dots_per_inch)

# PICK THE FOLD
fold = 5

# plot loss during training
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['loss'])+1, 20), fontsize=15, color='bl
ack')
plt.yticks(fontsize=17, color='black')
plt.legend(loc='upper right', fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/
pedro/final_model/train.png')

# plot accuracy during training
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy', fontsize=25, color='black')
plt.xticks(range(20, len(history.history['accuracy'])+1, 20), fontsize=15, color
='black')
plt.yticks(fontsize=17, color='black')
plt.legend(fontsize=17)
plt.grid()
plt.savefig('/content/drive/My Drive/Official folder for Trashout/Task_1/models/
pedro/final_model/test.png')

plt.show()
```




```python
"""
printing the confusion matrix
"""

preds = load_model(filepath_curr).predict(test_x).squeeze()
np.expand_dims(test_y, axis=1)

threshold = 0.5
preds[preds >= threshold] = 1
preds[preds < threshold] = 0

assert preds.shape == test_y.shape, 'The shapes of the two tensors are unequal'

print(confusion_matrix(test_y, preds))

print(classification_report(test_y, np.round(preds)))
```

```
[[5712 5328]
 [4675 6507]]
              precision    recall  f1-score   support

           0       0.55      0.52      0.53     11040
           1       0.55      0.58      0.57     11182

    accuracy                           0.55     22222
   macro avg       0.55      0.55      0.55     22222
weighted avg       0.55      0.55      0.55     22222
```

```
probs = load_model(filepath_coco).predict(test_x, argument)
print("Done")
print("X:", len(test_x[test_x==1]), len(test_x[test_x==-1]))
print("Y:")
for i in range(
    len(probs)):
    print(probs[i][0], probs[i][1])
```

```
Do Sandy LEROY 10192
```

```
0.0 2.2E-8
0.1 10732 11327
0.2 10732 11327
0.3000000000000004 10781 11298
0.4 10781 11298
0.5 10781 11298
0.6 10798 11283
0.7 10798 11283
0.7999999999999999 10794 11293
0.8999999999999999 11328 10263
0.9999999999999999 11328 10263
1.1999999999999998 10330 0
```