Assignment2_multi-classification of Network Traffic

Qing Zhao

1. Research Paper Review

1.1 Summarizing the Research Paper

In the paper titled "Characterization of Tor Traffic Using Time-Based Features," the authors proposed a method for detecting and characterizing Tor traffic by utilizing time-based features. The selected feature set was based solely on time statistics derived from the observation of traffic flows between a Tor client and a Tor entry node. The authors conducted controlled experiments to generate Tor traffic representative of the real world. They labeled the types of Tor traffic based on pairs of PCAP files—one representing regular traffic and the other Tor traffic. The controlled experiments involved over 18 representative applications, generating eight different types of Tor traffic, including browsing, chat, audio streaming, video streaming, mail, VOIP, P2P, and file transfer.

Building upon this, the authors utilized the ISCXFlowMeter tool to generate flows and processed 23 time-based features based on it. Additionally, to investigate the impact of flow timeout values on the final results, the authors set five different flow timeout values, resulting in the creation of five specific datasets with timeouts of 10s, 15s, 30s, 60s, and 120s.

The entire experiment comprised two scenarios. In Scenario A, the goal was to identify Tor traffic and non-Tor traffic. For the non-Tor traffic dataset, the authors introduced the encrypted traffic public dataset published by Draper-Gil and others in 2016. In Scenario B, the emphasis was on describing Tor traffic, specifically identifying different applications within Tor traffic. The authors applied Cfs-SubsetEval+BestFirst and Infogain+Ranker feature selection algorithms to each dataset and evaluated the model performance by computing weighted average precision and recall using 10-fold cross-validation.

In Scenario A, the authors employed three machine learning algorithms: ZeroR, C4.5, and KNN. Ultimately, the combination of the Infogain+Ranker feature selection algorithm with the C4.5 model achieved the best classification performance on the validation set, with a recall of 93.4% and precision of 94.8%.

For Scenario B, the authors tested the results obtained with classifiers of Random Forests, C4.5, and KNN, using 10-fold cross-validation. Random Forests classifier achieved the best results when combined with features selected by the IG+RK algorithm, with a weighted recall of 83.8% and weighted precision of 84.3%. Notably, classes such as VOIP, P2P, AUDIO, FT, and VIDEO obtained good precision and recall results. In contrast, the classifier performed less effectively in the classes of BROWSING, CHAT, and E-MAIL. Apart from that, the experiment results showed a relation between the flow timeout and the efficiency of the algorithms tested: the shorter the flow timeout, the better the results, with an optimal value of 15s.

Through the experiments, the authors arrived at the following conclusions. The first one was that time-base features could be used to detect Tor traffic efficiently. Then, time-base features could be used to characterize Tor traffic and efficiently detect different traffic applications. Finally, flow timeout influenced on the efficiency of the solution our classifiers performed better when the flows were generated using shorter timeout values, with 15s. as the optimal value, which contradicted the common assumption of using 600s as timeout duration.

1.2 A few thoughts about the research paper

When labeling Tor traffic, the authors only provided textual descriptions that Tor traffic flows exhibit some noise, leading to inaccurate labels that may impact the classifier's performance. However, they didn’t analyze or statistically quantify the noise components. Since the impact of noise on different types of labels may vary significantly, without assessing label accuracy, the classification results of experimental samples may not convincingly represent the true classification performance, especially for imbalanced and small samples where the noise effect was more pronounced.

For instance, in the conclusion about the relationship between flow timeout and the efficiency of tested algorithms, the influence of noise was more pronounced for small sample sets. As the timeout value increased, the sample size significantly decreased, with the 120s dataset being only 17% of the 15s dataset. Under the interference of noise, it became challenging to determine the universality of the experimental results.

Furthermore, the authors' explanations for misclassifications of different label types lacked empirical support. For example, the assumption that Browsing traffic may be the primary noise among all labels to explain the majority of misclassifications occurring in Browsing traffic lacked noise-related statistical analysis.

In future research, it is recommended to generate auxiliary labels for assessing both label accuracy and noise components when labeling Tor traffic. These auxiliary labels can be used for subsequent statistical analysis. For example, if Tor traffic labeled as Chat contains 1% of Browsing traffic, auxiliary labels can be generated indicating the label accuracy (99%) and noise type (Browsing traffic). Since the experiment involves generating pairs of PCAP files, this operation is feasible.

In this experiment, the authors employed Cfs-SubsetEval and Infogain for feature selection. Cfs-SubsetEval focuses on redundancy among features, and given the high correlation between time-based features, this method resulted in a small number of remaining features. As for Infogain, its treatment of continuous features is complex. Therefore, In Assignment 2, I will attempt to use the mutual information method combined with Ranker for feature selection, which is more simple.

2. Problem formulation and Machine Learning task

This assignment involves using 23 time-based features to identify 8 different types of traffic in Tor flows, namely Browsing, Email, Chat, Audio, Video, File Transfer, VoIP, and P2P. The dataset comprises 3360 samples, with an uneven distribution. The least represented label, Chat, accounts for only 1.3% of the total samples, while the most represented label, VoIP, constitutes as much as 44.9%.

Given the nature of the problem, I need to establish a multi-class machine learning model and consider appropriate adjustments for imbalanced samples during the learning process to prevent insufficient learning for less represented categories. To select the most effective classifiers, I will evaluate multiple classifiers using a 10-fold cross-validation method and choose the top two classifiers for further exploration.

In this task, I would like to make different attempts with the train dataset. The first one involves training models using different features. I will rank the features using mutual information evaluation and select the top-ranked feature combinations. During subsequent model training, I will compare the model performance using the selected feature combinations and all features. The second attempt involves training models using different train samples. By employing oversampling and undersampling techniques, I will adjust the imbalanced train samples and model them using oversampled, undersampled, and original distribution train datasets, respectively, and compare the model performance. Finally, I will fine-tune the models based on the optimal combinations obtained from these two explorations, using metrics such as weighted precision and F1 to evaluate the performance on the test dataset.

3. Dataset Exploration

3.1 Describing the data structure

The dataset consists of 3360 samples, including ID attribute, 23 time-based features, and a label column. All features are numerical variables, and the label is a categorical variable with 8 classes. The distribution among these classes is imbalanced shown as follows:

```
df.shape
```

```
(3360, 25)
```

```
df.info()
```

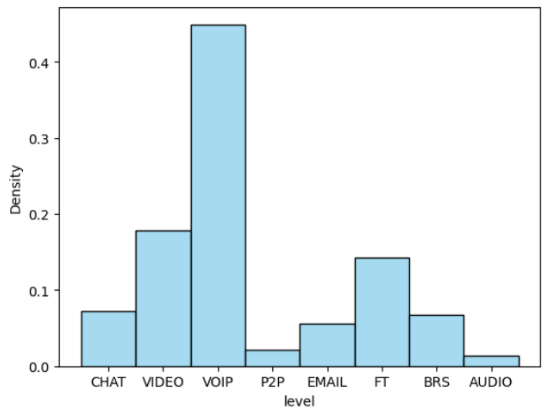
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3360 entries, 0 to 3359
Data columns (total 25 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID                  3360 non-null   int64
 1   duration            3360 non-null   int64
 2   total_fiat          3360 non-null   int64
 3   total_biat          3360 non-null   int64
 4   min_fiat            3360 non-null   int64
 5   min_biat            3360 non-null   int64
 6   max_fiat            3360 non-null   int64
 7   max_biat            3360 non-null   int64
 8   mean_fiat           3360 non-null   float64
 9   mean_biat           3360 non-null   float64
 10  flowPktsPerSecond   3360 non-null   float64
 11  flowBytesPerSecond  3360 non-null   float64
 12  min_flowiat         3360 non-null   int64
 13  max_flowiat         3360 non-null   int64
 14  mean_flowiat        3360 non-null   float64
 15  std_flowiat         3360 non-null   float64
 16  min_active          3360 non-null   int64
 17  mean_active         3360 non-null   float64
 18  max_active          3360 non-null   int64
 19  std_active          3360 non-null   float64
 20  min_idle            3360 non-null   int64
 21  mean_idle           3360 non-null   float64
 22  max_idle            3360 non-null   int64
 23  std_idle            3360 non-null   float64
 24  level               3360 non-null   object
dtypes: float64(10), int64(14), object(1)
memory usage: 656.4+ KB
```

```
df.level.value_counts(normalize=True)
```

```
VOIP               0.449107
VIDEO-STREAMING    0.177976
FILE-TRANSFER      0.142857
CHAT               0.072321
BROWSING           0.067560
EMAIL              0.055357
P2P                0.021131
AUDIO-STREAMING    0.013690
Name: level, dtype: float64
```

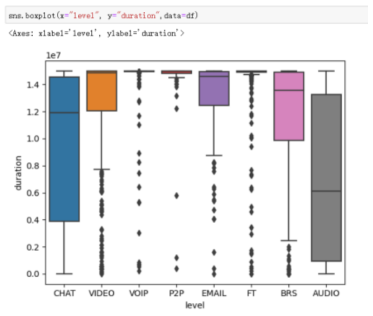
```
sns.histplot(df["level"],stat="density",color="skyblue")
```

```
<Axes: xlabel='level', ylabel='Density'>
```



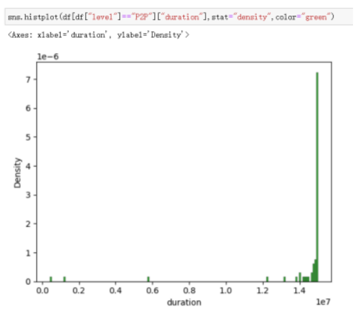
3.2 Visualizing the data and gaining some insights

The box plots illustrate substantial variations in the duration distribution across different types of traffic. Chat and Audio display relatively even distributions, whereas P2P shows a more concentrated distribution.



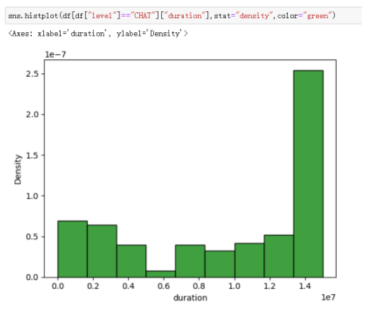
```
sns.boxplot(x="level", y="duration",data=df)
```

```
<Axes: xlabel='level', ylabel='duration'>
```



```
sns.histplot(df[df["level"]=="P2P"]["duration"],stat="density",color="green")
```

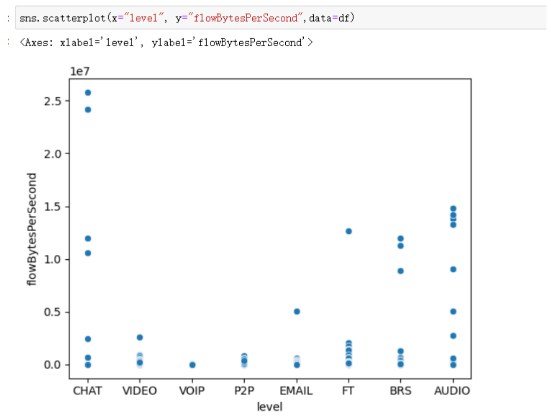
```
<Axes: xlabel='duration', ylabel='Density'>
```



```
sns.histplot(df[df["level"]=="CHAT"]["duration"],stat="density",color="green")
```

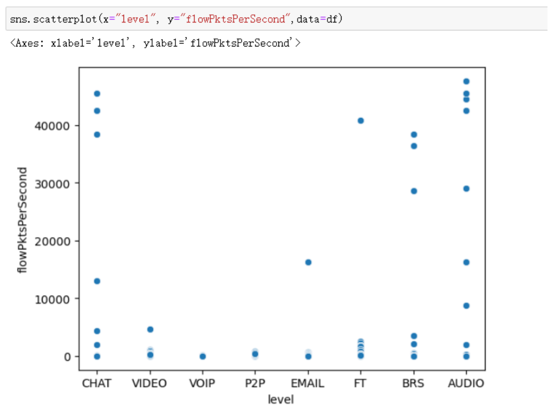
```
<Axes: xlabel='duration', ylabel='Density'>
```

The scatter plots indicate that there is a certain level of discrimination in the features of flowPktsPerSecond and flowBytesPerSecond, which can be considered to apply in subsequent model training.



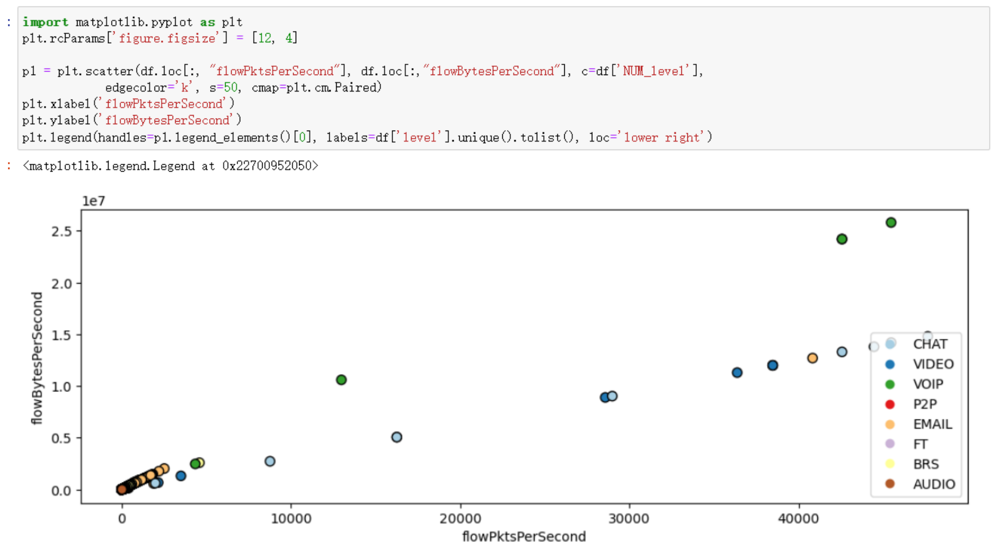
```
sns.scatterplot(x="level", y="flowBytesPerSecond",data=df)
```

```
<Axes: xlabel='level', ylabel='flowBytesPerSecond'>
```



```
sns.scatterplot(x="level", y="flowPktsPerSecond",data=df)
```

```
<Axes: xlabel='level', ylabel='flowPktsPerSecond'>
```



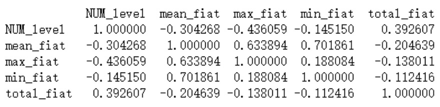
```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [12, 4]

p1 = plt.scatter(df.loc[:, "flowPktsPerSecond"], df.loc[:,"flowBytesPerSecond"], c=df['NUM_level'],
            edgecolor='k', s=50, cmap=plt.cm.Paired)
plt.xlabel('flowPktsPerSecond')
plt.ylabel('flowBytesPerSecond')
plt.legend(handles=p1.legend_elements()[0], labels=df['level'].unique().tolist(), loc='lower right')
```

```
<matplotlib.legend.Legend at 0x22700952050>
```

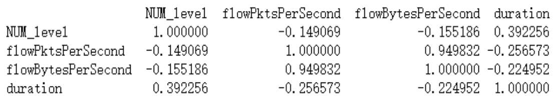
3.3 Analyzing feature correlation

After conducting a correlation analysis between features and the target variable, as well as among the features, I obtain the following information:

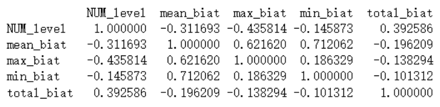
Firstly, through correlation analysis, the individual indicators' correlation with the target is not higher than 0.45, and most of them are negatively correlated.



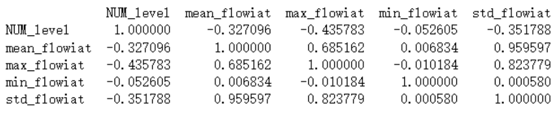
```
            NUM_level  mean_fiat  max_fiat  min_fiat  total_fiat
NUM_level    1.000000  -0.304268 -0.436059 -0.145150    0.392607
mean_fiat   -0.304268   1.000000  0.633894  0.701861   -0.204639
max_fiat    -0.436059   0.633894  1.000000  0.188084   -0.138011
min_fiat    -0.145150   0.701861  0.188084  1.000000   -0.112416
total_fiat   0.392607  -0.204639 -0.138011 -0.112416    1.000000
```



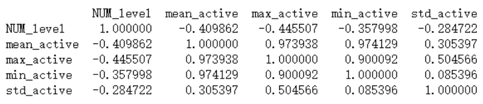
```
                    NUM_level  flowPktsPerSecond  flowBytesPerSecond  duration
NUM_level            1.000000          -0.149069           -0.155186  0.392256
flowPktsPerSecond   -0.149069           1.000000            0.949832 -0.256573
flowBytesPerSecond  -0.155186           0.949832            1.000000 -0.224952
duration             0.392256          -0.256573           -0.224952  1.000000
```



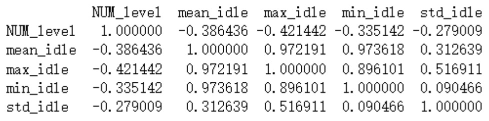
```
            NUM_level  mean_biat  max_biat  min_biat  total_biat
NUM_level    1.000000  -0.311693 -0.435814 -0.145873    0.392586
mean_biat   -0.311693   1.000000  0.621620  0.712062   -0.196209
max_biat    -0.435814   0.621620  1.000000  0.186329   -0.138294
min_biat    -0.145873   0.712062  0.186329  1.000000   -0.101312
total_biat   0.392586  -0.196209 -0.138294 -0.101312    1.000000
```



```
              NUM_level  mean_flowiat  max_flowiat  min_flowiat  std_flowiat
NUM_level      1.000000     -0.327096    -0.435783    -0.052605    -0.351788
mean_flowiat  -0.327096      1.000000     0.685162     0.006834     0.959597
max_flowiat   -0.435783      0.685162     1.000000    -0.010184     0.823779
min_flowiat   -0.052605      0.006834    -0.010184     1.000000     0.000580
std_flowiat   -0.351788      0.959597     0.823779     0.000580     1.000000
```

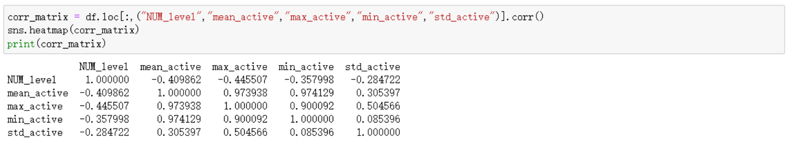


```
             NUM_level  mean_active  max_active  min_active  std_active
NUM_level     1.000000    -0.409862   -0.445507   -0.357998   -0.284722
mean_active  -0.409862     1.000000    0.973938    0.974129    0.305397
max_active   -0.445507     0.973938    1.000000    0.900092    0.504566
min_active   -0.357998     0.974129    0.900092    1.000000    0.085396
std_active   -0.284722     0.305397    0.504566    0.085396    1.000000
```



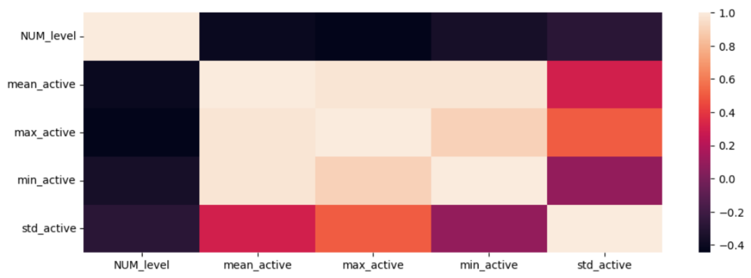
```
           NUM_level  mean_idle  max_idle  min_idle  std_idle
NUM_level   1.000000  -0.386436 -0.421442 -0.335142 -0.279009
mean_idle  -0.386436   1.000000  0.972191  0.973618  0.312639
max_idle   -0.421442   0.972191  1.000000  0.896101  0.516911
min_idle   -0.335142   0.973618  0.896101  1.000000  0.090466
std_idle   -0.279009   0.312639  0.516911  0.090466  1.000000
```

Secondly, features within the same dimension (with the same suffix) exhibit a high positive correlation, such as "_active".

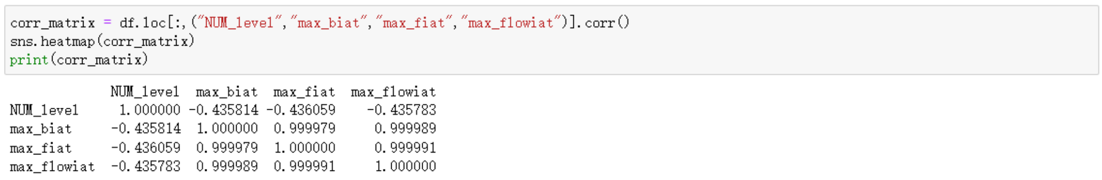


```
corr_matrix = df.loc[:,("NUM_level","mean_active","max_active","min_active","std_active")].corr()
sns.heatmap(corr_matrix)
print(corr_matrix)
```

```
             NUM_level  mean_active  max_active  min_active  std_active
NUM_level     1.000000    -0.409862   -0.445507   -0.357998   -0.284722
mean_active  -0.409862     1.000000    0.973938    0.974129    0.305397
max_active   -0.445507     0.973938    1.000000    0.900092    0.504566
min_active   -0.357998     0.974129    0.900092    1.000000    0.085396
std_active   -0.284722     0.305397    0.504566    0.085396    1.000000
```

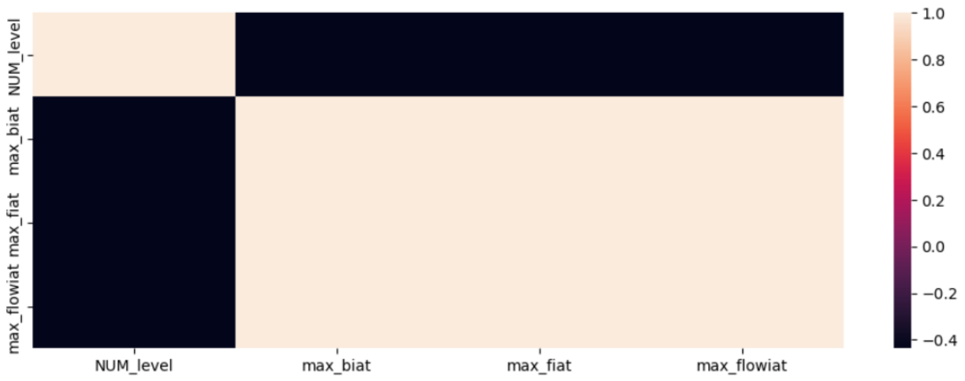


Additionally, features within the biat, fiat, and flowiat dimensions (with the same prefix) display a high positive correlation.



```
corr_matrix = df.loc[:,("NUM_level","max_biat","max_fiat","max_flowiat")].corr()
sns.heatmap(corr_matrix)
print(corr_matrix)
```

```
             NUM_level  max_biat  max_fiat  max_flowiat
NUM_level     1.000000 -0.435814 -0.436059    -0.435783
max_biat     -0.435814  1.000000  0.999979     0.999989
max_fiat     -0.436059  0.999979  1.000000     0.999991
max_flowiat  -0.435783  0.999989  0.999991     1.000000
```



4. Data Preparation

In the data preparation stage, I carried out five steps sequentially:

Step one was removing the ID attribute. Step two was splitting the dataset using stratified sampling, with 80% allocated to the training dataset and 20% to the testing dataset.

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2, random_state = True,stratify=Y)
```

Step three was standardizing the feature data.

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

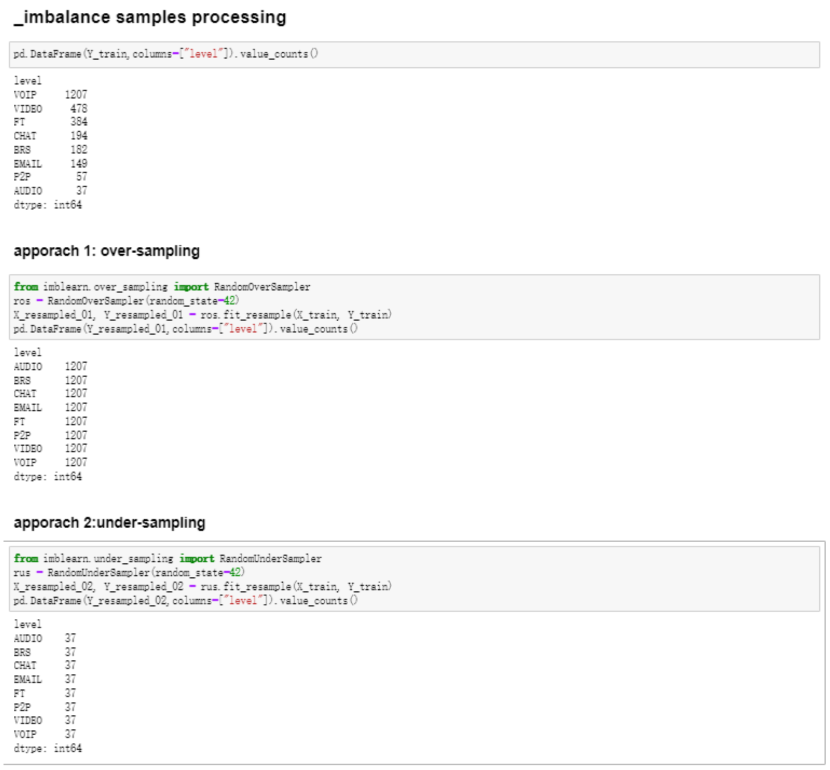
Step four was calculating feature importance using the mutual_info_classif method. From the results, the most significant decrease in weight between two consecutive features, specifically between min_fiat and total_biat. Consequently, the top 12 features were selected when training models. Showed as followed table:

Table 1: the ranker of features according to MutualInfomationValue.

ranker	features	MutualInfomationValue	gap
1	flowBytesPerSecond	0.7979	NILL
2	mean_flowiat	0.7678	0.030
3	mean_biat	0.7584	0.009
4	flowPktsPerSecond	0.7564	0.002
5	mean_fiat	0.7264	0.030
6	max_fiat	0.6631	0.063
7	max_flowiat	0.6346	0.028
8	max_biat	0.6231	0.011
9	min_biat	0.6063	0.017
10	std_flowiat	0.5929	0.013
11	min_fiat	0.5540	0.039
12	min_flowiat	0.4655	0.089
13	**total_fiat**	**0.3281**	**0.137**
14	total_biat	0.3278	0.000
15	duration	0.3184	0.009
16	max_idle	0.2976	0.021
17	mean_idle	0.2910	0.007
18	min_active	0.2894	0.002
19	mean_active	0.2861	0.003
20	max_active	0.2855	0.001
21	min_idle	0.2719	0.014

22	std_active	0.1176	0.154
23	std_idle	0.1035	0.014
24	std_active	0.1176	-0.014
25	std_idle	0.1035	0.014

Step five was conducting oversampling and undersampling separately on the training dataset to generate oversampled and undersampled training datasets. These sets were intended for subsequent model training or tuning purposes.



_imbalance samples processing

```
pd.DataFrame(Y_train,columns=["level"]).value_counts()
```

```
level
VOIP     1207
VIDEO     478
FT        384
CHAT      194
BRS       182
EMAIL     149
P2P        57
AUDIO      37
dtype: int64
```

apporach 1: over-sampling

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
X_resampled_01, Y_resampled_01 = ros.fit_resample(X_train, Y_train)
pd.DataFrame(Y_resampled_01,columns=["level"]).value_counts()
```

```
level
AUDIO    1207
BRS      1207
CHAT     1207
EMAIL    1207
FT       1207
P2P      1207
VIDEO    1207
VOIP     1207
dtype: int64
```

apporach 2:under-sampling

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_resampled_02, Y_resampled_02 = rus.fit_resample(X_train, Y_train)
pd.DataFrame(Y_resampled_02,columns=["level"]).value_counts()
```

```
level
AUDIO    37
BRS      37
CHAT     37
EMAIL    37
FT       37
P2P      37
VIDEO    37
VOIP     37
dtype: int64
```

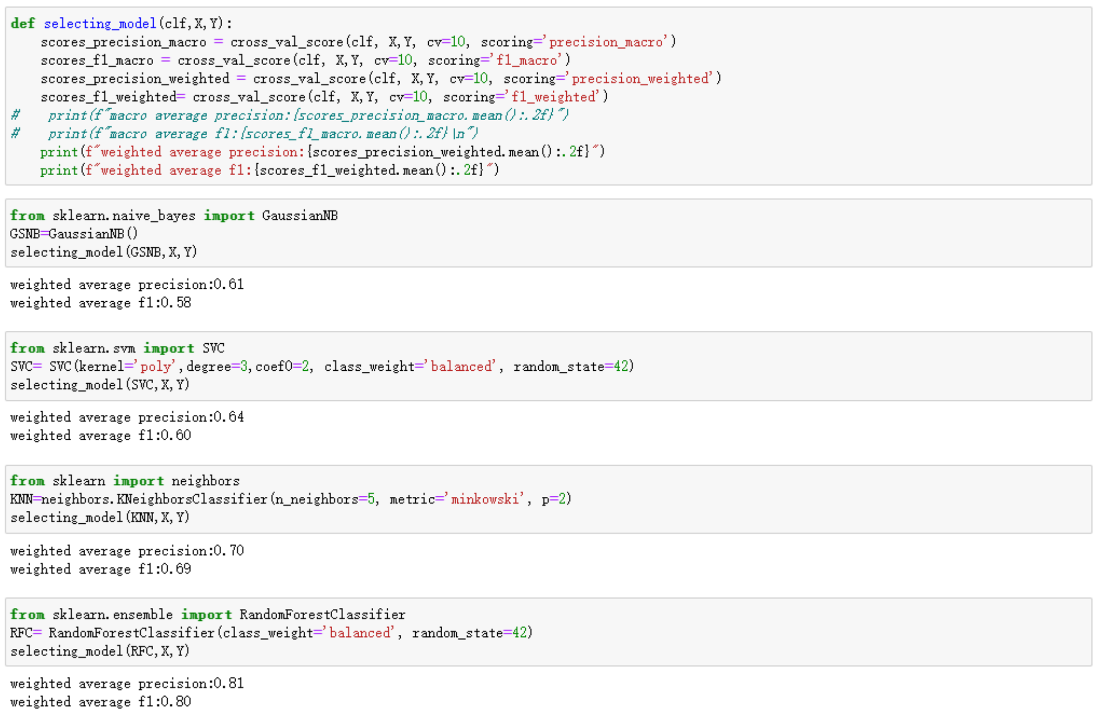
5. Model Building

5.1 Selecting the top-performing two models

Before starting the modeling process, I employed a 10-fold cross-validation method to select the top-performing two classifiers among GaussianNB, SVC, KNN, and RandomForest Classifier (RF), which served as the basis for subsequent model training. I used weighted precision and weighted F1 as metrics. Meanwhile, for SVC and RF, the parameter of class_weight="balanced" was applied to address sample imbalance. From the evaluations, KNN and RF with the best performance were selected for further modeling.

Table 2: the results of the 10-fold cross-validation method.

metrics	GaussianNB	SVC	KNN	RF
weighted precision	0.61	0.64	0.70	0.81
weighted f1	0.58	0.60	0.69	0.80



```
def selecting_model(clf,X,Y):
    scores_precision_macro = cross_val_score(clf, X,Y, cv=10, scoring='precision_macro')
    scores_f1_macro = cross_val_score(clf, X,Y, cv=10, scoring='f1_macro')
    scores_precision_weighted = cross_val_score(clf, X,Y, cv=10, scoring='precision_weighted')
    scores_f1_weighted= cross_val_score(clf, X,Y, cv=10, scoring='f1_weighted')
#    print(f"macro average precision:{scores_precision_macro.mean():.2f}")
#    print(f"macro average f1:{scores_f1_macro.mean():.2f}\n")
    print(f"weighted average precision:{scores_precision_weighted.mean():.2f}")
    print(f"weighted average f1:{scores_f1_weighted.mean():.2f}")
```

```
from sklearn.naive_bayes import GaussianNB
GSNB=GaussianNB()
selecting_model(GSNB,X,Y)
```

```
weighted average precision:0.61
weighted average f1:0.58
```

```
from sklearn.svm import SVC
SVC= SVC(kernel='poly',degree=3,coef0=2, class_weight='balanced', random_state=42)
selecting_model(SVC,X,Y)
```

```
weighted average precision:0.64
weighted average f1:0.60
```

```
from sklearn import neighbors
KNN=neighbors.KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
selecting_model(KNN,X,Y)
```

```
weighted average precision:0.70
weighted average f1:0.69
```

```
from sklearn.ensemble import RandomForestClassifier
RFC= RandomForestClassifier(class_weight='balanced', random_state=42)
selecting_model(RFC,X,Y)
```

```
weighted average precision:0.81
weighted average f1:0.80
```

5.2 Training models

In the training phase, I applied KNN and RF on two train datasets, one dataset with all 23 features and the other only with the selected 12 features mentioned above. From the performance, it appeared that The RF model trained on the top 12 features outperforms the one trained on all features, but the KNN model exhibits the opposite behavior.

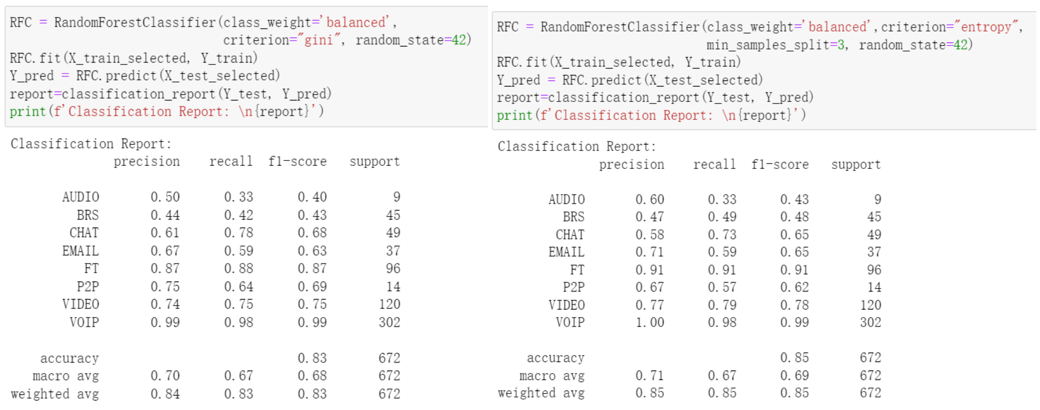
Table 3: the results of KNN and RF training with different features.

metrics	RF		KNN	
	all features	top12 features	all features	top12 features
weighted precision	0.83	**0.84**	**0.76**	0.73
weighted recall	0.83	**0.83**	**0.75**	0.72
weighted f1	0.83	**0.83**	**0.75**	0.72

5.3 Fine-tuning models

During the fine-tuning stage, I made different attempts to refine the RF and KNN models.

For the RF model, I improved the prediction performance by 0.02 by adjusting the criterion from “gini” to "entropy" and setting the min_samples_split equals to 3 as shown in the below figure.



```
RFC = RandomForestClassifier(class_weight='balanced',
                             criterion="gini", random_state=42)
RFC.fit(X_train_selected, Y_train)
Y_pred = RFC.predict(X_test_selected)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

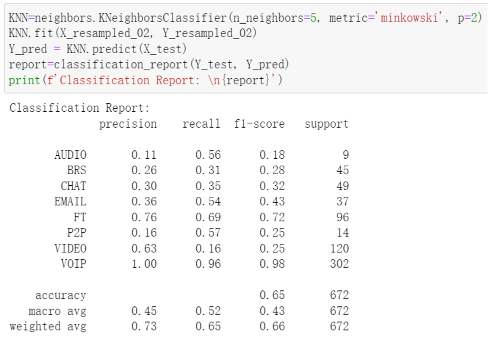
```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.50      0.33      0.40         9
         BRS       0.44      0.42      0.43        45
        CHAT       0.61      0.78      0.68        49
       EMAIL       0.67      0.59      0.63        37
          FT       0.87      0.88      0.87        96
         P2P       0.75      0.64      0.69        14
       VIDEO       0.74      0.75      0.75       120
        VOIP       0.99      0.98      0.99       302

    accuracy                           0.83       672
   macro avg       0.70      0.67      0.68       672
weighted avg       0.84      0.83      0.83       672
```

```
RFC = RandomForestClassifier(class_weight='balanced',criterion="entropy",
                             min_samples_split=3, random_state=42)
RFC.fit(X_train_selected, Y_train)
Y_pred = RFC.predict(X_test_selected)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.60      0.33      0.43         9
         BRS       0.47      0.49      0.48        45
        CHAT       0.58      0.73      0.65        49
       EMAIL       0.71      0.59      0.65        37
          FT       0.91      0.91      0.91        96
         P2P       0.67      0.57      0.62        14
       VIDEO       0.77      0.79      0.78       120
        VOIP       1.00      0.98      0.99       302

    accuracy                           0.85       672
   macro avg       0.71      0.67      0.69       672
weighted avg       0.85      0.85      0.85       672
```

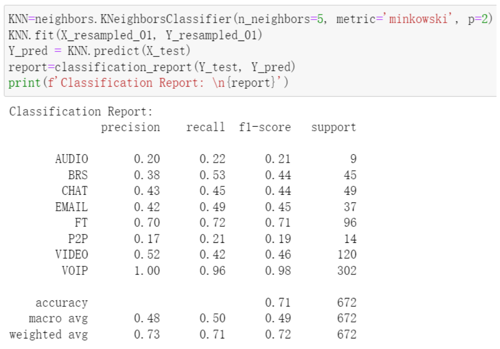
Figure 1: the comparison of before and after fine-tuning of RF

As for the KNN model, I experimented with more adjustments. Firstly, as KNN does not have parameters for adjusting imbalanced samples, I trained the model using oversampled and undersampled training datasets separately, which had been prepared in the data preparation phase. The results showed a decline in both oversampled and undersampled train datasets, compared with the original train dataset.



```
KNN=neighbors.KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
KNN.fit(X_resampled_02, Y_resampled_02)
Y_pred = KNN.predict(X_test)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.11      0.56      0.18         9
         BRS       0.26      0.31      0.28        45
        CHAT       0.30      0.35      0.32        49
       EMAIL       0.36      0.54      0.43        37
          FT       0.76      0.69      0.72        96
         P2P       0.16      0.57      0.25        14
       VIDEO       0.63      0.16      0.25       120
        VOIP       1.00      0.96      0.98       302

    accuracy                           0.65       672
   macro avg       0.45      0.52      0.43       672
weighted avg       0.73      0.65      0.66       672
```



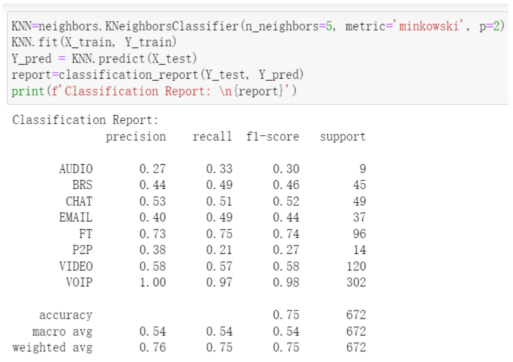
```
KNN=neighbors.KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
KNN.fit(X_resampled_01, Y_resampled_01)
Y_pred = KNN.predict(X_test)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.20      0.22      0.21         9
         BRS       0.38      0.53      0.44        45
        CHAT       0.43      0.45      0.44        49
       EMAIL       0.42      0.49      0.45        37
          FT       0.70      0.72      0.71        96
         P2P       0.17      0.21      0.19        14
       VIDEO       0.52      0.42      0.46       120
        VOIP       1.00      0.96      0.98       302

    accuracy                           0.71       672
   macro avg       0.48      0.50      0.49       672
weighted avg       0.73      0.71      0.72       672
```

Figure 2: Training on oversampled training datasets

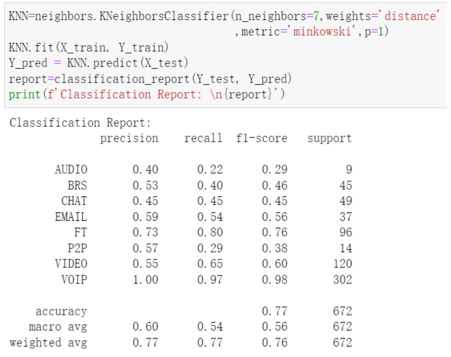
Figure 3: Training on undersampled training datasets

Then, I modified the distance calculation parameters to weighted distance, changed the distance calculation method to Manhattan distance, and set the neighbors to 7. This adjustment led to the optimal performance of the KNN model, with weighted precision and weighted F1 improving by 0.01.



```
KNN=neighbors.KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
KNN.fit(X_train, Y_train)
Y_pred = KNN.predict(X_test)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.27      0.33      0.30         9
         BRS       0.44      0.49      0.46        45
        CHAT       0.53      0.51      0.52        49
       EMAIL       0.40      0.49      0.44        37
          FT       0.73      0.75      0.74        96
         P2P       0.38      0.21      0.27        14
       VIDEO       0.58      0.57      0.58       120
        VOIP       1.00      0.97      0.98       302

    accuracy                           0.75       672
   macro avg       0.54      0.54      0.54       672
weighted avg       0.76      0.75      0.75       672
```



```
KNN=neighbors.KNeighborsClassifier(n_neighbors=7,weights='distance'
                                   ,metric='minkowski',p=1)
KNN.fit(X_train, Y_train)
Y_pred = KNN.predict(X_test)
report=classification_report(Y_test, Y_pred)
print(f'Classification Report: \n{report}')
```

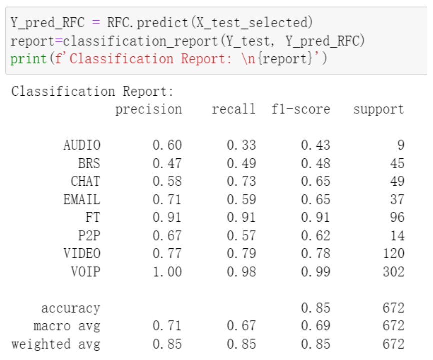
```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.40      0.22      0.29         9
         BRS       0.53      0.40      0.46        45
        CHAT       0.45      0.45      0.45        49
       EMAIL       0.59      0.54      0.56        37
          FT       0.73      0.80      0.76        96
         P2P       0.57      0.29      0.38        14
       VIDEO       0.55      0.65      0.60       120
        VOIP       1.00      0.97      0.98       302

    accuracy                           0.77       672
   macro avg       0.60      0.54      0.56       672
weighted avg       0.77      0.77      0.76       672
```

Figure 4: training on original training datasets

Figure 5:adjusting the parameters

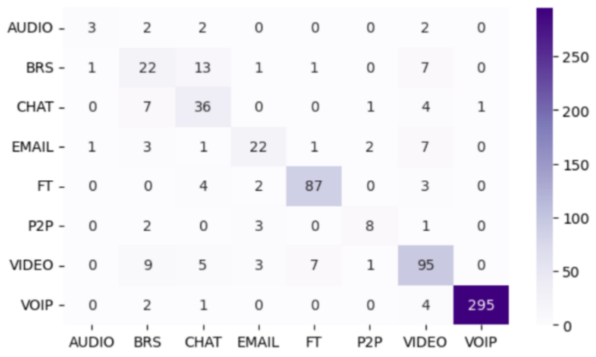
6. **Prediction and Evaluation:**

In the end, I applied the optimal RF and KNN models on the test dataset and used confusion matrices to reveal each model's abilities to detect different categories. The RF model outperformed the KNN model. The former achieved a weighted precision and weighted F1 of 0.85, which is 0.08 points higher than the latter. Both models exhibited the poorest recognition performance for the class with the lowest sample size, Audio. However, for the class with the second-lowest sample size, P2P, the RF model showed relatively consistent recognition capabilities compared to other classes, and its recognition performance was almost always higher than that of the KNN model, as evaluated by the weighted F1 metric. Below are the final performances of the KNN and RF models on the test dataset.



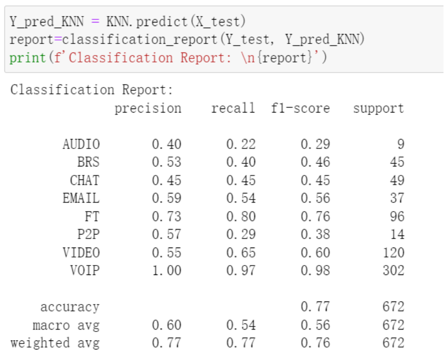
```
Y_pred_RFC = RFC.predict(X_test_selected)
report=classification_report(Y_test, Y_pred_RFC)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.60      0.33      0.43         9
         BRS       0.47      0.49      0.48        45
        CHAT       0.58      0.73      0.65        49
       EMAIL       0.71      0.59      0.65        37
          FT       0.91      0.91      0.91        96
         P2P       0.67      0.57      0.62        14
       VIDEO       0.77      0.79      0.78       120
        VOIP       1.00      0.98      0.99       302

    accuracy                           0.85       672
   macro avg       0.71      0.67      0.69       672
weighted avg       0.85      0.85      0.85       672
```



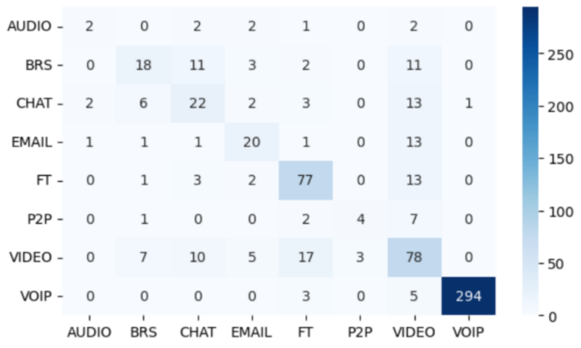
	AUDIO	BRS	CHAT	EMAIL	FT	P2P	VIDEO	VOIP
AUDIO	3	2	2	0	0	0	2	0
BRS	1	22	13	1	1	0	7	0
CHAT	0	7	36	0	0	1	4	1
EMAIL	1	3	1	22	1	2	7	0
FT	0	0	4	2	87	0	3	0
P2P	0	2	0	3	0	8	1	0
VIDEO	0	9	5	3	7	1	95	0
VOIP	0	2	1	0	0	0	4	295

Figure6:The classification Report and confusion matrix of the RF model on the test dataset.



```
Y_pred_KNN = KNN.predict(X_test)
report=classification_report(Y_test, Y_pred_KNN)
print(f'Classification Report: \n{report}')
```

```
Classification Report:
              precision    recall  f1-score   support

       AUDIO       0.40      0.22      0.29         9
         BRS       0.53      0.40      0.46        45
        CHAT       0.45      0.45      0.45        49
       EMAIL       0.59      0.54      0.56        37
          FT       0.73      0.80      0.76        96
         P2P       0.57      0.29      0.38        14
       VIDEO       0.55      0.65      0.60       120
        VOIP       1.00      0.97      0.98       302

    accuracy                           0.77       672
   macro avg       0.60      0.54      0.56       672
weighted avg       0.77      0.77      0.76       672
```



	AUDIO	BRS	CHAT	EMAIL	FT	P2P	VIDEO	VOIP
AUDIO	2	0	2	2	1	0	2	0
BRS	0	18	11	3	2	0	11	0
CHAT	2	6	22	2	3	0	13	1
EMAIL	1	1	1	20	1	0	13	0
FT	0	1	3	2	77	0	13	0
P2P	0	1	0	0	2	4	7	0
VIDEO	0	7	10	5	17	3	78	0
VOIP	0	0	0	0	3	0	5	294

Figure7: The classification Report and confusion matrix of the KNN model on test dataset.