Divergents_Eval3_Modelling(2)

December 14, 2022

1 Aid Escalating Final Evaluation - 3 (Modelling)

[1]:

```
#Loading Important Libraries

import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import langdetect
import gensim
```

[2]:

```
from gensim.models import Doc2Vec
from sklearn import utils
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from gensim.models.doc2vec import TaggedDocument
import re
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:

```
#Importing necessary libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.neighbors import KNeighborsClassifier


import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns



import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
import re

lst_stopwords = nltk.corpus.stopwords.words("english")
```

[3]:

```
#Reading Training data
df = pd.read_csv("train.csv")
```

[4]:

```
#Reading Test data
df_test = pd.read_csv("test.csv")
```

[5]:

```
#Replacing ? with Nan in Train
df.replace("?", np.nan, inplace=True)
```

[6]:

```
#Replacing ? with Nan in Test
df_test.replace("?", np.nan, inplace=True)
```

[7]:

```
#Total Nan Values
df.isna().sum()
```

[7]:

```
link                                     0
link_id                                  0
page_description                         0
alchemy_category                      1397
alchemy_category_score                1397
avg_link_size                            0
common_word_link_ratio_1                 0
common_word_link_ratio_2                 0
common_word_link_ratio_3                 0
common_word_link_ratio_4                 0
compression_ratio                        0
embed_ratio                              0
frame_based                              0
frame_tag_ratio                          0
has_domain_link                          0
html_ratio                               0
image_ratio                              0
```

```
is_news                                1688
lengthy_link_domain                       0
link_word_score                           0
news_front_page                         727
non_markup_alphanumeric_characters        0
count_of_links                            0
number_of_words_in_url                    0
parametrized_link_ratio                   0
spelling_mistakes_ratio                   0
label                                     0
dtype: int64
```

2 NLP Preprocessing

3 Part -1

[21]:
```
import re
import nltk

#Appending extra words to Stopwords
lst_stopwords = nltk.corpus.stopwords.words("english")
lst_stopwords.append('title')
lst_stopwords.append('url')
```

[22]:
```
# NLP preprocessing to carry out Punctuation Removal, LOwercase Conversion,␣
 ↪Lemmatization, Tokenization

def utils_preprocess_text(text, flg_stemm=False, flg_lemm=True,␣
 ↪lst_stopwords=None):
    ## clean (convert to lowercase and remove punctuations and characters and␣
 ↪then strip)
    text = text.lower()
#     text = re.sub("[\W,\d]"," ",str(text).lower().strip())
    text = re.sub("[\W,\d]"," ",text)

    ## Tokenize (convert from string to list)
    lst_text = text.split()     ## remove Stopwords
    if lst_stopwords is not None:
        lst_text = [word for word in lst_text if word not in
                    lst_stopwords]


#     ## Stemming (remove -ing, -ly, ...)
#     if flg_stemm == True:
#         ps = nltk.stem.porter.PorterStemmer()
#         lst_text = [ps.stem(word) for word in lst_text]
```

```
    ## Lemmatisation (convert the word into root word)
    if flg_lemm == True:
        lem = nltk.stem.wordnet.WordNetLemmatizer()




        lst_text = [lem.lemmatize(word) for word in lst_text]
    while "  " in lst_text:
        lst_text = lst_text.replace("  "," ")

    ## back to string from list
    text = " ".join(lst_text)
    return text
```

[23]:

```
#Applying previous function on Train data
df["text_clean"] = df["page_description"].apply(lambda x:␣
 ↪utils_preprocess_text(x, flg_stemm=False, flg_lemm=True,␣
 ↪lst_stopwords=lst_stopwords))
```

[24]:

```
##Applying previous function on Test data
df_test["text_clean"] = df_test["page_description"].apply(lambda x:␣
 ↪utils_preprocess_text(x, flg_stemm=False, flg_lemm=True,␣
 ↪lst_stopwords=lst_stopwords))
```

[25]:

```
# train_text = df["text_clean"]
# test_text = df_test["text_clean"]
# complete_text = pd.concat([df["text_clean"], df_test["text_clean"]])
```

[26]:

```
# Train Test Split
train, test = train_test_split(df, test_size=0.3, random_state=0)
import nltk
# train = df1

#Retokenizing text to convert into Doc2vec format
from nltk.corpus import stopwords
def tokenize_text(text):
    tokens = []
    for sent in nltk.sent_tokenize(text):
        for word in nltk.word_tokenize(sent):
            if len(word) < 2:
                continue
            tokens.append(word.lower())
    return tokens
train_tagged = train.apply(lambda r:␣
 ↪TaggedDocument(words=tokenize_text(r['text_clean']), tags=[r.label]), axis=1)
```

```
test_tagged = test.apply(lambda r:␣
↪TaggedDocument(words=tokenize_text(r['text_clean']), tags=[r.label]), axis=1)
```

4 Part-2

[ ]:

```
# # Remove all punctuations from the text
# import string as st
# def remove_punct(text):
#     return ("".join([ch for ch in text if ch not in st.punctuation]))
# df["page_description"]  = df["page_description"].apply(lambda x:␣
↪remove_punct(x))
# df_test["page_description"]  = df_test["page_description"].apply(lambda x:␣
↪remove_punct(x))


# #To remove all digits

# import re
# def rem_digits(text):
#     text = re.sub("\d"," ",text)
#     return text

# df["page_description"]  = df["page_description"].apply(lambda x:␣
↪rem_digits(x))
# df_test["page_description"]  = df_test["page_description"].apply(lambda x:␣
↪rem_digits(x))


# #Remove url/title related words
# def rem_url(text):
#     text = re.sub("http|https|url|title|www"," ",text)
#     return text
# df["page_description"]  = df["page_description"] .apply(lambda x: rem_url(x))
# df_test["page_description"]  = df_test["page_description"] .apply(lambda x:␣
↪rem_url(x))


# ''' Convert text to lower case tokens. Here, split() is applied on␣
↪white-spaces. But, it could be applied
#     on special characters, tabs or any other string based on which text is to␣
↪be seperated into tokens.
# '''
# def tokenize(text):
#     text = re.split('\s+' ,text)
#     return [x.lower() for x in text]
# import re
# df["page_description"]  = df["page_description"].apply(lambda msg :␣
↪tokenize(msg))
# df_test["page_description"]  = df_test["page_description"].apply(lambda msg :␣
↪tokenize(msg))


# # Remove tokens of length less than 3
# def remove_small_words(text):
#     return [x for x in text if len(x) > 3 and len(x) < 11]


# df["page_description"] = df["page_description"].apply(lambda x :␣
↪remove_small_words(x))
# df_test["page_description"] = df_test["page_description"].apply(lambda x :␣
↪remove_small_words(x))

# lst_stopwords = nltk.corpus.stopwords.words("english")
# lst_stopwords.extend(["url", "youre",␣
↪"dont","havent","hadnt","wont","wouldnt","cant","cannot","can not","im",
#                        "m","am","ill", "i will","its","it is","s","␣
↪is","thats", "werent", "doesnt", "didnt", "hasnt"
#                       "do␣
↪not","doesnt","doesnot","didnt","didnot","hasnt","hasnot","havent","havenot","hadnt","wont"
#                        "wouldnt","im","iam", "want", "onto", "into", "www",␣
↪"url", "http", "https"])
# def remove_stopwords(text):
#     return [word for word in text if word not in lst_stopwords]


# df["page_description"] = df["page_description"].apply(lambda x :␣
↪remove_stopwords(x))
# df_test["page_description"] = df_test["page_description"].apply(lambda x :␣
↪remove_stopwords(x))

# # Apply lemmatization on tokens
# def lemmatize(text):
#     word_net = WordNetLemmatizer()
#     return [word_net.lemmatize(word) for word in text

# df["page_description"] = df["page_description"].apply(lambda x : lemmatize(x))
# df_test["page_description"] = df_test["page_description"].apply(lambda x :␣
↪lemmatize(x))

# #con
# m = [df["page_description"][i] for i in range(0, len(df["page_description"]))]
# m1 = [df_test["page_description"][i] for i in range(0, len(df_test["page_description"]))]
# x = [" ".join(i) for i in m]
# x1 = [" ".join(j) for j in m1]
# x

# df["page_description"] = [i.split(" ")for i in x]
# df_test["page_description"] = [i.split(" ")for i in x1]


# #Unique word Extraction
# def unique(sequence):
#     seen = set()
#     return [x for x in sequence if not (x in seen or seen.add(x))]


# df["page_description"] = df["page_description"].apply(lambda x : unique(x))
# df_test["page_description"] = df_test["page_description"].apply(lambda x : unique(x))
```

5 Building Vocabulary for Vectorization using Doc2vec

[27]:

```
import multiprocessing
cores = multiprocessing.cpu_count()

#Buiding vocab for Train data(After splitting actual train data)
model_dbow = Doc2Vec(dm=0, vector_size=6000, negative=5, hs=0, min_count=2, sample = 0, workers=cores, alpha=0.025, min_alpha=0.001)
model_dbow.build_vocab([x for x in tqdm(train_tagged)])
```

```
100%|          | 3105/3105 [00:00<00:00, 3885236.85it/s]
```

[28]:

```
import multiprocessing
cores = multiprocessing.cpu_count()

#Building vocab for Test data with labels (After splitting actual train data)
# model_dbow2 = Doc2Vec(dm=0, vector_size=6000, negative=5, hs=0, min_count=2, sample = 0, workers=cores, alpha=0.025, min_alpha=0.001)
model_dbow.build_vocab([x for x in tqdm(test_tagged)])
```

```
100%|          | 1332/1332 [00:00<00:00, 2792010.46it/s]
```

[29]:

```
#Tagging each document with its unique key
```

```
final_tagged = df_test.apply(lambda r:␣
↪TaggedDocument(words=tokenize_text(r['text_clean']),tags = [r.
↪alchemy_category]), axis=1)
```

[30]:
```
len(df_test)
```

[30]:
```
2958
```

[31]:
```
import multiprocessing
cores = multiprocessing.cpu_count()

#Building vocab for Final tagged data
model_dbow1 = Doc2Vec(dm=0, vector_size=6000, negative=5, hs=0, min_count=2,␣
↪sample = 0, workers=cores, alpha=0.025, min_alpha=0.001)
model_dbow1.build_vocab([x for x in tqdm(final_tagged)])
```

```
100%|       | 2958/2958 [00:00<00:00, 3600334.08it/s]
```

6 Embedding & Vectorization using Doc2Vec

[32]:
```
train_documents  = utils.shuffle(train_tagged)

#Function for Vectorization using Doc2vec
model_dbow.train(train_documents,total_examples=len(train_documents), epochs=60)
def vector_for_learning(model, input_docs):
    sents = input_docs
    targets, feature_vectors = zip(*[(doc.tags[0], model.infer_vector(doc.
↪words)) for doc in sents])
    return targets, feature_vectors
```

[33]:
```
test_documents  = utils.shuffle(test_tagged)
```

[34]:
```
# test_documents  = utils.shuffle(test_tagged)
# model_dbow2.train(test_documents,total_examples=len(test_documents),␣
↪epochs=30)
# def vector_for_learning2(model, input_docs):
#     sents = input_docs
#     targets, feature_vectors = zip(*[(doc.tags[0], model.infer_vector(doc.
↪words)) for doc in sents])
#     return targets, feature_vectors
```

[35]:
```
test_documents_final= final_tagged
# model_dbow1.
↪train(test_documents_final,total_examples=len(test_documents_final),␣
↪epochs=30)
# def vector_for_learning1(model, input_docs):
```

```
#     sents = input_docs
#     targets ,feature_vectors = zip(*[(doc.tags[0], model.infer_vector(doc.
 ↪words)) for doc in sents])
#     return targets , feature_vectors
```

[36]:
```
# from sklearn.model_selection import StratifiedKFold
# #kf = StratifiedKFold(n_splits=10, shuffle=False)
# search_space = {"penalty":['l1', 'l2', 'elasticnet', None],
#                  "solver": ['liblinear', 'newton-cg', 'newton-cholesky',␣
 ↪'sag', 'saga'],
#                  "max_iter" : [50,80,100,120],

#                  "multi_class" : ['auto', 'ovr', 'multinomial']}
```

[37]:
```
#Main Train data
y_train, X_train = vector_for_learning(model_dbow, train_documents)
```

[38]:
```
#Main Test data
y_test, X_test = vector_for_learning(model_dbow, test_documents)
```

[40]:
```
#Given test (Unknown label)
y_test1,X_test1 = vector_for_learning(model_dbow,test_documents_final)
```

[41]:
```
# importing utility modules
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss

# importing machine learning models for prediction

from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

# importing voting classifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import RidgeClassifier
```

7 Embedding & Vectorization using TFIDF

[59]:

```
# #For Training data

# word_vectorizer = TfidfVectorizer(
#     sublinear_tf = True,
#     strip_accents = 'unicode',
#     analyzer = 'word',
#     token_pattern = '(?u)\\b\\w\\w+\\b\\w{,1}',
#     lowercase = False,
#     stop_words = 'english',

#     ngram_range = (1, 1),
#     min_df = 5,
#     max_df = 0.95,
#     norm = 'l2',
#     max_features = 8600
# )
# x = word_vectorizer.fit_transform(m).toarray()
# df_ = pd.DataFrame(x)
# df_

# #For Testing data
# word_vectorizer = TfidfVectorizer(
#     sublinear_tf = True,
#     strip_accents = 'unicode',
#     analyzer = 'word',
#     token_pattern = '(?u)\\b\\w\\w+\\b\\w{,1}',
#     lowercase = False,
#     stop_words = 'english',

#     ngram_range = (1, 1),
#     min_df = 5,
#     max_df = 0.95,
#     norm = 'l2',
#     max_features = 8600
# )
# x1 = word_vectorizer.fit_transform(m1).toarray()
# df1_ = pd.DataFrame(x1)
# df1_

# #Concatenating with Label

# # df__ = pd.concat([df_, df["label"]], axis=1)
```

8 Embedding & Vectorization using Word2Vec

[60]:

```
# w2v_model = gensim.models.Word2Vec(X_train,
#                                    vector_size=5000,
#                                    window=5,
#                                    min_count=2)

# words = set(w2v_model.wv.index_to_key )
# X_train_vect = np.array([np.array([w2v_model.wv[i] for i in ls if i in words])
#                          for ls in X_train])
# X_test_vect = np.array([np.array([w2v_model.wv[i] for i in ls if i in words])
#                          for ls in X_test])
```

9 1. Modelling using Voting Classifier

[42]:

```
# initializing all the model objects with default parameters
model_1 = LogisticRegression(max_iter=2000,
 multi_class= 'multinomial',
 penalty= 'l2',
 solver= 'sag')
model_2 = XGBClassifier(booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0.4,eta=0.2,␣
 ↪gpu_id=-1,alpha=7,
              grow_policy='lossguide', importance_type=None,
              interaction_constraints='', learning_rate=0.05, max_bin=256,
              max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
              max_depth=15, max_leaves=0, min_child_weight=3, missing=None,
              monotone_constraints='()', n_estimators=300, n_jobs=0,
              num_parallel_tree=1, predictor='auto', random_state=0,
                    sampling_method='uniform',
                     tree_method='hist', subsample=0.5, )
model_3 = RandomForestClassifier(criterion="entropy", max_depth=500,␣
 ↪min_samples_split=10, n_estimators=5000)
#model_4 = AdaBoostClassifier(n_estimators=150,learning_rate=0.
 ↪01,algorithm="SAMME")
#model_5 = RidgeClassifier(alpha=5.0,fit_intercept=True, copy_X=True,␣
 ↪max_iter=5000, tol=0.001, class_weight="balanced", solver='auto')
```

[43]:

```
# # Making the final model using voting classifier
# final_model = VotingClassifier(
#     estimators=[('lr', model_1), ('xgb', model_2), ('rf', model_3)],␣
 ↪voting='soft', weights = [1.35,1.67, 0.65])
```

[44]:

```
#final_model = VotingClassifier(estimators=[('lr', model_1), ('xgb', model_2),␣
 ↪('rf', model_3),('ad', model_4) ,('rc', model_5)], voting='hard', weights =␣
 ↪[1.9,1.7,1.67,1.33,1.45])
final_model = VotingClassifier(estimators=[('lr', model_1), ('xgb', model_2),␣
 ↪('rf', model_3)], voting='hard', weights = [1,1,2])
```

[45]:

```
#Model Fitting
final_model.fit(X_train, y_train)
```

[45]:

```
VotingClassifier(estimators=[('lr',
                              LogisticRegression(max_iter=2000,
                                                 multi_class='multinomial',
                                                 solver='sag')),
                             ('xgb',
                              XGBClassifier(alpha=7, base_score=None,
                                            booster='gbtree', callbacks=None,
                                            colsample_bylevel=1,
                                            colsample_bynode=1,
                                            colsample_bytree=0.7,
                                            early_stopping_rounds=None,
                                            enable_categorical=False, eta=0.2,
                                            eval_metric=None,
                                            feature_types=None, gamma=0…
                                            interaction_constraints='',
                                            learning_rate=0.05, max_bin=256,
                                            max_cat_threshold=64,
                                            max_cat_to_onehot=4,
                                            max_delta_step=0, max_depth=15,
                                            max_leaves=0, min_child_weight=3,
                                            missing=None,
                                            monotone_constraints='()',
                                            n_estimators=300, n_jobs=0,
                                            num_parallel_tree=1, …)),
                             ('rf',
                              RandomForestClassifier(criterion='entropy',
                                                     max_depth=500,
                                                     min_samples_split=10,
                                                     n_estimators=5000))],
                 weights=[1, 1, 2])
```

[45]:

```
# params = {'voting':['hard', 'soft'],
#           'weights':[(1,1,1,1), (2,1,1,1),
#                      (1,2,1,1), (1,1,2,1),
#                      (1,1,1,2), (1,2,1,1),
#                      (1,1,2,2), (2,1,1,2)]}

# #fit gridsearch & print best params
# grid = GridSearchCV(final_model, params)
# grid.fit(X_train, y_train)
# print('\n')
# print(f'The best params is : {grid.best_params_}')
```

```
[ ]: # Best params came out to be 'hard' and (1,1,2,1)
```

```
[46]: y_pred_90 = final_model.predict(X_test1)
```

```
[47]: y_pred_90
```

```
[47]: array([1, 0, 0, …, 0, 0, 0])
```

```
[48]: y_pred_check = final_model.predict(X_test)
```

```
[49]: from sklearn.metrics import accuracy_score, f1_score

      print('Testing accuracy %s' % accuracy_score(y_test, y_pred_check))
      print('Testing F1 score : {}'.format(f1_score(y_test, y_pred_check,␣
       ↪average='weighted')))
```

```
Testing accuracy 0.8040540540540541
Testing F1 score : 0.8027717095518264
```

```
[50]: y_pred_90
```

```
[50]: array([1, 0, 0, …, 0, 0, 0])
```

```
[51]: x = pd.DataFrame(y_pred_90)
```

```
[52]: x.value_counts()
```

```
[52]: 0    1695
      1    1263
      dtype: int64
```

```
[53]: final_sub = pd.concat([df_test['link_id'],x],axis=1)
```

```
[54]: final_sub.to_csv("final_sub_list.csv")
```

```
[55]: y_pred_t = final_model.predict(X_train)
```

```
[56]: y_pred_t
```

```
[56]: array([1, 1, 1, …, 1, 0, 0])
```

```
[57]: print('Testing accuracy %s' % accuracy_score(y_train, y_pred_t))
```

```
Testing accuracy 0.996135265700483
```

10 2. Modelling using Logistic Regression

[ ]:
```
# #Initializing the model
# lr = LogisticRegression()
```

[ ]:
```
# from sklearn.model_selection import GridSearchCV
```

[ ]:
```
# #Best Params after Hyperparameter tuning
# GS = GridSearchCV(estimator = lr,
#                   param_grid = search_space,
#                   scoring = ["r2","neg_root_mean_squared_error","accuracy"],
#                   refit = "r2",
#                   cv = 5,
#                   verbose = 4)
```

[ ]:
```
# #Model Fitting
# GS.fit(X_train, y_train)
```

[ ]:
```
# #Best Params
# GS.best_params_
```

[ ]:
```
# #Best Score
# GS.best_score_
```

[29]:
```
# y_train, X_train = vector_for_learning(model_dbow, train_documents)
# y_test, X_test = vector_for_learning(model_dbow, test_documents)

# logreg = LogisticRegression(max_iter=50,
#  multi_class= multinomial,
#  penalty= 'l1',
#  solver= 'saga')
# logreg.fit(X_train, y_train)
# y_pred = logreg.predict(X_test)
# from sklearn.metrics import accuracy_score, f1_score
# print('Testing accuracy %s' % accuracy_score(y_test, y_pred))
# print('Testing F1 score : {}'.format(f1_score(y_test, y_pred,␣
 ↪average='weighted')))
```

```
Testing accuracy 0.7905405405405406
Testing F1 score : 0.7904502155254269
```

```
/home/ibab/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[33]:
```
# #Test Prediction
# y_pred_t = logreg.predict(X_train)
```

[34]:
```
# print('Testing accuracy %s' % accuracy_score(y_train, y_pred_t))
```

```
Testing accuracy 0.879549114331723
```

[35]:
```
# #Vector Learning
# y_test1,X_test1 = vector_for_learning(model_dbow,test_documents_final)
```

[36]:
```
# # Final prediction
# y_pred_final = logreg.predict(X_test1)
```

[37]:
```
# y_pred_final
```

[37]:
```
array([1, 0, 0, …, 0, 0, 0])
```

[38]:
```
# x = pd.DataFrame(y_pred_final)
```

[40]:
```
# x.value_counts()
```

[40]:
```
0    1569
1    1389
dtype: int64
```

[42]:
```
# final_sub = pd.concat([df_test['link_id'],x],axis=1)
```

[43]:
```
# final_sub
```

[43]:
```
      link_id  0
0        4049  1
1        3692  0
2        9739  0
3        1548  1
4        5574  1
…         … ..
2953     4257  0
2954    10236  0
2955     5494  0
```

```
2956      9302   0
2957      2633   0

[2958 rows x 2 columns]
```

[44]:
```
# final_sub.to_csv('final_sub_4.csv')
```

11 3. Modelling using XGBClassifier

[45]:
```
# from xgboost import XGBClassifier
# model = XGBClassifier()
# model.fit(X_train, y_train)
```

[45]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
…)
```

[47]:
```
# y_pred1 = model.predict(X_test)
# from sklearn.metrics import accuracy_score, f1_score
# print('Testing accuracy %s' % accuracy_score(y_test, y_pred1))
# print('Testing F1 score : {}'.format(f1_score(y_test, y_pred1,␣
 ↪average='weighted')))
```

```
Testing accuracy 0.7912912912912913
Testing F1 score : 0.7913294116033841
```

[50]:
```
# y_pred_final_1 = model.predict(X_test1)
```

[52]:
```
# x1 = pd.DataFrame(y_pred_final_1)
```

[53]:
```
# x1.value_counts()
```

[53]:
```
0    1509
1    1449
dtype: int64
```

[58]:
```
# final_sub34 = pd.concat([df_test['link_id'],x1],axis=1)
```

[59]:

```
# final_sub34.to_csv('final_sub_99.csv')
```

12 3. Modelling using Random Forest

[ ]:

```
# from sklearn.ensemble import RandomForestRegressor

#  # create regressor object
# regressor = RandomForestRegressor()
```

[ ]:

```
# from sklearn.model_selection import StratifiedKFold
# kf = StratifiedKFold(n_splits=10, shuffle=False)
# search_space = {'n_estimators':[100,200,500],
#                 'criterion':["gini", "entropy", "log_loss"],
#                 'max_depth':[None,50,100],
#                  'max_features':["sqrt", "log2", None],

#                 }
```

[ ]:

```
# from sklearn.model_selection import GridSearchCV
# GS1 = GridSearchCV(estimator = regressor,
#                   param_grid = search_space,
#                   scoring = ["r2","accuracy"],
#                   refit = "r2",
#                   cv = 5,
#                   verbose = 4)
```

[ ]:

```
# GS1.fit(X_train, y_train)
```

[ ]:

```
# # fit the regressor with x and y data
# regressor.fit(x_train, y_train)
```

[ ]:

```
# y_pred1 = model.predict(X_test)
# from sklearn.metrics import accuracy_score, f1_score
# print('Testing accuracy %s' % accuracy_score(y_test, y_pred1))
# print('Testing F1 score : {}'.format(f1_score(y_test, y_pred1,␣
↪average='weighted')))
```

[ ]:

```
# y_pred_final_1 = model.predict(X_test1)
```

[ ]:

```
# x1 = pd.DataFrame(y_pred_final_1)
```

13 4. Modelling using AdaBoost Classifier

[ ]:

```
# # Load libraries
# from sklearn.ensemble import AdaBoostClassifier

# # Import Support Vector Classifier
# from sklearn.svm import SVC
# #Import scikit-learn metrics module for accuracy calculation
# from sklearn import metrics
# svc=SVC(probability=True, kernel='linear')

# # Create adaboost classifer object
# abc =AdaBoostClassifier(n_estimators=100, base_estimator=svc,learning_rate=0.
↪25)

# model_addf = abc.fit(X_train, y_train)
# #Predict the response for test dataset
# y_pred = model_addf.predict(X_test)


# # Model Accuracy, how often is the classifier correct?
# print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

[ ]:

```

```

[ ]:

```
# # initializing all the model objects with default parameters
# model_1 = LogisticRegression(max_iter=700,
#  multi_class= 'multinomial',
#  penalty= 'l1',
#  solver= 'sag')
# model_2 = XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
#               colsample_bylevel=0.5, colsample_bynode=0.5, colsample_bytree=1,
#               early_stopping_rounds=None, enable_categorical=False,
#               eval_metric=None, feature_types=None, gamma=0.4,eta=0.2,␣
↪gpu_id=-1,alpha=7,
#               grow_policy='depthwise', importance_type=None,
#               interaction_constraints='', learning_rate=0.05, max_bin=256,
#               max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
#               max_depth=15, max_leaves=0, min_child_weight=3, missing=None,
#               monotone_constraints='()', n_estimators=300, n_jobs=0,
#               num_parallel_tree=1, predictor='auto', random_state=0,
#                     sampling_method='uniform',
#                      tree_method='hist')
# model_3 = RandomForestClassifier(criterion="entropy", max_depth=100,␣
↪min_samples_split=5, n_estimators=1000)
# #model_4 = AdaBoostClassifier(n_estimators=150,learning_rate=0.
↪01,algorithm="SAMME")
```

[ ]:

```
# final_model = VotingClassifier(estimators=[('lr', model_1), ('xgb', model_2),␣
 ↪('rf', model_3)], voting='hard', weights = [1, 2, 3])
```

[ ]:

```

```