Chronus Data Analysis

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import os
```

In [2]:

```

# Read the data for cell A

# input_dir_dvv is the file location for dvv data folder
# input_dir_dv is the file location for dv data folder


input_dir_dvv = "/Users/jayashrijagannathan/Documents/chronus_project/An
input_dir_dv = "/Users/jayashrijagannathan/Documents/chronus_project/Ana

dvv_files = [f for f in os.listdir(input_dir_dvv)]
dv_files = [f for f in os.listdir(input_dir_dv)]

df_list = []  # initialize dataframes list

for dvv_f, dv_f in zip(dvv_files, dv_files):


    dvv_df = pd.read_csv(input_dir_dvv + "/" + dvv_f)
    dv_df = pd.read_csv(input_dir_dv + "/" + dv_f)


    # Here we combine the data of 106999 frequency dvv data with 2799999
    dvv_df = dvv_df[["real_time_106999","real_data_106999","imag_data_10
    dv_df = dv_df[["real_data_27999999","imag_data_27999999"]]

    # Scaling the dv data to match the dvv data
    dv_df = dv_df.apply(lambda x: x * 10)

    # Use the combined df data to proceed with analysis.
    combined_df = pd.concat([dvv_df,dv_df], axis=1, sort=False)
    df_list.append(combined_df)

concat_df = pd.concat(df_list, axis = 0)
```

In [3]:
```
concat_df.describe()
```

Out[3]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	422382.000000	422382.000000	422382.000000	422382.000000	422382.0
mean	5.120791	0.000078	0.000184	0.000429	0.0
std	2.693977	0.001900	0.003402	0.002649	0.0
min	0.009220	-0.008488	-0.013550	-0.011477	-0.0
25%	3.108961	-0.000701	-0.000916	-0.000718	-0.0
50%	5.176916	0.000006	0.000032	0.000013	0.0
75%	7.345117	0.000725	0.000991	0.000807	0.0
max	9.998788	0.008563	0.017257	0.019669	0.0

In [4]:
```
num_records = concat_df['real_time_106999'].count()
print(num_records)
```

```
422382
```

In [5]:
```
concat_df['cell_type'] = [0 for x in range(num_records)]
concat_df.describe()
```

Out[5]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	422382.000000	422382.000000	422382.000000	422382.000000	422382.0
mean	5.120791	0.000078	0.000184	0.000429	0.0
std	2.693977	0.001900	0.003402	0.002649	0.0
min	0.009220	-0.008488	-0.013550	-0.011477	-0.0
25%	3.108961	-0.000701	-0.000916	-0.000718	-0.0
50%	5.176916	0.000006	0.000032	0.000013	0.0
75%	7.345117	0.000725	0.000991	0.000807	0.0
max	9.998788	0.008563	0.017257	0.019669	0.0

In [6]:
```
# Read the data for cell B

# input_dir_dvv is the file location for dvv data folder
# input_dir_dv is the file location for dv data folder


input_dir_dvv = "/Users/jayashrijagannathan/Documents/chronus_project/An
input_dir_dv = "/Users/jayashrijagannathan/Documents/chronus_project/Ana

dvv_files = [f for f in os.listdir(input_dir_dvv)]
dv_files = [f for f in os.listdir(input_dir_dv)]

df_list = []  # initialize dataframes list

for dvv_f, dv_f in zip(dvv_files, dv_files):


    dvv_df = pd.read_csv(input_dir_dvv + "/" + dvv_f)
    dv_df = pd.read_csv(input_dir_dv + "/" + dv_f)


    # Here we combine the data of 106999 frequency dvv data with 2799999
    dvv_df = dvv_df[["real_time_106999","real_data_106999","imag_data_10
    dv_df = dv_df[["real_data_27999999","imag_data_27999999"]]

    # Scaling the dv data to match the dvv data
    dv_df = dv_df.apply(lambda x: x * 10)

    # Use the combined df data to proceed with analysis.
    combined_df = pd.concat([dvv_df,dv_df], axis=1, sort=False)
    df_list.append(combined_df)

concat_df1 = pd.concat(df_list, axis = 0)
```

In [7]:
```
concat_df1.describe()
```

Out[7]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	389250.000000	389250.000000	389250.000000	389250.000000	389250.0
mean	5.070524	0.000189	0.000651	0.001054	0.0
std	2.815230	0.004440	0.007685	0.005316	0.0
min	0.017045	-0.029131	-0.044101	-0.017850	-0.0
25%	3.126056	-0.000842	-0.000998	-0.000743	-0.0
50%	4.821506	-0.000002	0.000016	-0.000029	0.0
75%	7.451487	0.000831	0.001055	0.000788	0.0
max	9.999300	0.033728	0.062087	0.033611	0.0

In [8]:
```
num_records = concat_df1['real_time_106999'].count()
print(num_records)
```

```
389250
```

In [9]:
```
concat_df1['cell_type'] = [1 for x in range(num_records)]
concat_df1.describe()
```

Out[9]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	389250.000000	389250.000000	389250.000000	389250.000000	389250.0
mean	5.070524	0.000189	0.000651	0.001054	0.0
std	2.815230	0.004440	0.007685	0.005316	0.0
min	0.017045	-0.029131	-0.044101	-0.017850	-0.0
25%	3.126056	-0.000842	-0.000998	-0.000743	-0.0
50%	4.821506	-0.000002	0.000016	-0.000029	0.0
75%	7.451487	0.000831	0.001055	0.000788	0.0
max	9.999300	0.033728	0.062087	0.033611	0.0

In [10]:
```
# Read the data for cell C

# input_dir_dvv is the file location for dvv data folder
# input_dir_dv is the file location for dv data folder


input_dir_dvv = "/Users/jayashrijagannathan/Documents/chronus_project/An
input_dir_dv = "/Users/jayashrijagannathan/Documents/chronus_project/Ana

dvv_files = [f for f in os.listdir(input_dir_dvv)]
dv_files = [f for f in os.listdir(input_dir_dv)]

df_list = []  # initialize dataframes list

for dvv_f, dv_f in zip(dvv_files, dv_files):


    dvv_df = pd.read_csv(input_dir_dvv + "/" + dvv_f)
    dv_df = pd.read_csv(input_dir_dv + "/" + dv_f)


    # Here we combine the data of 106999 frequency dvv data with 2799999
    dvv_df = dvv_df[["real_time_106999","real_data_106999","imag_data_10
    dv_df = dv_df[["real_data_27999999","imag_data_27999999"]]

    # Scaling the dv data to match the dvv data
    dv_df = dv_df.apply(lambda x: x * 10)

    # Use the combined df data to proceed with analysis.
    combined_df = pd.concat([dvv_df,dv_df], axis=1, sort=False)
    df_list.append(combined_df)

concat_df2 = pd.concat(df_list, axis = 0)
```

In [13]:
```
concat_df2.describe()
```

Out[13]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	289621.000000	289621.000000	289621.000000	289621.000000	289621.0(
mean	5.187166	0.000329	0.000617	0.001723	0.0(
std	2.801033	0.007311	0.011412	0.009931	0.0(
min	0.012892	-0.043792	-0.067887	-0.039366	-0.0
25%	2.402103	-0.000721	-0.000991	-0.000886	-0.0(
50%	5.394943	0.000015	-0.000007	-0.000117	0.0(
75%	7.749698	0.000770	0.001012	0.000717	0.0(
max	9.997930	0.052416	0.080439	0.073876	0.0

In [14]:
```
num_records = concat_df2['real_time_106999'].count()
print(num_records)
```

```
289621
```

In [15]:
```
concat_df2['cell_type'] = [2 for x in range(num_records)]
concat_df2.describe()
```

Out[15]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_279
count	289621.000000	289621.000000	289621.000000	289621.000000	289621.0(
mean	5.187166	0.000329	0.000617	0.001723	0.0(
std	2.801033	0.007311	0.011412	0.009931	0.0(
min	0.012892	-0.043792	-0.067887	-0.039366	-0.0
25%	2.402103	-0.000721	-0.000991	-0.000886	-0.0(
50%	5.394943	0.000015	-0.000007	-0.000117	0.0(
75%	7.749698	0.000770	0.001012	0.000717	0.0(
max	9.997930	0.052416	0.080439	0.073876	0.0

In [16]:
```
frames = [concat_df, concat_df1, concat_df2]
final_df = pd.concat(frames)
```

In [17]:
```
final_df.describe()
```

Out[17]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_2799
count	1.101253e+06	1.101253e+06	1.101253e+06	1.101253e+06	1.10125
mean	5.120479e+00	1.832169e-04	4.627595e-04	9.900060e-04	1.61581
std	2.765931e+00	4.734893e-03	7.720983e-03	6.235154e-03	3.95202
min	9.220000e-03	-4.379175e-02	-6.788670e-02	-3.936649e-02	-4.11156
25%	2.946435e+00	-7.537055e-04	-9.647404e-04	-7.703195e-04	-6.92662
50%	5.077063e+00	6.025665e-06	1.718565e-05	-3.546471e-05	1.92559
75%	7.489495e+00	7.731317e-04	1.018462e-03	7.795763e-04	7.71254
max	9.999300e+00	5.241596e-02	8.043873e-02	7.387561e-02	4.21115

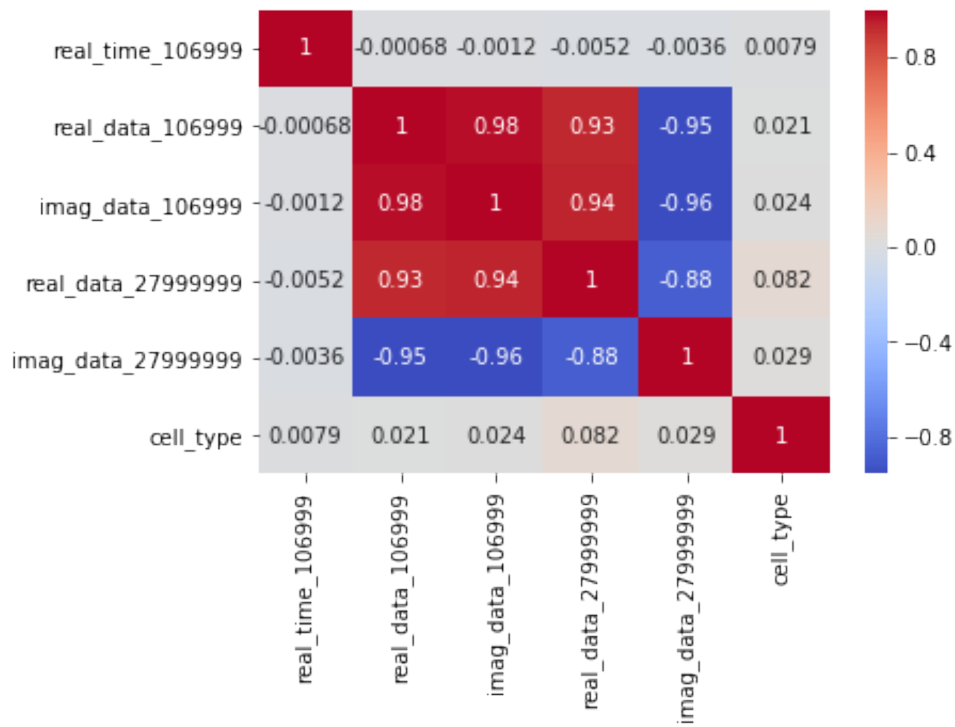
In [18]:
```
# Get the correlation matrix
corrMatrix = final_df.corr()
print(corrMatrix)
```

```
                    real_time_106999  real_data_106999  imag_data_1069
99  \
real_time_106999            1.000000         -0.000679         -0.0012
37
real_data_106999           -0.000679          1.000000          0.9833
39
imag_data_106999           -0.001237          0.983339          1.0000
00
real_data_27999999         -0.005224          0.933277          0.9435
11
imag_data_27999999         -0.003626         -0.953801         -0.9575
33
cell_type                   0.007922          0.020980          0.0240
67

                    real_data_27999999  imag_data_27999999  cell_type
real_time_106999             -0.005224           -0.003626   0.007922
real_data_106999              0.933277           -0.953801   0.020980
imag_data_106999              0.943511           -0.957533   0.024067
real_data_27999999            1.000000           -0.878623   0.082289
imag_data_27999999           -0.878623            1.000000   0.028896
cell_type                     0.082289            0.028896   1.000000
```

In [19]:
```
sns.heatmap(corrMatrix, annot = True, cmap = 'coolwarm')
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a15ddf898>
```



In [20]:
```
final_df.head()
```

Out[20]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_2799999
0	9.797785	-0.001080	0.000197	-0.000086	-0.00021
1	9.797788	-0.001103	0.000159	-0.000018	-0.00027
2	9.797790	-0.001122	0.000124	0.000066	-0.00034
3	9.797793	-0.001135	0.000093	0.000161	-0.00041
4	9.797795	-0.001143	0.000070	0.000259	-0.00048

In [21]:
```
# Find out if there is missing data
print(final_df.isnull().sum())
```

```
real_time_106999      0
real_data_106999      0
imag_data_106999      0
real_data_27999999    0
imag_data_27999999    0
cell_type             0
dtype: int64
```

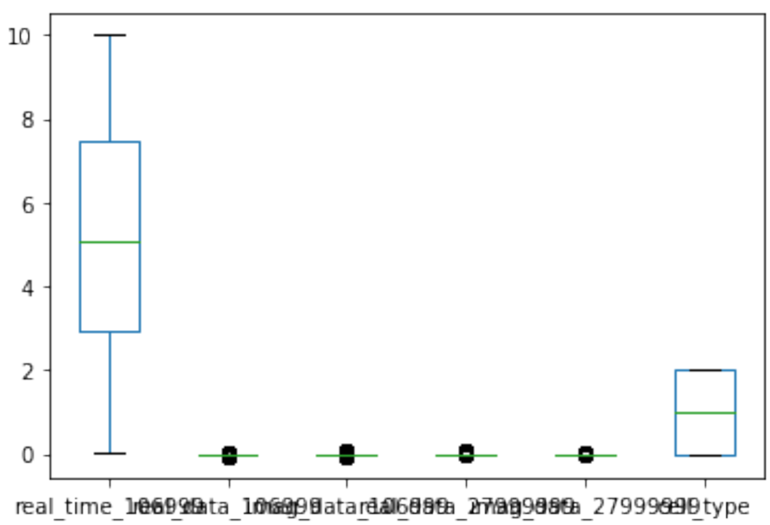
Observation: There is no missing data

In [22]:
```
final_df.columns
```

Out[22]:
```
Index(['real_time_106999', 'real_data_106999', 'imag_data_106999',
       'real_data_27999999', 'imag_data_27999999', 'cell_type'],
      dtype='object')
```

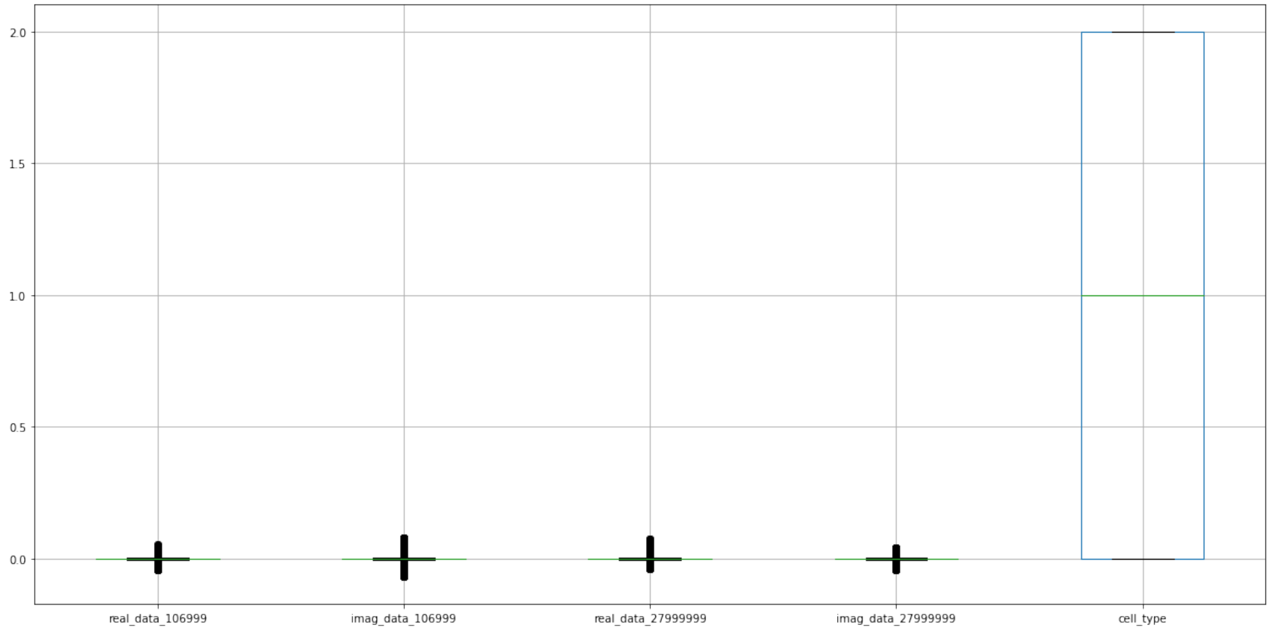
In [23]:
```
final_df.plot.box()
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a160449e8>
```



In [24]:
```
# Drop real_time_106999

boxplot = final_df.boxplot(column = ['real_data_106999', 'imag_data_1069
                    'imag_data_27999999', 'cell_type'], figsize = (20, 1
```



In [25]:
```
#Preprocessing for data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [26]:
```
# Do the scaler fit for all variables except cell_type (i.e. the depende
# and real_time_106999
final_df = final_df[['real_data_106999', 'imag_data_106999','real_data_2
                    'imag_data_27999999', 'cell_type']]
```

In [27]:
```
final_df.columns
```

Out[27]:
```
Index(['real_data_106999', 'imag_data_106999', 'real_data_27999999',
       'imag_data_27999999', 'cell_type'],
      dtype='object')
```

In [28]:
```
# Separate the pandas dataframe into input and output components
X = final_df[['real_data_106999', 'imag_data_106999', 'real_data_2799999
Y = final_df['cell_type']
```

In [29]:
```
scaler.fit(X)
```

Out[29]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [30]:
```
scaled_features = scaler.transform(final_df[['real_data_106999', 'imag_d
                                               'real_data_27999999','imag_
```

In [31]:
```
df_feat=pd.DataFrame(scaled_features, columns = final_df.columns[:-1] )
df_feat.head()
```

Out[31]:

	real_data_106999	imag_data_106999	real_data_27999999	imag_data_27999999
0	-0.266839	-0.034465	-0.172628	-0.095304
1	-0.271733	-0.039367	-0.161736	-0.110536
2	-0.275579	-0.043911	-0.148214	-0.127548
3	-0.278400	-0.047841	-0.132992	-0.145318
4	-0.280180	-0.050887	-0.117244	-0.162631

Our data is fitted and scaled and now it is ready

In [32]:
```
# Define X and Y
X = df_feat
y = final_df['cell_type']
```

In [33]:
```
# import train test split and metrics for evaluation
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
```

In [35]:
```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
```

In [36]:
```
classifier.fit(X_train, y_train)
```

Out[36]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [37]:
```
y_pred = classifier.predict(X_test)
```

In [38]:
```
#Summary of predictions made by the Decision Tree Classifier
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.63      0.66      0.64    126569
           1       0.57      0.55      0.56    116658
           2       0.51      0.49      0.50     87149

    accuracy                           0.58    330376
   macro avg       0.57      0.57      0.57    330376
weighted avg       0.58      0.58      0.58    330376
```

In [39]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[83022 24990 18557]
 [28713 64696 23249]
 [20956 23269 42924]]
```

In [40]:
```
from sklearn.metrics import accuracy_score
print('accuracy = ', accuracy_score(y_pred, y_test))
```

```
accuracy =  0.5770455481027678
```

In [41]:
```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[41]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [42]:
```
y_pred =classifier.predict(X_test)
```

In [43]:
```
#Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.42      0.81      0.55    126569
           1       0.29      0.10      0.15    116658
           2       0.48      0.24      0.32     87149

    accuracy                           0.41    330376
   macro avg       0.40      0.39      0.34    330376
weighted avg       0.39      0.41      0.35    330376
```

In [44]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[103062  23174    333]
 [ 82654  11777  22227]
 [ 60104   5901  21144]]
```

In [45]:
```
from sklearn.metrics import accuracy_score
print('accuracy = ', accuracy_score(y_pred, y_test))
```

```
accuracy =  0.4116007216020534
```

In [46]:

```
#K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier

neighbor_list = [10, 20, 30, 40, 50, 60, 70, 80, 100]
accuracy_list = []

print('K-Nearest Neighbors')
for neighbor in neighbor_list:
    knn_model = KNeighborsClassifier(n_neighbors = neighbor)
    knn_model.fit(X_train, y_train)
    y_pred = knn_model.predict(X_test)

    print("n_neighbors = ", neighbor)
    print(classification_report(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    accuracy = accuracy_score(y_pred, y_test)
    accuracy_list.append(accuracy_score)
    print("-------------------------------------------")
```

```
[[95207 23179  8183]
 [42858 64375  9425]
 [32379 22087 32683]]
-------------------------------------------
n_neighbors =  50
              precision    recall  f1-score   support

           0       0.55      0.76      0.64    126569
           1       0.59      0.55      0.57    116658
           2       0.66      0.36      0.47     87149

    accuracy                           0.58    330376
   macro avg       0.60      0.56      0.56    330376
weighted avg       0.59      0.58      0.57    330376

[[96034 23034  7501]
 [43943 63921  8794]
 [33276 22125 31748]]
-------------------------------------------
```

In [47]:
```
print('K_Nearest Neighbors Results')
print('n_neighbors accuracy')
n = len(neighbor_list)
for i in range(n):
    print(neighbor_list[i], accuracy)
```

```
K_Nearest Neighbors Results
n_neighbors accuracy
10 0.5726051529166768
20 0.5726051529166768
30 0.5726051529166768
40 0.5726051529166768
50 0.5726051529166768
60 0.5726051529166768
70 0.5726051529166768
80 0.5726051529166768
100 0.5726051529166768
```

The accuracy seems to be the same for all values of n_neighbors.

Looking at the classification report,

n_neighbors = 10 seems to be have weighted average accuracy of 0.6.

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier

learning_rate_list = [0.05, 0.1, 0.15, 0.2, 0.25]
n_estimators_list = [10, 20, 30, 40, 50, 75, 100]

#initialize result list
res_learning_rate = []
res_n_estimators = []
res_train_accuracy = []
res_test_accuracy = []
count = 0

print('Gradient Boosting Regressor\n')
for estimator in n_estimators_list:
    for rate in learning_rate_list:
        gb = GradientBoostingClassifier(n_estimators = estimator, learni
                                        max_features = 2, max_depth = 2,
        gb.fit(X_train, y_train)
        res_learning_rate.append(rate)
```

```
        res_n_estimators.append(estimator)
        gb_train_score = gb.score(X_train, y_train)
        res_train_accuracy.append(gb_train_score)
        gb_test_score = gb.score(X_test, y_test)
        res_test_accuracy.append(gb_test_score)
        count += 1
        print(count, rate, estimator, gb_train_score, gb_test_score)
        print()
```

```
Gradient Boosting Regressor

1 0.05 10 0.5005947771174909 0.4987953120081362

2 0.1 10 0.5010942082848496 0.49925539385427514

3 0.15 10 0.5006038576841702 0.4988225536963944

4 0.2 10 0.5013627336137931 0.49945516623483543

5 0.25 10 0.5027637353300203 0.5006931496234593

6 0.05 20 0.501406839223378 0.49951570331985373

7 0.1 20 0.5021255012148501 0.5001119936072839

8 0.15 20 0.5060833310631917 0.5040680921132286

9 0.2 20 0.5090500819196837 0.5068830665665787

10 0.25 20 0.512130988471572 0.5099825653195147

11 0.05 30 0.5016468256284725 0.4996186163643848

12 0.1 30 0.5050792798332289 0.5028634041213648

13 0.15 30 0.5104290308311183 0.5083601714410247

14 0.2 30 0.5135851763640633 0.5112296292708913

15 0.25 30 0.5212349051794255 0.5186121267888708

16 0.05 40 0.5029440494397939 0.5008898951497688

17 0.1 40 0.5085350840665891 0.5064260115746907

18 0.15 40 0.5158695874958003 0.5135603070440952
```

```
19 0.2 40 0.5201452371779155 0.5179492457079207

20 0.25 40 0.5276613519407117 0.5248353391287502

21 0.05 50 0.504857454561493 0.5027695716395865

22 0.1 50 0.511150287270213 0.5088717098094293

23 0.15 50 0.520138751058859 0.5173953313800034

24 0.2 50 0.5254820159376917 0.5228073467806378

25 0.25 50 0.5316295595795438 0.5285916652541347

26 0.05 75 0.5085999452571551 0.5063866624694288

27 0.1 75 0.5194551141102925 0.5167657456958132

28 0.15 75 0.5297433961578825 0.5268784657481173

29 0.2 75 0.5346806299837717 0.5314823110637577

30 0.25 75 0.53819351206483 0.5349813545778144

31 0.05 100 0.5116678795709303 0.5093620601980774

32 0.1 100 0.5284786029418441 0.5258342010315519
```

In [ ]:

```
gb_result_df = pd.DataFrame({'n_estimators': res_n_estimators,
                             'learning_rate': res_learning_rate,
                             'train_accuracy': res_train_accuracy,
                             'test_accuracy': res_test_accuracy})
gb_result_df.head()
```

In [ ]:

```
max_test_accuracy = gb_result_df['test_accuracy'].max()
print(gb_result_df[gb_result_df['test_accuracy'] == max_test_accuracy])
```

In [106]:
```
# Read the data for cell Mixed_Cells_Type_1

# input_dir_dvv is the file location for dvv data folder
# input_dir_dv is the file location for dv data folder


input_dir_dvv = "/Users/jayashrijagannathan/Documents/chronus_project/An
input_dir_dv = "/Users/jayashrijagannathan/Documents/chronus_project/Ana

dvv_files = [f for f in os.listdir(input_dir_dvv)]
dv_files = [f for f in os.listdir(input_dir_dv)]

df_list = []  # initialize dataframes list

for dvv_f, dv_f in zip(dvv_files, dv_files):


    dvv_df = pd.read_csv(input_dir_dvv + "/" + dvv_f)
    dv_df = pd.read_csv(input_dir_dv + "/" + dv_f)


    # Here we combine the data of 106999 frequency dvv data with 2799999
    dvv_df = dvv_df[["real_time_106999","real_data_106999","imag_data_10
    dv_df = dv_df[["real_data_27999999","imag_data_27999999"]]

    # Scaling the dv data to match the dvv data
    dv_df = dv_df.apply(lambda x: x * 10)

    # Use the combined df data to proceed with analysis.
    combined_df = pd.concat([dvv_df,dv_df], axis=1, sort=False)
    df_list.append(combined_df)

concat_mixed_df1 = pd.concat(df_list, axis = 0)
```

In [107]:
```
# Read the data for cell Mixed_Cells_Type_2

# input_dir_dvv is the file location for dvv data folder
# input_dir_dv is the file location for dv data folder


input_dir_dvv = "/Users/jayashrijagannathan/Documents/chronus_project/An
input_dir_dv = "/Users/jayashrijagannathan/Documents/chronus_project/Ana

dvv_files = [f for f in os.listdir(input_dir_dvv)]
dv_files = [f for f in os.listdir(input_dir_dv)]

df_list = []  # initialize dataframes list

for dvv_f, dv_f in zip(dvv_files, dv_files):


    dvv_df = pd.read_csv(input_dir_dvv + "/" + dvv_f)
    dv_df = pd.read_csv(input_dir_dv + "/" + dv_f)


    # Here we combine the data of 106999 frequency dvv data with 2799999
    dvv_df = dvv_df[["real_time_106999","real_data_106999","imag_data_10
    dv_df = dv_df[["real_data_27999999","imag_data_27999999"]]

    # Scaling the dv data to match the dvv data
    dv_df = dv_df.apply(lambda x: x * 10)

    # Use the combined df data to proceed with analysis.
    combined_df = pd.concat([dvv_df,dv_df], axis=1, sort=False)
    df_list.append(combined_df)

concat_mixed_df2 = pd.concat(df_list, axis = 0)
```

In [108]:
```
# Mixed Cell Type 1

concat_mixed_df1.describe()
```

Out[108]:

	real_time_106999	real_data_106999	imag_data_106999	real_data_27999999	imag_data_2799
count	909420.000000	909420.000000	909420.000000	909420.000000	9.09420(
mean	4.828389	0.000087	0.000345	0.000783	1.20319
std	2.834325	0.004388	0.007431	0.005996	3.84528
min	0.012960	-0.038856	-0.062456	-0.030852	-3.60458
25%	2.344528	-0.000741	-0.000900	-0.000749	-6.72135
50%	4.556157	-0.000018	-0.000016	-0.000053	3.89607
75%	7.463443	0.000705	0.000897	0.000699	6.95663
max	9.988067	0.040986	0.071008	0.061863	3.48860

In [109]:
```
#Decision Tree Model

dtree = DecisionTreeClassifier()
```

In [110]:
```
dtree.fit(X_train, y_train)
```

Out[110]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [111]:
```
y_pred = dtree.predict(X_test)
```