Data PP

May 3, 2025

1 Importing Necessary Libraries

[5]:
```
import os
import wfdb
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dense, Flatten,␣
↪Dropout, BatchNormalization
from sklearn.model_selection import train_test_split
```

2 Extracting the Patient IDs

[26]:
```
# Patient IDs
pid = sorted(list({i.split(".")[0] for i in os.listdir("../mitdb/")}))
```

[28]:
```
len(pid)
```

[28]: 48

[30]:
```
pid
```

[30]:
```
['100',
 '101',
 '102',
 '103',
 '104',
 '105',
 '106',
 '107',
 '108',
 '109',
 '111',
```

```
'112',
'113',
'114',
'115',
'116',
'117',
'118',
'119',
'121',
'122',
'123',
'124',
'200',
'201',
'202',
'203',
'205',
'207',
'208',
'209',
'210',
'212',
'213',
'214',
'215',
'217',
'219',
'220',
'221',
'222',
'223',
'228',
'230',
'231',
'232',
'233',
'234']
```

3 Data Exploration and Processing

3.1 Annotation

[38]:
```
df = pd.DataFrame()
```

[42]:
```
ann = wfdb.rdann(f"../mitdb/{pid[0]}", "atr")
```

[ ]:
```
ann.symbol
```

[44]:
```
np.unique(ann.symbol, return_counts = True)
```

[44]:
```
(array(['+', 'A', 'N', 'V'], dtype='<U1'),
 array([   1,   33, 2239,    1], dtype=int64))
```

[48]:
```
for i in pid:
    # Annotation Extraction
    ann = wfdb.rdann(f"../mitdb/{i}", "atr")
    sym = ann.symbol
    values, counts = np.unique(sym, return_counts = True)

    data = pd.DataFrame({"Patient ID": [i] * len(counts), "Symbol": values,␣
 ↪"Count": counts})

    df = pd.concat([df, data], axis = 0)
```

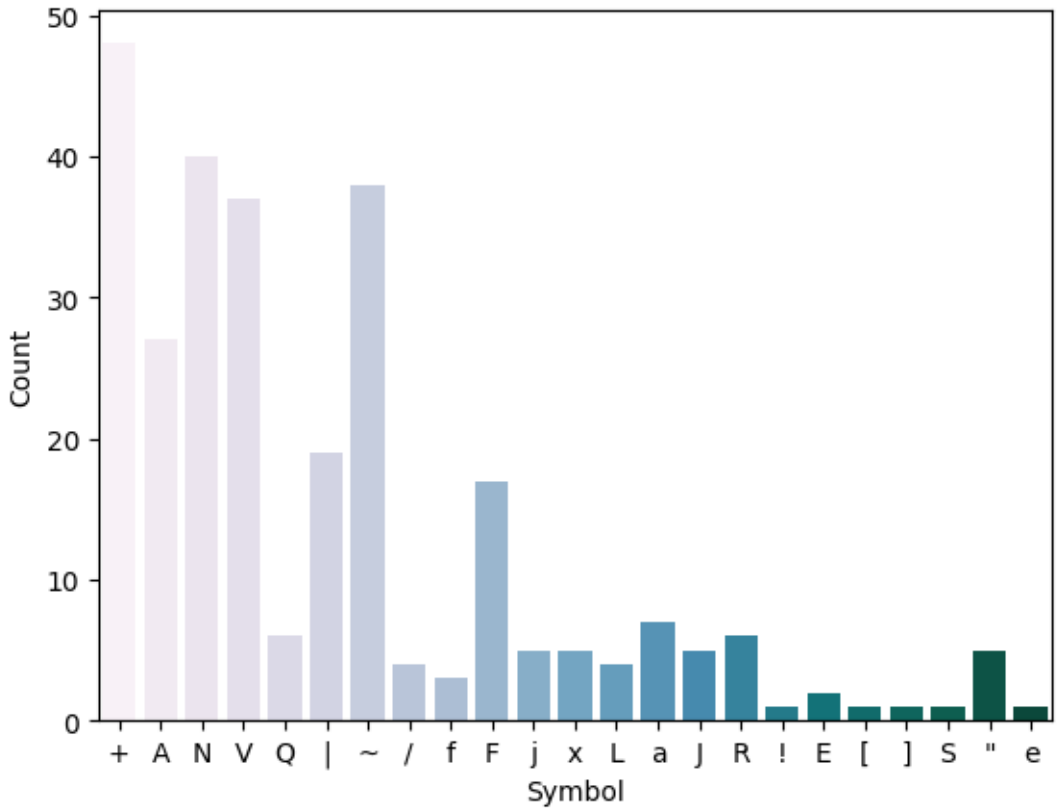
[49]:
```
df
```

[49]:

	Patient ID	Symbol	Count
0	100	+	1
1	100	A	33
2	100	N	2239
3	100	V	1
0	101	+	1
..	…	…	…
0	234	+	3
1	234	J	50
2	234	N	2700
3	234	V	3
4	234	~	8

[283 rows x 3 columns]

[52]:
```
sns.countplot(x = "Symbol", data = df, hue = "Symbol", palette = "PuBuGn",␣
 ↪legend = False)
plt.ylabel("Count")
plt.show()
```



[54]:
```
# Obtained from external sources

# Non Beat Symbols
nonbeat = ['[','!',']','x','(',')','p','t','u','`',␣
 ↪'\'','^','|','~','+','s','T','*','D','=','"','@','Q','?']

# Abnormal Beat Symbols
abnormal = ['L','R','V','/','A','f','F','j','a','E','J','e','S']

# Normal Beat Symbols
normal = ['N']
```

[56]:
```
df["Category"] = -1
df.loc[df.Symbol == "N", "Category"] = 0

for i in abnormal:
    df.loc[df.Symbol == i, "Category"] = 1
```

[58]:
```
df
```

[58]:
```
    Patient ID Symbol  Count  Category
0          100      +      1        -1
1          100      A     33         1
2          100      N   2239         0
3          100      V      1         1
0          101      +      1        -1
..         …    …     …       …
0          234      +      3        -1
1          234      J     50         1
2          234      N   2700         0
3          234      V      3         1
4          234      ~      8        -1

[283 rows x 4 columns]
```

[60]:
```
df.groupby("Category").Count.sum()
```

[60]:
```
Category
-1     3186
 0    75052
 1    34409
Name: Count, dtype: int64
```

[62]:
```
# Removing the non-beat data from the dataframe
df = df[df["Category"] != -1]
```

[64]:
```
df.groupby("Category").Count.sum()
```

[64]:
```
Category
0    75052
1    34409
Name: Count, dtype: int64
```

[66]:
```
# Saving the DataFrame
df.to_csv("../Annotation.csv", index = False)
```

3.2 ECG

The p_signal contains the raw ECG signals as a 2D array.
The r_peaks (ann.sample) obtained from annotation contains the indexes where the heart beats (biggest deflection in the ECG graph) as 1D array

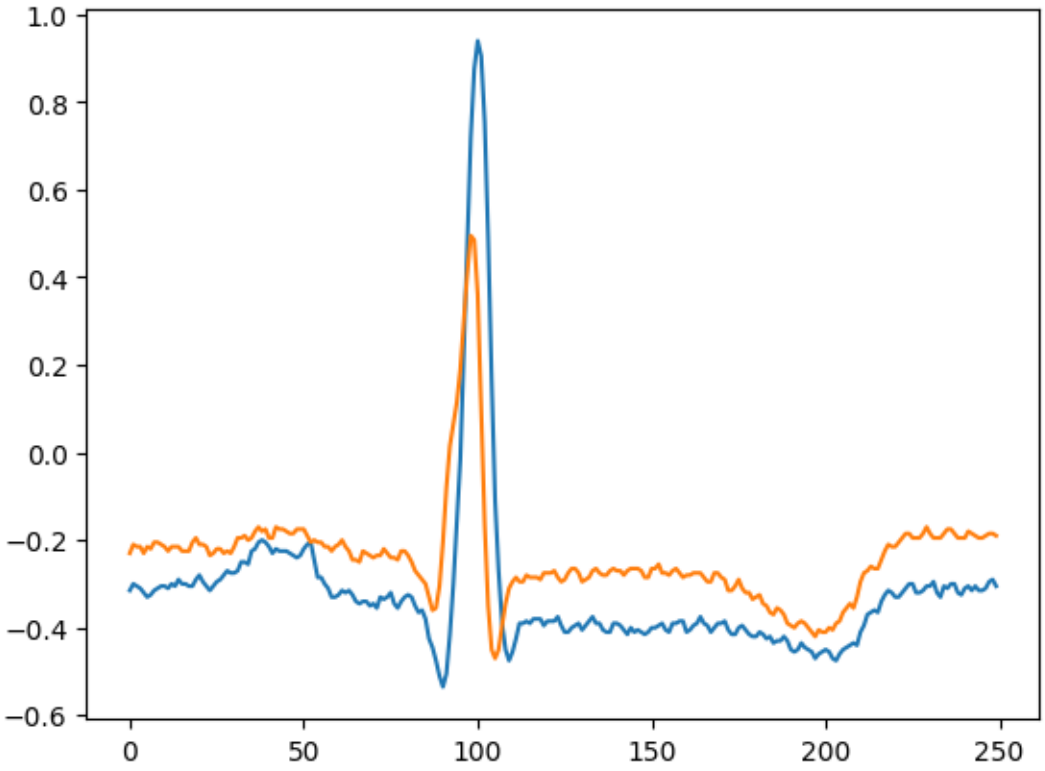
[73]:
```
# Load ECG signal
rec = wfdb.rdrecord('../mitdb/100')
sig = rec.p_signal

# Load annotations
ann = wfdb.rdann('../mitdb/100', 'atr')
```

```
# Get the indices of R-peaks in the signal
# Assuming that the heart beat is 75, then the heart beats around 2250 times
r_peaks_indices = ann.sample
```

[75]:
```
len(sig)
```

[75]: 650000

[79]:
```
plt.plot(sig[270:520])
plt.show()
```



[81]:
```
sig[270:520].dtype
```

[81]: dtype('float64')

[83]:
```
ann.sample.shape, len(ann.symbol)
```

[83]: ((2274,), 2274)

[85]:
```
ann.sample[:5]
```

[85]:
```
array([ 18,  77, 370, 662, 946], dtype=int64)
```

[87]:
```
ann.symbol[:5]
```

[87]:
```
['+', 'N', 'N', 'N', 'N']
```

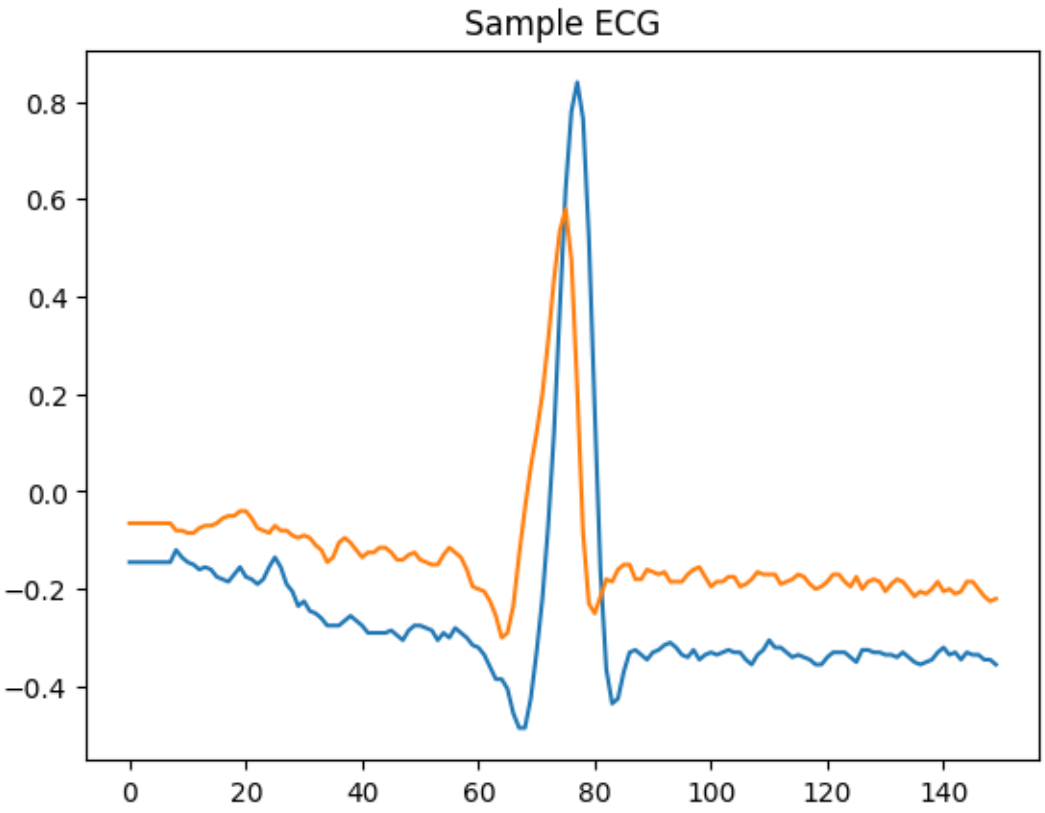
[89]:
```
sig.shape, r_peaks_indices.shape
```

[89]:
```
((650000, 2), (2274,))
```

[91]:
```
r_peaks_indices[:5]
```

[91]:
```
array([ 18,  77, 370, 662, 946], dtype=int64)
```

[93]:
```
plt.plot(sig[:150])
plt.title("Sample ECG")
plt.show()
```



[95]:
```
X = [] # Features
y = [] # Labels
```

[99]:
```
for id in pid:
    rec = wfdb.rdrecord(f"../mitdb/{id}")
    signal = rec.p_signal

    ann = wfdb.rdann(f"../mitdb/{id}", "atr")
    r_peaks_indices = ann.sample

    symbols = ann.symbol

    for idx, r_peak in enumerate(r_peaks_indices):
        start = r_peak - 100
        end = r_peak + 150

        if start >= 0 and end < len(signal):
            X.append(signal[start:end, :].astype(np.float16))
            y.append(symbols[idx])
```

[ ]:
```
X = np.array(X)
```

[ ]:
```
y = np.array(y)
```

[15]:
```
X.dtype
```

[15]:
```
dtype('float16')
```

[18]:
```
encoder = LabelEncoder()

y = encoder.fit_transform(y).astype(np.int8)
```

[20]:
```
y.dtype
```

[20]:
```
dtype('int8')
```

[22]:
```
X.shape, y.shape
```

[22]:
```
((112559, 250, 2), (112559,))
```

[24]:
```
mapping_file = '../Encoded Classes.txt'
label_encoder_mapping = {}
with open(mapping_file, 'r', encoding='utf-8') as file:
    for line in file:
        if '→' in line:
            symbol, code = line.strip().split('→')
            symbol = symbol.strip()
            code = int(code.strip())
            label_encoder_mapping[symbol] = code
```

```
nonbeat_symbols = ['[', '!', ']', 'x', '(', ')', 'p', 't', 'u', '`', '\'', '^',␣
 ↪'|', '~', '+', 's', 'T', '*', 'D', '=', '"', '@', 'Q', '?']

nonbeat_encoded = [label_encoder_mapping[symbol] for symbol in nonbeat_symbols␣
 ↪if symbol in label_encoder_mapping]
print(f"Non-beat encoded labels: {nonbeat_encoded}")

mask = np.isin(y, nonbeat_encoded, invert=True)

X_cleaned = X[mask]
y_cleaned = y[mask]
print(f"Shape after cleaning: X_cleaned = {X_cleaned.shape}, y_cleaned =␣
 ↪{y_cleaned.shape}")
```

```
Non-beat encoded labels: [14, 0, 15, 20, 21, 22, 2, 1, 10]
Shape after cleaning: X_cleaned = (109416, 250, 2), y_cleaned = (109416,)
```

[ ]:
```
unique_classes = np.unique(y_cleaned)
class_counts = np.array([np.sum(y_cleaned == c) for c in unique_classes])
print("Remaining classes and their counts:")
for c, count in zip(unique_classes, class_counts):
    print(f"Class {c}: {count} samples")

max_class_id = np.max(unique_classes)
print(f"New output layer size needed: {max_class_id + 1}")

label_encoder = LabelEncoder()
y_cleaned = label_encoder.fit_transform(y_cleaned)

reverse_mapping = {v: k for k, v in label_encoder_mapping.items()}

symbol_to_new_id = {reverse_mapping[old_id]: new_id
                    for old_id, new_id in zip(label_encoder.classes_,␣
 ↪label_encoder.transform(label_encoder.classes_))
                    if old_id in reverse_mapping}

with open('../Remapped_Symbol_Classes.txt', 'w', encoding='utf-8') as f:
    for symbol, new_id in symbol_to_new_id.items():
        f.write(f"{symbol} → {new_id}\n")

print("Symbol-to-new-ID mapping saved to 'Remapped_Symbol_Classes.txt'.")
```

[78]:
```
np.save("Features.npy", X_cleaned)
np.save("Labels.npy", y_cleaned)
```