Analytics Vidhya

Anomaly Detection in ECG Signals: Identifying Abnormal Heart Patterns Using Deep Learning

COMPUTER VISION DATA EXPLORATION DATA SCIENCE DEEP LEARNING GUIDE INTERMEDIATE

Introduction

Depending on the sector and the particular example, anomaly detection entails spotting out-of-the-ordinary or erratic patterns in data to spot undesirable or odd events.

Anomaly detection can assist in seeing surges in partially completed or fully completed transactions in sectors like e-commerce, marketing, and others, allowing for aligning to shifts in demand or spotting overpriced things.

Anomaly detection is important in healthcare to spot potential health hazards, avoid injury, and improve patient care. It can help monitor patients' chronic disorders metrics, identify fraudulent medical claims, and use medical imaging to diagnose diseases.

In conclusion, Anomaly Detection is a crucial tool in the healthcare industry that helps identify potential health issues and improves patient care.
Learning Objective

This project aims to provide a thorough understanding of ECG signals, their use in anomaly detection, and their use in healthcare. Participants will learn about time series data filtering and the importance of picking the best method for their use case. Additionally, they will have practical experience extracting features from ECG data and discover how these features impact the model's effectiveness. The construction, training, and evaluation of a deep learning model for anomaly detection, such as an LSTM, will all be covered in this project. The importance of feature engineering and subject-matter knowledge will be emphasized as essential elements in creating efficient models for time series data. The reader will learn to use machine learning algorithms to analyze ECG readings and spot irregularities in practice.

.

This article was published as a part of the Data Science Blogathon.

Table of Contents

Project Pipeline

The project involves several steps, which are outlined below. The code for each step will be explored in the next section.

1. Data extraction: This step involves extracting the ECG data from TensorFlow datasets and loading it into the workspace.
2. Exploratory data analysis: This step involves analyzing and visualizing the dataset to check missing values. We will visualize the normal and aberrant ECG signals to identify the trends.
3. Data pre-processing: This step will prepare the data for model training and validation. The data will be split into features and labels, and the ECG signals will be filtered using a variety of algorithms to choose the method based on Mean Squared Error measurement.
4. Splitting train and test set: We will split the data into a training set and a test set for model evaluation purposes.
5. Feature extraction: This is a crucial step in the project. We will extract relevant features from the time series data to train the model, as meaningful feature extraction greatly impacts the model's performance.
6. Building and training deep learning model: We will build the architecture of the LSTM deep learning model and set the necessary parameters, followed by training the model.
7. Model testing: The model will be tested by calculating its predictions on the test data.
8. Model evaluation and visualizations: The model will be assessed based on its predictions on the test data, and the results will be analyzed. Additionally, the training and validation error will be visualized after each epoch of model training.

Prerequisites

The prerequisites to completing this project are:

1. Proficiency in python.
2. Basics of numpy, pandas, sklearn, scipy, Keras, and Plotly.
3. Basics of deep learning, especially the LSTM model.

Dataset Description

The ECG dataset has 4998 examples with 140-time points and 1 target variable.

Each row represents an ECG signal, and the values in the columns are the voltage levels at each time point.

The last column is the target variable with two values: 1 for normal ECG and 0 for aberrant ECG.

All columns have float data types, and the dataset has no missing values.

Now that we have an understanding of the problem statement and the data. Let's go ahead and start coding.

Project Code

The project code is structured in a step-by-step format, making it easy to follow. Each step is divided into sub-sections to simplify the understanding of the code.

Step 1: Import all the Necessary Libraries

First of all, we will import the libraries

```
#Import all the libraries import pandas as pd import numpy as np from sklearn.preprocessing import MinMaxScaler import plotly.graph_objs as go import plotly.express as px from scipy.signal import medfilt, butter, filtfilt import pywt from sklearn.model_selection import train_test_split import scipy.signal from keras.models import Sequential from keras.layers import LSTM, Dense, Reshape from sklearn.metrics import confusion_matrix, accuracy_score, classification_report from sklearn.metrics import roc_auc_score
```

Step 2: Data Extraction

This step involves acquiring the dataset and loading it into your python environment.

```
#import the dataset from tensorflow datasets df = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv', header=None)
```

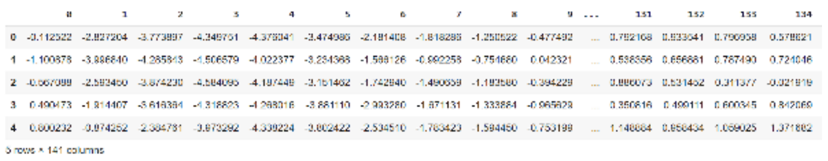
Step 3: Exploratory Data Analysis

Now, let's examine the data by checking its shape, data types, and the presence of any missing values. Additionally, we will visualize the aberrant and normal ECG signals to search for any noticeable patterns.

1. Display the first 5 rows of data

```
df.head()
```

Output:



	0	1	2	3	4	5	6	7	8	9	...	131	132	133	134
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522	-0.477492	...	0.792168	0.933541	0.796958	0.578621
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680	0.042321	...	0.538356	0.656881	0.787490	0.724046
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580	-0.394229	...	0.886073	0.531452	0.311377	-0.021919
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884	-0.965629	...	0.350816	0.499111	0.600345	0.842069
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450	-0.753199	...	1.148884	0.958434	1.059025	1.371682

5 rows × 141 columns

2. The shape of the dataset

```
df.shape
```

Output:

```
(4998, 141)
```

3. Dataframe information

```
df.info()
```

Output:

```
RangeIndex: 4998 entries, 0 to 4997 Columns: 141 entries, 0 to 140 dtypes: float64(141) memory usage: 5.4 MB
```

4. Display all the column names

```
df.columns
```

Output:

```
Int64Index([  0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  ...  131,  132,  133,  134,  135,  136,  137,  138,  139,  140],
dtype='int64', length=141)
```

5. Labels of type of ECG signals

Display the unique values of the target variable, the last column with an index “140.”

df[140].unique()
output:

```
array([1., 0.])
```

Display the number of examples for each label: aberrant and normal ECG.

```
df[140].value_counts() # 1 is normal & 0 is abnormal ecg
```

Output:

```
1.0 2919 0.0 2079 Name: 140, dtype: int64
```

6. Display the datatypes for each column

```
df.dtypes
```

Output:

```
0 float64 1 float64 2 float64 3 float64 4 float64 ... 136 float64 137 float64 138 float64 139 float64 140
float64 Length: 141, dtype: object
```

7. Check if the data has any missing values

No. of missing values per column

```
#looking at missing values for each column df.isna().sum()
```

Output:

```
0 0 1 0 2 0 3 0 4 0 .. 136 0 137 0 138 0 139 0 140 0 Length: 141, dtype: int64
```

Missing values for the entire dataset

```
#missing values for entire dataset df.isna().sum().sum()
```

Output:

```
0
```

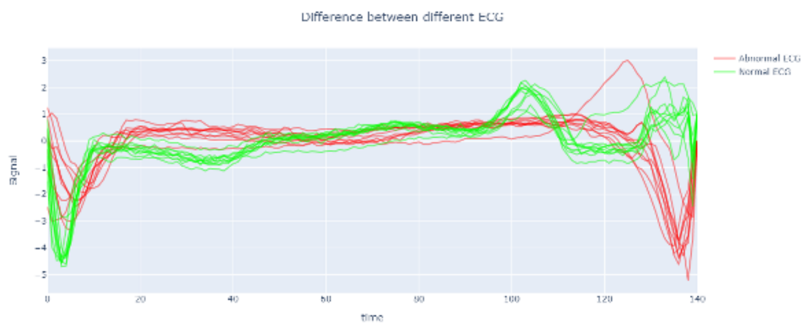
8. Plotting Normal and Aberrant ECG signals

Let’s visualize both the aberrant and normal ECG signals. We will divide the dataset into normal and aberrant ECG and then plot 10 examples using the Plotly library.

```
#plot graphs of normal and abnormal ECG to visualise the trends abnormal = df[df.loc[:,140] ==0][:10] normal
= df[df.loc[:,140] ==1][:10] # Create the figure fig = go.Figure() #create a list to display only a single
legend leg = [False] * abnormal.shape[0] leg[0] = True
```

```
# Plot training and validation error for i in range(abnormal.shape[0]): fig.add_trace(go.Scatter(
x=np.arange(abnormal.shape[1]),y=abnormal.iloc[i,:],name='Abnormal ECG', mode='lines',
marker_color='rgba(255, 0, 0, 0.9)', showlegend= leg[i])) for j in range(normal.shape[0]):
fig.add_trace(go.Scatter( x=np.arange(normal.shape[1]),y=normal.iloc[j,:],name='Normal ECG', mode='lines',
marker_color='rgba(0, 255, 0, 1)', showlegend= leg[j])) fig.update_layout(xaxis_title="time",
yaxis_title="Signal", title= {'text': 'Difference between different ECG', 'xanchor': 'center', 'yanchor':
'top', 'x':0.5} , bargap=0,) fig.update_traces(opacity=0.5) fig.show()
```

Output graph:



Step 4: Data Preprocessing

To prepare the data for our machine-learning model, we need to preprocess it. This includes splitting the data into labels and features, normalizing the dataset, and removing noise from the ECG signal using different filtering algorithms. We will evaluate the results using visualization and error calculation to determine the best filtering algorithm.

1. Split the dataset into features and labels

```
# split the data into labels and features ecg_data = df.iloc[:,:-1] labels = df.iloc[:,-1]
```

2. Normalize the dataset

```
# Normalize the data between -1 and 1 scaler = MinMaxScaler(feature_range=(-1, 1)) ecg_data =
scaler.fit_transform(ecg_data)
```

3. Filter the dataset

Now, let’s filter the data to eliminate noise. We will use three algorithms: Median filtering, Low-pass filtering, and Wavelet filtering.

```
#filtering the raw signals # Median filtering ecg_medfilt = medfilt(ecg_data, kernel_size=3)
```

```
# Low-pass filtering lowcut = 0.05 highcut = 20.0 nyquist = 0.5 * 360.0 low = lowcut / nyquist high = highcut
/ nyquist b, a = butter(4, [low, high], btype='band') ecg_lowpass = filtfilt(b, a, ecg_data)
```

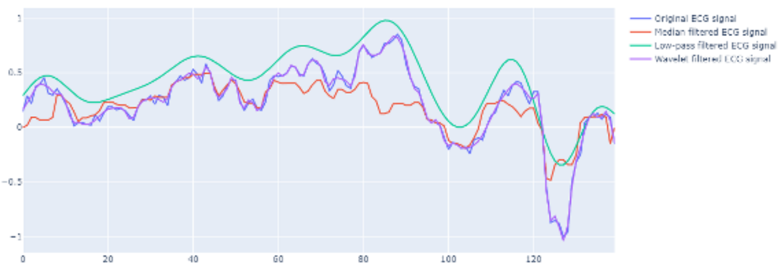
```
# Wavelet filtering coeffs = pywt.wavedec(ecg_data, 'db4', level=1) threshold = np.std(coeffs[-1]) *
np.sqrt(2*np.log(len(ecg_data))) coeffs[1:] = (pywt.threshold(i, value=threshold, mode='soft') for i in
coeffs[1:]) ecg_wavelet = pywt.waverec(coeffs, 'db4')
```

4. Plot the graphs of unfiltered and filtered signals

For visual comparison, let’s plot the unfiltered and filtered signals on a single graph. This visualization will aid in selecting the most effective filtering algorithm by eyeballing the graph.

```
# Plot original ECG signal fig = go.Figure() fig.add_trace(go.Scatter(x=np.arange(ecg_data.shape[0]),
y=ecg_data[30], mode='lines', name='Original ECG signal')) # Plot filtered ECG signals
fig.add_trace(go.Scatter(x=np.arange(ecg_medfilt.shape[0]), y=ecg_medfilt[30], mode='lines', name='Median
filtered ECG signal')) fig.add_trace(go.Scatter(x=np.arange(ecg_lowpass.shape[0]), y=ecg_lowpass[30],
mode='lines', name='Low-pass filtered ECG signal'))
fig.add_trace(go.Scatter(x=np.arange(ecg_wavelet.shape[0]), y=ecg_wavelet[30], mode='lines', name='Wavelet
filtered ECG signal')) fig.show()
```

Output graph:



5. Choosing the best filtering technique

We will evaluate the filtering methods by calculating each algorithm’s mean squared error (MSE). The algorithm with the lowest MSE will be selected.

To calculate MSE, both signals’ dataframe must have an equal number of rows.

If the dataset is reduced after filtering, this step involves padding the data with zeros.

```
#pad the signal with zeroes
```

```
def pad_data(original_data,filtered_data): # Calculate the difference in length between the original data and filtered data diff = original_data.shape[1] - filtered_data.shape[1] # pad the shorter array with zeroes if diff > 0: # Create an array of zeros with the same shape as the original data padding = np.zeros((filtered_data.shape[0], original_data.shape[1])) # Concatenate the filtered data with the padding padded_data = np.concatenate((filtered_data, padding)) elif diff < 0: padded_data = filtered_data[:,:-abs(diff)] elif diff == 0: padded_data = filtered_data return padded_data
```

Then, we will compute the mean squared error for all the techniques.

```
def mse(original_data, filtered_data):
```

```
filter_data = pad_data(original_data,filtered_data) return np.mean((original_data - filter_data) ** 2) # Calculate MSE mse_value_m = mse(ecg_data, ecg_medfilt) mse_value_l = mse(ecg_data, ecg_lowpass) mse_value_w = mse(ecg_data, ecg_wavelet) print("MSE value of Median Filtering:", mse_value_m) print("MSE value of Low-pass Filtering:", mse_value_l) print("MSE value of Wavelet Filtering:", mse_value_w)
```

Output:

```
MSE value of Median Filtering: 0.017260298402611125 MSE value of Low-pass Filtering: 0.36750805414756493 MSE value of Wavelet Filtering: 0.0010818752598698714
```

Based on the MSE value displayed above, wavelet filtering is chosen.

Step 5: Splitting Data into Train & Test Set

The dataset is divided into 80% for training and 20% for testing and validation purposes.

```
# Splitting the data into train and test sets X_train, X_test, y_train, y_test = train_test_split(ecg_wavelet, labels, test_size=0.2, random_state=42)
```

Step 6: Feature Extraction

What are the features?

Features are the attributes that describe a particular data and can be used to make predictions or classify unusual cases. It is important to extract relevant features because feeding the model irrelevant or redundant information could negatively impact its performance and accuracy.

Why extract features?

After preprocessing the data, we must extract features from the ECG signals for our deep learning model. Directly training the model on all the time series data could result in overfitting and be computationally expensive. We can reduce the amount of data by extracting relevant features and improving the model’s generalization ability to new, unseen data.
Choosing relevant features

In this context, the following features are being used:

1. T amplitude: The height of the T wave on the electrocardiogram (ECG) graph, which symbolizes the heart’s return to rest following a contraction
2. R amplitude: The height of the R wave on the ECG graph represents the heart muscle’s initial contraction.
3. RR interval: The time between two consecutive R waves on the ECG graph, representing the time between heartbeats.
4. QRS duration: The time it takes for the QRS complex, representing the electrical signal traveling through the ventricles.

Process of feature extraction:

To extract these features, we first calculate the R & T peaks, the R amplitude, the RR interval, etc., using the scipy.signals library. Then, we calculate each feature’s mean, median, sum, and other statistical metrics to capture its characteristics. All of these features are then stored in an array.

5. Feature extraction of the train set

```
# Initializing an empty list to store the features features = [] # Extracting features for each sample for i
in range(X_train.shape[0]): #Finding the R-peaks r_peaks = scipy.signal.find_peaks(X_train[i])[0] #Initialize
lists to hold R-peak and T-peak amplitudes r_amplitudes = [] t_amplitudes = [] # Iterate through R-peak
locations to find corresponding T-peak amplitudes for r_peak in r_peaks: # Find the index of the T-peak
(minimum value) in the interval from R-peak to R-peak + 200 samples t_peak = np.argmin(X_train[i]
[r_peak:r_peak+200]) + r_peak #Append the R-peak amplitude and T-peak amplitude to the lists
r_amplitudes.append(X_train[i][r_peak]) t_amplitudes.append(X_train[i][t_peak]) # extracting singular value
metrics from the r_amplitudes std_r_amp = np.std(r_amplitudes) mean_r_amp = np.mean(r_amplitudes)
median_r_amp = np.median(r_amplitudes) sum_r_amp = np.sum(r_amplitudes) # extracting singular value metrics
from the t_amplitudes std_t_amp = np.std(t_amplitudes) mean_t_amp = np.mean(t_amplitudes) median_t_amp =
np.median(t_amplitudes) sum_t_amp = np.sum(t_amplitudes) # Find the time between consecutive R-peaks
rr_intervals = np.diff(r_peaks) # Calculate the time duration of the data collection time_duration =
(len(X_train[i]) - 1) / 1000 # assuming data is in ms # Calculate the sampling rate sampling_rate =
len(X_train[i]) / time_duration # Calculate heart rate duration = len(X_train[i]) / sampling_rate heart_rate
= (len(r_peaks) / duration) * 60 # QRS duration qrs_duration = [] for j in range(len(r_peaks)):
qrs_duration.append(r_peaks[j]-r_peaks[j-1]) # extracting singular value metrics from the qrs_durations
std_qrs = np.std(qrs_duration) mean_qrs = np.mean(qrs_duration) median_qrs = np.median(qrs_duration) sum_qrs
= np.sum(qrs_duration) # Extracting the singular value metrics from the RR-interval std_rr =
np.std(rr_intervals) mean_rr = np.mean(rr_intervals) median_rr = np.median(rr_intervals) sum_rr =
np.sum(rr_intervals) # Extracting the overall standard deviation std = np.std(X_train[i]) # Extracting the
overall mean mean = np.mean(X_train[i]) # Appending the features to the list features.append([mean, std,
std_qrs, mean_qrs,median_qrs, sum_qrs, std_r_amp, mean_r_amp, median_r_amp, sum_r_amp, std_t_amp, mean_t_amp,
median_t_amp, sum_t_amp, sum_rr, std_rr, mean_rr,median_rr, heart_rate]) # Converting the list to a numpy
array features = np.array(features)
```

We have now extracted 19 features from the dataset.

The shape of this training set after feature extraction is:

```
(3998, 19)
```

6. Feature extraction of the test set

Similarly, we will extract the features of the test set.

```
# Initializing an empty list to store the features X_test_fe = [] # Extracting features for each sample for i
in range(X_test.shape[0]): # Finding the R-peaks r_peaks = scipy.signal.find_peaks(X_test[i])[0] # Initialize
```

```
lists to hold R-peak and T-peak amplitudes r_amplitudes = [] t_amplitudes = [] # Iterate through R-peak locations to find corresponding T-peak amplitudes for r_peak in r_peaks: # Find the index of the T-peak (minimum value) in the interval from R-peak to R-peak + 200 samples t_peak = np.argmin(X_test[i][r_peak:r_peak+200]) + r_peak # Append the R-peak amplitude and T-peak amplitude to the lists r_amplitudes.append(X_test[i][r_peak]) t_amplitudes.append(X_test[i][t_peak]) #extracting singular value metrics from the r_amplitudes std_r_amp = np.std(r_amplitudes) mean_r_amp = np.mean(r_amplitudes) median_r_amp = np.median(r_amplitudes) sum_r_amp = np.sum(r_amplitudes) #extracting singular value metrics from the t_amplitudes std_t_amp = np.std(t_amplitudes) mean_t_amp = np.mean(t_amplitudes) median_t_amp = np.median(t_amplitudes) sum_t_amp = np.sum(t_amplitudes) # Find the time between consecutive R-peaks rr_intervals = np.diff(r_peaks) # Calculate the time duration of the data collection time_duration = (len(X_test[i]) - 1) / 1000 # assuming data is in ms # Calculate the sampling rate sampling_rate = len(X_test[i]) / time_duration # Calculate heart rate duration = len(X_test[i]) / sampling_rate heart_rate = (len(r_peaks) / duration) * 60 # QRS duration qrs_duration = [] for j in range(len(r_peaks)): qrs_duration.append(r_peaks[j]-r_peaks[j-1]) #extracting singular value metrics from the qrs_duartions std_qrs = np.std(qrs_duration) mean_qrs = np.mean(qrs_duration) median_qrs = np.median(qrs_duration) sum_qrs = np.sum(qrs_duration) # Extracting the standard deviation of the RR-interval std_rr = np.std(rr_intervals) mean_rr = np.mean(rr_intervals) median_rr = np.median(rr_intervals) sum_rr = np.sum(rr_intervals) # Extracting the standard deviation of the RR-interval std = np.std(X_test[i]) # Extracting the mean of the RR-interval mean = np.mean(X_test[i]) # Appending the features to the list X_test_fe.append([mean, std, std_qrs, mean_qrs,median_qrs, sum_qrs, std_r_amp, mean_r_amp, median_r_amp, sum_r_amp, std_t_amp, mean_t_amp, median_t_amp, sum_t_amp, sum_rr, std_rr, mean_rr,median_rr,heart_rate]) # Converting the list to a numpy array X_test_fe = np.array(X_test_fe)
```

The shape of the test set after feature extraction is as follows:

```
(1000, 19)
```

Step 7: Model Building and Training

We will now build a Recurrent Neural Network LSTM model. First, we will reshape the data to make it compatible with the model. Then, we will create an LSTM model with only 2 layers. Then, we will train it on the features extracted from the data. Finally, we will make the predictions on the validation/test set.

```
# Define the number of features in the train dataframe num_features = features.shape[1] # Reshape the features data to be in the right shape for LSTM input features = np.asarray(features).astype('float32') features = features.reshape(features.shape[0], features.shape[1], 1) X_test_fe = X_test_fe.reshape(X_test_fe.shape[0], X_test_fe.shape[1], 1) # Define the model architecture model = Sequential() model.add(LSTM(64, input_shape=(features.shape[1], 1))) model.add(Dense(1, activation='sigmoid')) # Compile the model model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']) # Train the model history = model.fit(features, y_train, validation_data=(X_test_fe, y_test), epochs=50, batch_size=32) # Make predictions on the validation set y_pred = model.predict(X_test_fe) # Convert the predicted values to binary labels y_pred = [1 if p>0.5 else 0 for p in y_pred] X_test_fe = np.asarray(X_test_fe).astype('float32')
```

Step 8: Model Evaluation

Now, we will evaluate the model’s performance using metrics, e.g., accuracy, AUC score precision, etc. Furthermore, we will visualize the confusion matrix in Plotly.

1. Calculating all the metrics

```
# calculate the accuracy acc = accuracy_score(y_test, y_pred) #calculate the AUC score auc =
round(roc_auc_score(y_test, y_pred),2) #classification report provides all metrics e.g. precision, recall,
etc. all_met = classification_report(y_test, y_pred)
```

2. Displaying all the metrics

```
# Print the accuracy print("Accuracy: ", acc*100, "%") print(" n") print("AUC:", auc) print(" n")
print("Classification Report: n", all_met) print(" n")
```

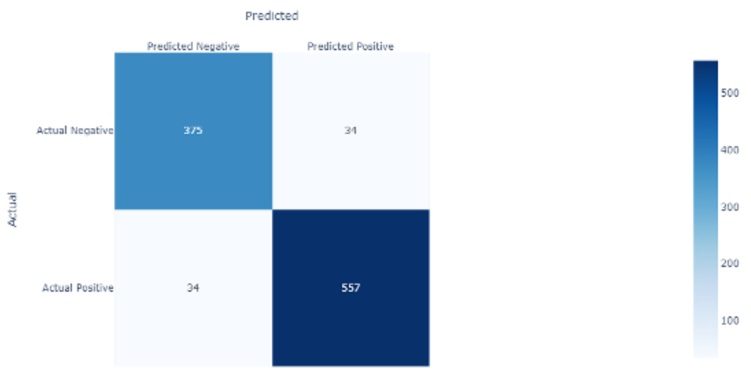
Output:

```
Accuracy:  93.2 %

AUC: 0.93

Classification Report:
               precision    recall  f1-score   support

         0.0       0.92      0.92      0.92       409
         1.0       0.94      0.94      0.94       591

    accuracy                           0.93      1000
   macro avg       0.93      0.93      0.93      1000
weighted avg       0.93      0.93      0.93      1000
```

3. Calculating and displaying the confusion matrix

```
# Calculate the confusion matrix conf_mat = confusion_matrix(y_test, y_pred) conf_mat_df =
pd.DataFrame(conf_mat, columns=['Predicted Negative', 'Predicted Positive'], index=['Actual Negative',
'Actual Positive']) fig = px.imshow(conf_mat_df, text_auto= True, color_continuous_scale='Blues')
fig.update_xaxes(side='top', title_text='Predicted') fig.update_yaxes(title_text='Actual') fig.show()
```

Output:

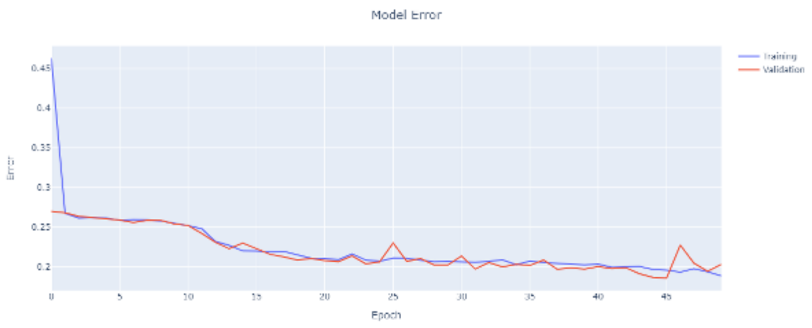


Step 9: Plotting the Training and Validation Error

Finally, let's visualize the training error and validation error after every epoch of the model training.

```
# Plot training and validation error fig = go.Figure() fig.add_trace(go.Scatter( y=history.history['loss'],
mode='lines', name='Training')) fig.add_trace(go.Scatter( y=history.history['val_loss'], mode='lines',
name='Validation')) fig.update_layout(xaxis_title="Epoch", yaxis_title="Error", title= {'text': 'Model
Error', 'xanchor': 'center', 'yanchor': 'top', 'x':0.5} , bargap=0) fig.show()
```

Output:



Results

The model achieved a recall value of 0.92 and an AUC score of 0.93, exhibiting its effectiveness with a simple deep-learning architecture. In the healthcare sector, recall is a crucial metric, especially in disease screening, where false negatives can lead to serious consequences, including missed early detection and treatment opportunities. The good recall score in this project highlights the potential for this model to impact disease screening efforts positively.

Future Work

The data has an imbalance with around 2919 examples of normal cases and 2079 examples of aberrant cases. It would be better to balance the dataset first to improve the Recall and AUC scores. Also, in the future, a more intricate deep learning model with more layers could be applied to improve the results.

Conclusion

In this project, we have gone through a comprehensive time series data analysis and feature extraction process. We filtered the data using different techniques and selected the best one based on our specific use case. Further, we extracted features from the ECG signals and used these features to train a two-layer LSTM model. We assessed the model's performance using metrics e.g., accuracy, AUC score, precision, etc., and visualized the results using a confusion matrix and training and validation error plots.

These are the project's main learning objectives:

1. Understanding the importance of filtering time series data and choosing the best filtering algorithm.
2. Hands-on experience with feature extraction from time series data, especially ECG signals, and the impact of these features on the model's performance.
3. Understanding the process of building, training, and evaluating a deep learning model, specifically an LSTM model, for anomaly detection.

4. Overall, this project highlights the importance of subject matter knowledge and feature engineering in building effective models for time series data.

You can access the full jupyter notebook here on this link.

Thank you for reading this article! If you enjoyed the content on anomaly detection using deep learning and would like to stay updated on similar topics, please follow me on LinkedIn.

Article Url - https://www.analyticsvidhya.com/blog/2023/02/anomaly-detection-in-ecg-signals-identifying-abnormal-heart-patterns-using-deep-learning/

Syed Huma Shah