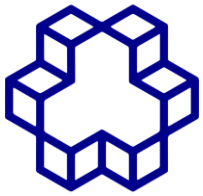
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

امتحان میانترم

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/1_P3-nvhOeYg_IHdjLAWpdWicf3BTGy_N?usp=sharing

دانشجو:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

بهار ۱۴۰۳

Contents

پیشگفتار ۲
سوالات تحلیلی ۳
سوال اول ۳
سوال دوم ۵
سوال سوم ۶
سوالات شبیه سازی ۷

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک های هر سوال آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک هر سوال به ترتیب آورده می‌شود:

لینک نوت بوک:

https://colab.research.google.com/drive/1A28J76cMFHrjVafQF2D4V3SW9hBP9Pd4?usp=sharing

سوالات تحلیلی

سوال اول

الف)نادرست- طبقه بند بیز به دلیل پیروی از توابع توزیع چگالی احتمال optimal است. اما از آنجایی که یافتن توابع چگالی احتمال اکثرا دشوار است. بیشتر از logedticRegression برای طبقه بندی استفاده می شود. فرض محدود کننده طبقه بند بیز عدم وابستگی ویژگی هاست.

ب)درست- به دلیل اینکه بیز بر مبنای احتمال وقوع هر کلاس و فراوانی طبقه بندی را انجام می دهد امکان بیش برازش کمتر است. همچنین به دلیل مقاومت نسبت به ویژگی های نامربوط میتواند نسبت به برازش داده حساسیت کمتری از خود نشان می دهد.

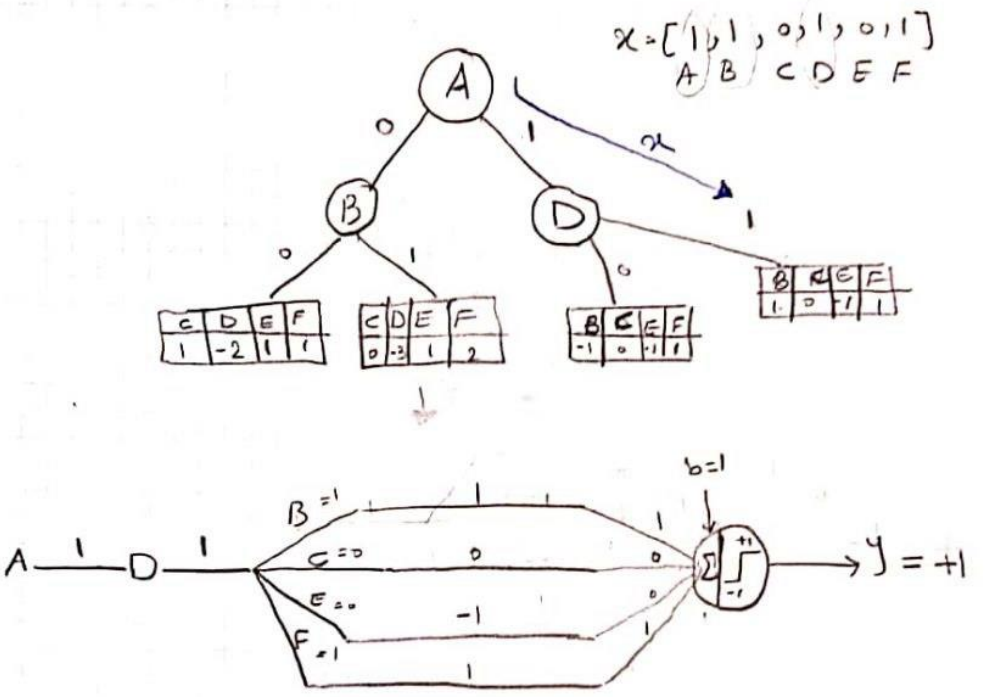
ج)درست- این معیار برا اساس کاهش انتروپی داده پس از Splitکردن داده بر مبنای یک ویژگی عمل می کند. ویژگی با تعداد زیاد حالت سبب پیچیدگی درخت تصمیم می شود. IG برای تک تک ویژگی ها

محاسبه می شود و تقسیم بندی بر مبنای بهترین ویژگی انجام می شود. حال اگر حالات ویژگی زیاد باشد انتروپی زیاد می شود و معیار IG پاسخ مناسبی نخواهد داشت. همچنین اگر مقادیر یکتا داده در یک ویژگی بیشتر باشد ممکن است این ویژگی انتخاب شده و بی معنا باشد و سبب بیش برازش شود.

د)درست- از انجا که هر نورون بیانگر یک شبکه عصبی از ترکیب خطی ورودی هاست، هر لاه پنهان در ورودی اش یک ترکیب خطی از ورودی ها دریافت می کند و چون تابع فعالساز همه لایه های پنهان خطی است، می توان در خروجی هم این ترکیب خطی را مشاهده کرد و بجای چند لایه پنهان از یک شبکه عصبی با تابع فعالسازی خطی استفاده کرد. چرا که ترکیبی از ضرب چند تابع خطی خود خطی است.

سوال دوم

Date: .....



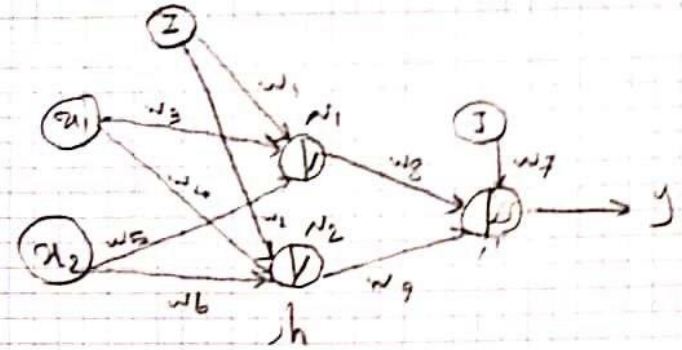
۱- درخت پرسیدن برای نمونه x برچسب +1 را نشان خواهد داد.

۲- تا زمانی که تابع فعالساز پرسیدن را مطابق آنچه روز جلسات معرفی کرد step قرار دهیم درخت بصورت ترکیب خطی از ویژگی ورودی خواهد بود. اگر تابع فعالساز غیرخطی باشد درخت هم غیرخطی می‌شود.

برای مقادیر کوچک عمق درخت تصمیم بهتر است زیرا به دلیل کوچک بودن ابعاد امکان تعلیق کلاس‌ها با درخت تصمیم راحت‌تر و واضح‌تر خواهد بود زیرا بهترین ویژگی انتخاب شده و نمودی به انتهای آن می‌رسد دلیل مشخص می‌شود. اما چنانچه عمق زیاد باشد درخت پرسیدن با وزن‌های مناسب درفرض ما را زودتر به لیبل می‌رساند تا درخت تصمیم. درخت تصمیم با ابعاد بالا ماندگار بهتری خواهد داشت.

سوال سوم

۱-



$$h_1 = (x_1 w_3 + x_2 w_5 + w_1)$$

$$h_2 = (x_1 w_4 + x_2 w_6 + w_2)$$

$$z = h_1 w_8 + h_2 w_9 + w_7$$

$$y = \frac{1}{1 + e^{-z}}$$

مرز تصمیم جایی است که خروجی سیگموید ۰.۵ برسد

if $z = 0$ Then $y = 0.5$

Decision region is: $x_1 w_3 w_8 + x_2 w_5 w_8 + w_1 w_8 + x_1 w_4 w_9 + x_2 w_6 w_9 + w_2 w_9 + w_7 w_9 = 0$

اگر فرض کنیم جمع بردار از وزن حاول بایاس ها صفر شوند:

$x_1 + x_2 + x_1 + x_2 = 0 \rightarrow x_1 + x_2 = 0 \rightarrow$ مرز تصمیم گیری

۲- اگر یک شبکه مدل کلاسیک پرسپترون پیشنهاد می‌شود که در آن یک لایه خروجی با تابع فعال‌ساز سیگموید داریم. لایه پنهان چون خطی است می‌تواند در آگومان سیگموید قرار گیرد.

سوالات شبیه سازی

آرگومان random_state که به‌منظور ایجاد قابلیت تولید مجدد استفاده می‌شود، در طول شبیه سازی ها برابر دو رقم اخر شماره دانشجویی(۶۴) در نظر گرفته می‌شود.

ابتدا دیتاست را بصورت csv در گوگل درایو بارگذاری و با gdownدر کولب وارد می کنیم.دو دیتاست در اختیار داریم. از دیتاست ۹ نوامبر برای اموزش مدل و از دیتاست ۱۷ نوامبر برای ارزیابی عملکرد مدل استفاده می کنیم. بنابراین داده های ۱۷ نوامبر حکم validation دارند.

پس از فراخوانی داده ها در محیط کولب، برای خواندن فایل csv و دریافت اطلاعات(تعداد ردیف و ستون داده، مقادیر پوچ و...) از دستور df.info() از کتابخانه pandas استفاده می‌کنیم:

```
# Display information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86399 entries, 0 to 86398
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       86399 non-null  int64  
 1   699.1   86399 non-null  float64
 2   300.6   86399 non-null  float64
 3   97.8    86399 non-null  float64
 4   296.2   86399 non-null  float64
 5   42.7    86399 non-null  float64
 6   42      86399 non-null  float64
 7   46.1    86399 non-null  float64
 8   129.1   86399 non-null  float64
 9   133.4   86399 non-null  float64
 10  12.2    86399 non-null  float64
 11  26.4    86399 non-null  float64
 12  53      86399 non-null  float64
 13  242     86399 non-null  float64
 14  136.9   86399 non-null  float64
 15  164     86399 non-null  float64
 16  132.1   86399 non-null  float64
dtypes: float64(16), int64(1)
memory usage: 11.2 MB
```

```
df.shape
```

```
(86400, 17)
```

مشاهده می شود که دیتاست شامل ۸۶۴۰۰ داده و ۱۷ ستون است. همچنین تمام ویژگی ها عددی هستند.

برای نمایش مقادیر Null یا NaN از دستور df.isna() استفاده می‌شود که برای هر سلول از دیتافریم مقدار True (دارای مقدار خالی) یا False (دارای مقدار غیر خالی)را بر می‌گرداند:

```
df.isnull().sum()
# Remove rows with any null values
# df.dropna(inplace=True)
```

```
0        0
699.1    0
300.6    0
97.8     0
296.2    0
42.7     0
42       0
46.1     0
129.1    0
133.4    0
12.2     0
26.4     0
53       0
242      0
136.9    0
164      0
132.1    0
dtype: int64
```

دیتاست داده پوچ ندارد.

برای داده های ۱۷ نوامبر که در قسمت ازمون از انها استفاده می کنیم نیز موارد فوق را بررسی می کنیم: مشاهده می شود در اینجا نیز داده پوچ نداریم.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 86399 entries, 0 to 86398
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       86399 non-null  int64
 1   699.1   86399 non-null  float64
 2   300.6   86399 non-null  float64
 3   97.8    86399 non-null  float64
 4   296.2   86399 non-null  float64
 5   42.7    86399 non-null  float64
 6   42      86399 non-null  float64
 7   46.1    86399 non-null  float64
 8   129.1   86399 non-null  float64
 9   133.4   86399 non-null  float64
 10  12.2    86399 non-null  float64
 11  26.4    86399 non-null  float64
 12  53      86399 non-null  float64
 13  242     86399 non-null  float64
 14  136.9   86399 non-null  float64
 15  164     86399 non-null  float64
 16  132.1   86399 non-null  float64
dtypes: float64(16), int64(1)
memory usage: 11.2 MB
```

```
df1.isnull().sum()
# Remove rows with any null values
# df1.dropna(inplace=True)
```

```
0        0
699.1    0
300.6    0
97.8     0
296.2    0
42.7     0
42       0
46.1     0
129.1    0
133.4    0
12.2     0
26.4     0
53       0
242      0
136.9    0
164      0
132.1    0
dtype: int64
```

برای کلاس f16 از لیبل صفر و برای کلاس دیگر از لیبل یک استفاده می کنیم و مطابق با بازه نمونه در جدول داده شده لیبل می زنیم:

```
# Assign labels
label_ranges = [(57274, 57549, 0), (58829, 58929, 1), (58519, 58624 , 1)]
# Initialize a new column for the labels, filling with NaN or a default label if preferred
df['Label'] = pd.NA  # Use NaN or another placeholder if no label should be default

# Apply labels based on the defined ranges
for start, end, label in label_ranges:
    df.loc[start:end, 'Label'] = label

print(df['Label'].value_counts(dropna=False))  # Check how many of each label, including NaN
labeled_df = df.dropna(subset=['Label'])
print(labeled_df.head())
labeled_df.shape
```

```
Label
<NA>    85917
0         276
1         207
Name: count, dtype: int64
           0      1      2     3      4     5     6     7      8      9    10  \
57274  57274  649.6  348.5  96.8  362.6  34.8  33.7  46.7  127.9  131.9  13.2
57275  57275  649.3  359.5  96.8  362.9  34.8  32.9  46.5  127.8  131.9  13.2
57276  57276  650.1  352.6  96.8  363.0  34.6  33.6  46.7  127.8  131.8  13.2
57277  57277  648.6  361.2  96.9  364.5  34.9  33.5  46.9  127.9  131.8  13.2
57278  57278  646.9  357.8  96.9  365.1  35.3  32.8  46.9  127.8  131.9  13.2

         11    12     13     14     15     16 Label
57274  20.5  68.3  271.9  139.9  168.7  133.0     0
57275  20.5  68.8  272.0  139.9  168.8  133.0     0
57276  20.6  65.9  271.8  139.9  168.9  133.0     0
```

همانطور که مشاهده می شود داده هایی که لیبل ندارند را دور ریخته و صرفا داده هایی که لیبل زدیم موجودند. برای داده های ۱۷ نوامبر نیز به همین صورت عمل می کنیم.

تعداد و فراوانی هر کلاس را مشاهده و بر مبنای کمترین تعداد داده در کلاس ها دیتاست را تغییر می دهیم:

همچنین ستون اول دیتاست به دلیل بی تاثیربودن حذف می شود:

```python
import pandas as pd
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
df = df.drop(df.columns[0], axis=1)
# Define the index ranges and corresponding labels
label_ranges = [(57274, 57549, 0), (58829, 58929, 1), (58519, 58624 , 1)]

# Initialize a new column for labels with default values (NaN)
df['Label'] = pd.NA

# Apply labels based on the defined ranges
for start, end, label in label_ranges:
    df.loc[start:end, 'Label'] = label

# Filter only labeled data
labeled_df = df.dropna(subset=['Label'])

# Split data into features (X) and labels (y)
X = labeled_df.drop('Label', axis=1)
y = labeled_df['Label'].astype(int)  # Ensure y is of integer type

# Count the number of samples in each class before balancing
unique, counts = np.unique(y, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))

# Use RandomUnderSampler to balance the classes
rus = RandomUnderSampler(random_state=64)
X_resampled, y_resampled = rus.fit_resample(X, y)

# Count the number of samples in each class after balancing
unique_resampled, counts_resampled = np.unique(y_resampled, return_counts=True)
print("Class counts after balancing:", dict(zip(unique_resampled, counts_resampled)))

# Print the shape of the new balanced dataset
print("\nNew balanced dataset shape:", X_resampled.shape, y_resampled.shape)
```

```
Class counts before balancing: {0: 276, 1: 207}
Class counts after balancing: {0: 207, 1: 207}

New balanced dataset shape: (414, 16) (414,)
```

برای داده های اموزشی از هر کلاس ۲۰۷ داده برداشتیم. برای داده های آزمون نیز با تکرار این مراحل از هر کلاس ۱۰۱ داده بر می داریم.

```python
import pandas as pd
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
df1 = df1.drop(df.columns[0], axis=1)
# Define the index ranges and corresponding labels
label_ranges1 = [(56669, 56769, 0), (54599, 54699, 1)]

# Initialize a new column for labels with default values (NaN)
df1['Label'] = pd.NA

# Apply labels based on the defined ranges
for start, end, label in label_ranges1:
    df1.loc[start:end, 'Label'] = label

# Filter only labeled data
labeled_df1 = df1.dropna(subset=['Label'])

# Split data into features (X) and labels (y)
X1 = labeled_df1.drop('Label', axis=1)
y1 = labeled_df1['Label'].astype(int)  # Ensure y is of integer type

# Count the number of samples in each class before balancing
unique, counts = np.unique(y1, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))

# Use RandomUnderSampler to balance the classes
rus = RandomUnderSampler(random_state=64)
X_resampled1, y_resampled1 = rus.fit_resample(X1, y1)

# Count the number of samples in each class after balancing
unique_resampled, counts_resampled = np.unique(y_resampled1, return_counts=True)
print("Class counts after balancing:", dict(zip(unique_resampled, counts_resampled)))

# Print the shape of the new balanced dataset
print("\nNew balanced dataset shape:", X_resampled1.shape, y_resampled1.shape)
```

```
Class counts before balancing: {0: 101, 1: 101}
Class counts after balancing: {0: 101, 1: 101}

New balanced dataset shape: (202, 16) (202,)
```

حال بااستخراج ویژگی مطابق جدول مینی پروژه اول می پردازیم:

```
mean_values = np.mean(X_resampled, axis=1)
std_dev_values = np.std(X_resampled, axis=1)
rms_value = np.sqrt(np.mean(X_resampled**2, axis=1))
crest_factor = peak_values / rms_value
Abs_Mean_value=np.mean(np.abs(X_resampled), axis=1)
Impulse_Factor=peak_values/Abs_Mean_value
label= y_resampled

# Concatenate mean and standard deviation as features
features = np.column_stack((mean_values, std_dev_values, peak_values, rms_value, crest_factor, Abs_Mean_value, Impulse_Factor))
# Create DataFrame
Data = pd.DataFrame(features, columns=['Mean', 'Standard Deviation','Peak', 'RMS', 'Crest Factor', 'Absolute Mean', 'Impulse Factor'])
print(Data)
```

```
          Mean  Standard Deviation   Peak         RMS  Crest Factor  \
0    168.60625          148.116287  495.0  224.424825      2.205638
1    173.33125          157.042477  608.1  233.893270      2.599904
2    173.30000          156.928527  611.7  233.793611      2.616410
3    166.84375          161.950035  616.4  232.518065      2.650977
4    169.15000          148.461241  502.8  225.061020      2.234061
..         ...                 ...    ...         ...           ...
409  170.40000          167.515522  711.4  238.951062      2.977179
410  169.53125          168.260496  720.6  238.856525      3.016874
411  168.77500          170.421340  739.2  239.850857      3.081915
412  167.42500          172.169569  753.6  240.153058      3.137999
413  166.86250          176.078881  776.8  242.583730      3.202193

     Absolute Mean  Impulse Factor
0         168.60625        2.935834
1         173.33125        3.508311
2         173.30000        3.529717
3         166.84375        3.694475
4         169.15000        2.972510
..              ...             ...
409       170.40000        4.174883
410       169.53125        4.250544
411       168.77500        4.379796
412       167.42500        4.501120
413       166.86250        4.655330

[414 rows x 7 columns]
```

همین کار برای دیتای ۱۷ نوامبر تکرار می شود. اکنون سراغ تقسیم داده ها به اموزش وتست می رویم.

با نسبت ۸۰٪ و ۲۰٪ دیتا را تقسیم می کنیم. ابتدا داده را مخلوط و سپس تقسیم را انجام می دهیم

دیتای ۱۷ نوامبر را به عنوان دیتای اعتبار سنجی نگه می داریم.

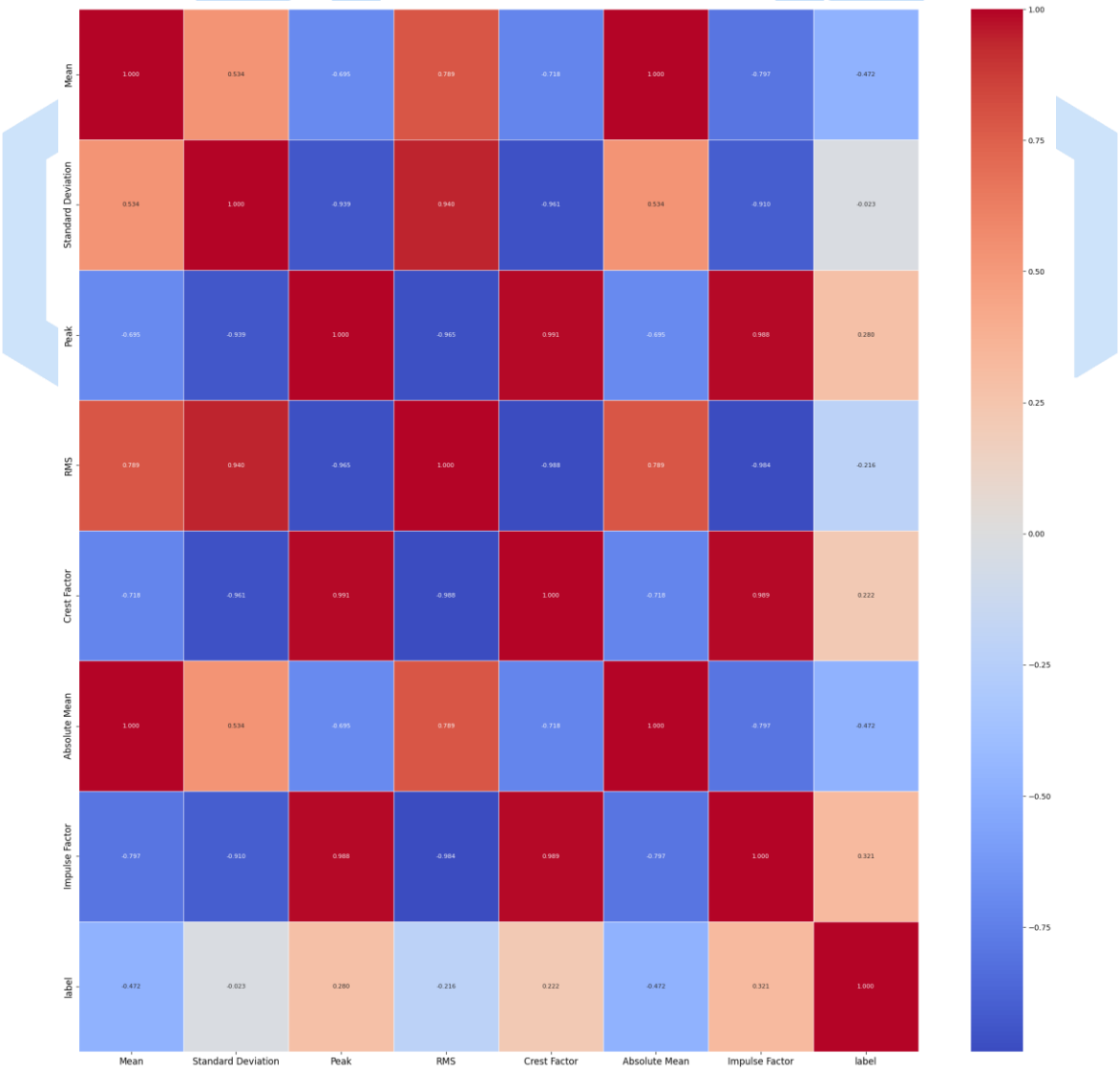
```
datalabel=np.column_stack((mean_values, std_dev_values, peak_values, rms_value, crest_factor, Abs_Mean_value, Impulse_Factor,label))
Data_labeled = pd.DataFrame(datalabel, columns=['Mean', 'Standard Deviation','Peak', 'RMS', 'Crest Factor', 'Absolute Mean', 'Impulse Factor','label'])
np.random.seed(64)
np.random.shuffle(Data_labeled.values)

# Remove the header from the variables
data = Data.values
label = y_resampled
# Split the data into train/validation/test sets
train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.2, random_state=64)
# Print shape of each set
print(f"train_data shape: {train_data.shape}")

print(f"test_data shape: {test_data.shape}")
print(f"train_label shape: {train_label.shape}")

print(f"test_label shape: {test_label.shape}")
```

```
train_data shape: (331, 7)
test_data shape: (83, 7)
train_label shape: (331,)
test_label shape: (83,)
```

با استفاده از minMaxScaler داده ها را بین صفر و یک مقیاس کنیم در این عملیات توجه داریم که از اطلاعات داده های آموزشی استفاده نکنیم؛ چراکه باعث نشت اطلاعات میگردد. نشت اطلاعات به معنای انتقال اطلاعات از داده های تستی به مدل یادگیری است که باعث میشود مدل بهترین عملکرد را بر روی داده های تستی نشان دهد؛ اما در واقع مدل تعمیم پذیری و عمل کرد خوبی ندارد. دستورات مربوط به این قسمت به شرح زیر است:

```
# Scale the data using MinMaxScaler
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_data)
test_data = scaler.transform(test_data)
```

حال ماتریس همبستگی را رسم می کنیم



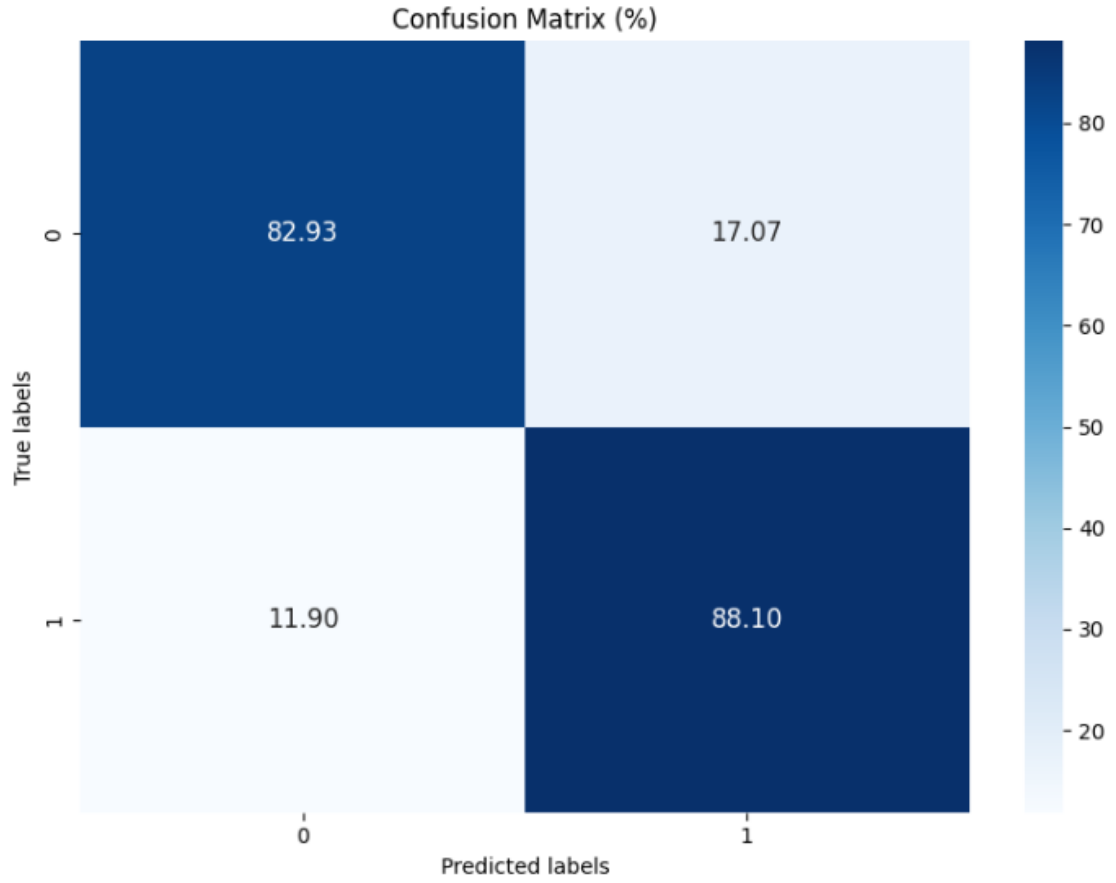
همانطور که پیداست بسیاری از فیچر ها بسیار باهم همبستگی دارند و این مطلوب نیست. راهی که داریم استخراج ویژگی های دیگر است که به دلیل کمبود زمان این راه هم نداریم!

حال با استفاده از sklearnو با mlp سه لایه با۵۰ و ۲۵ نورون در لایه های پنهان مدل را اموزش می دهیم عملکرد مدل با داده های تست و نه اعتبار سنجی به صورت زیر است:

تعداد نورون ها و لایه ها با ازمایش و بررسی دقت مدل در هر مرحله بدست امد در اینجا نتیجه نهایی ذکر می شود:

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming you've trained your model already
model = MLPClassifier(hidden_layer_sizes=(50,25), random_state=12)
model.fit(x_train, y_train)

# Making predictions on the test set
y_pred = model.predict(x_test)

# Calculating confusion matrix
cf_matrix = confusion_matrix(y_test, y_pred)

# Calculating percentages for each cell
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

# Plotting confusion matrix as a heatmap with percentages
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues', annot_kws={"size": 12})

# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix (%)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix_percentage.png', dpi=300)
plt.show()

# Printing classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

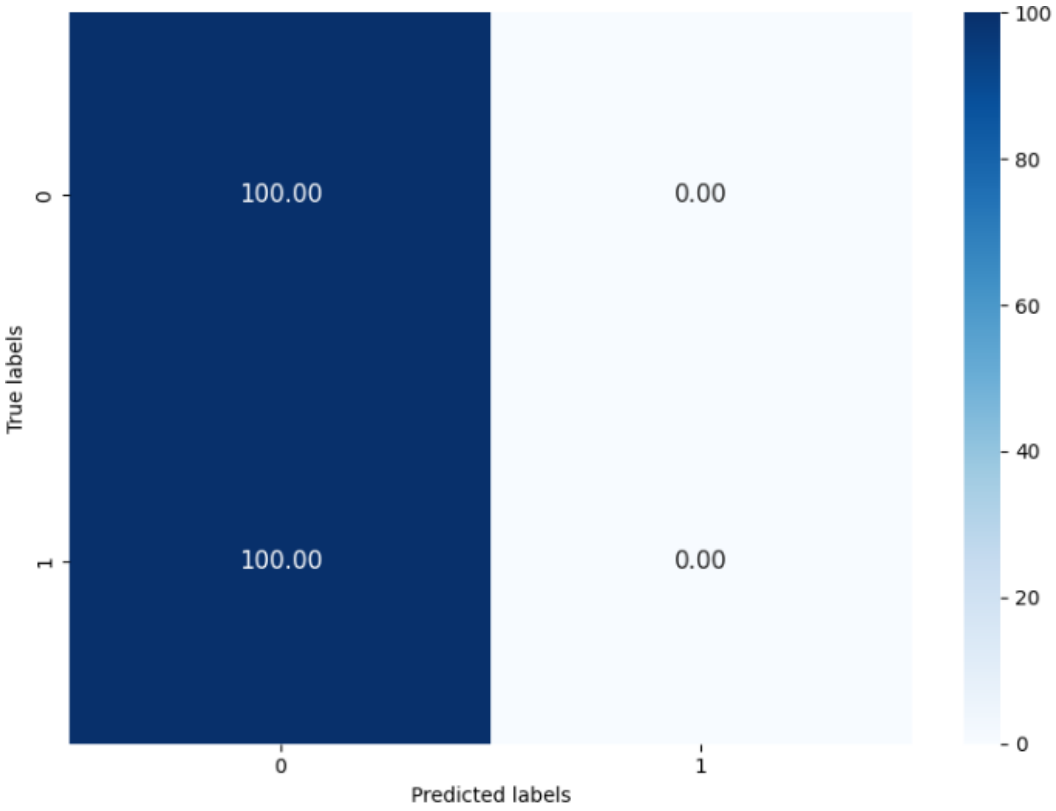
نتایج به صورت زیر است:



Classification Report:

	precision	recall	f1-score	support
0	0.87	0.83	0.85	41
1	0.84	0.88	0.86	42
accuracy			0.86	83
macro avg	0.86	0.86	0.86	83
weighted avg	0.86	0.86	0.86	83

حال مدل را با داده های ۱۷ اکتبر می سنجیم




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

           0       0.50      1.00      0.67       101
           1       0.00      0.00      0.00       101

    accuracy                           0.50       202
   macro avg       0.25      0.50      0.33       202
eighted avg       0.25      0.50      0.33       202
```

نتایج قابل قبول نیست و در صورت زمان باید روی هایپر پارامترها کار می شد...