ECG Anomaly Classification

In [1]:
```
# Put these at the top of every notebook, to get automatic reloading and inline plotting
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
# Import packages
import glob
import random
from collections import OrderedDict
from biosppy.signals import ecg
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.interpolate as interp
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from joblib import Parallel, delayed


import tensorflow as tf
from keras.models import Model, Sequential
from keras.layers import Input, Dense, LSTM, Dropout
from keras.preprocessing import sequence
from keras.backend.tensorflow_backend import set_session
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils import shuffle

seed = 9441
random.seed(seed)
np.random.seed(seed)

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.visible_device_list = "0"
set_session(tf.Session(config=config))
```

```
Using TensorFlow backend.
```

Working on the preprocessed dataset

There are 4 types of anomalies: Control, High P Amplitude, SA and ST.

All data under 1000 Hz sampling frequency.

In [3]:
```
con_dir = 'cleaned_data/control'
highp_dir = 'cleaned_data/high p wave'
sa_dir = 'cleaned_data/sinoatrial arrest'
st_dir = 'PROCESSED ECG database/processedST'

sampling_freq = 1000
```

In [4]:
```
def chunks(l, n):
    """Return successive n-sized chunks from l."""
    chunked_list = []
    for i in range(0, len(l), n):
        sublist = l[i:i + n]
        if len(sublist) == n:
            chunked_list.append(l[i:i + n])
    return chunked_list
```

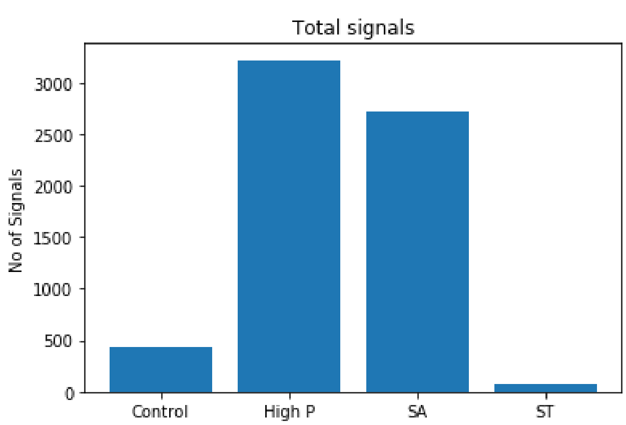
In [5]:
```
def get_len_sublists(alist, level=1):
        if level == 1:
            return [len(sublist) for sublist in alist]
        elif level == 2:
            lens = []
            for inlist in alist:
                lens.extend(get_len_sublists(inlist))
            return lens
```

In [6]:
```
def select_random(alist):
    aux = list(alist)
    np.random.shuffle(aux)
    return aux[0]
```

Original signals whose lengths are greater than 10k data points (10 seconds of data recorded) are split into multiple samples of the same length of 10k if possible.

In [7]:
```
def read_signals(adir, all_files=False):
    # If a signal sequence is too long, split it into sublists
    standard_size = 10000 # 10 seconds
    if all_files:
        files = sorted(glob.glob(adir + '/*.csv'))
    else:
        files = sorted(glob.glob(adir + '/_*.csv'))

    signals = [pd.read_csv(fi, header=None)[0].values for fi in files]
    standard_signals = [signal for signal in signals if len(signal) < standard_size]
    oversized_signals = [signal for signal in signals if len(signal) >= standard_size]
    for signal in oversized_signals:
        standard_signals.extend(chunks(signal, standard_size))
    return standard_signals

con_signals = read_signals(con_dir, all_files=True)
highp_signals = read_signals(highp_dir, all_files=True)
sa_signals = read_signals(sa_dir, all_files=True)
st_signals = read_signals(st_dir)
```

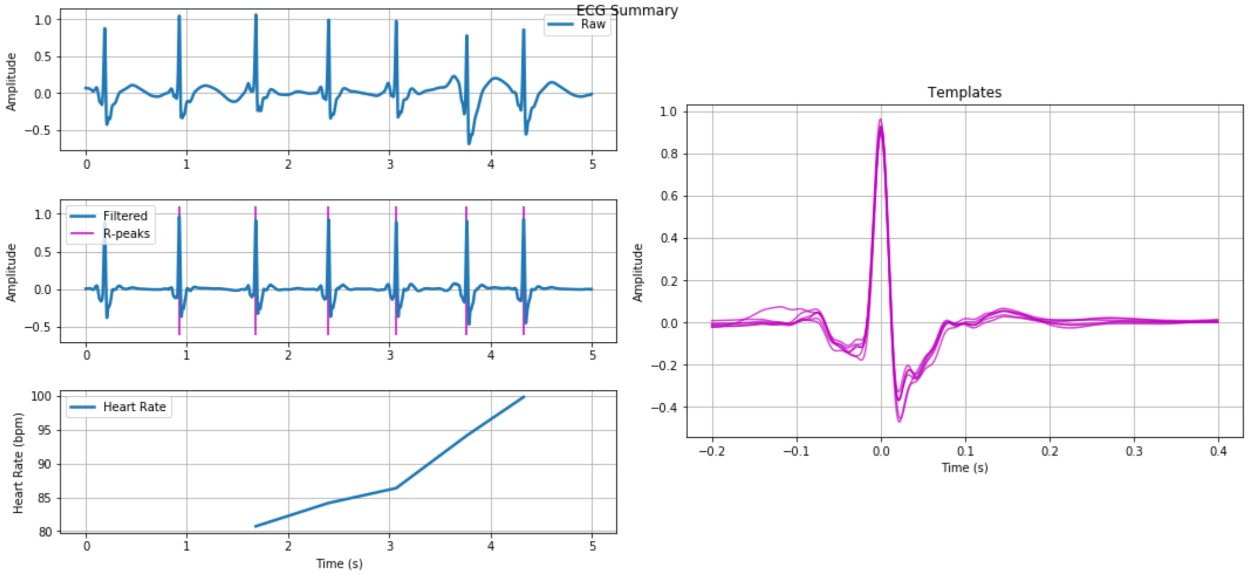
In [8]:
```
anomalies = ['Control', 'High P', 'SA', 'ST']
y_pos = np.arange(len(anomalies))
lengths = [len(con_signals), len(highp_signals), len(sa_signals), len(st_signals)]

plt.bar(y_pos, lengths, align='center')
plt.xticks(y_pos, anomalies)
plt.ylabel('No of Signals')
plt.title('Total signals')
plt.show()
```



In [9]:
```
plt.rcParams['figure.figsize'] = [15, 7]
```

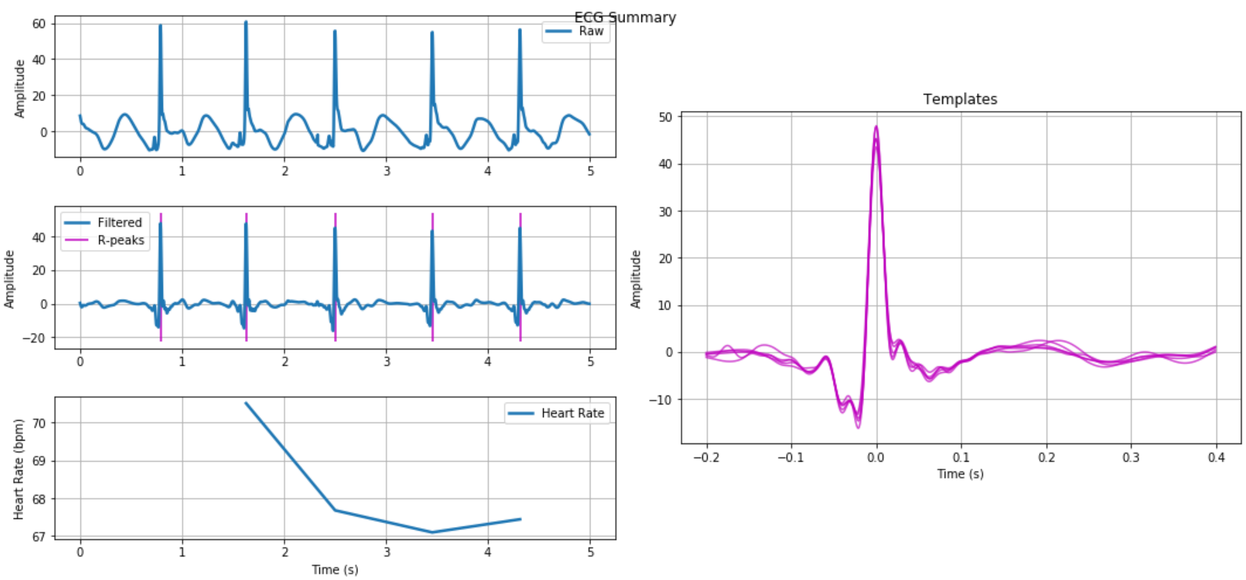
Plot a Control ECG Singal

```
In [10]: out_a_con = ecg.ecg(signal=select_random(con_signals), sampling_rate=sampling_freq, show=True)
```



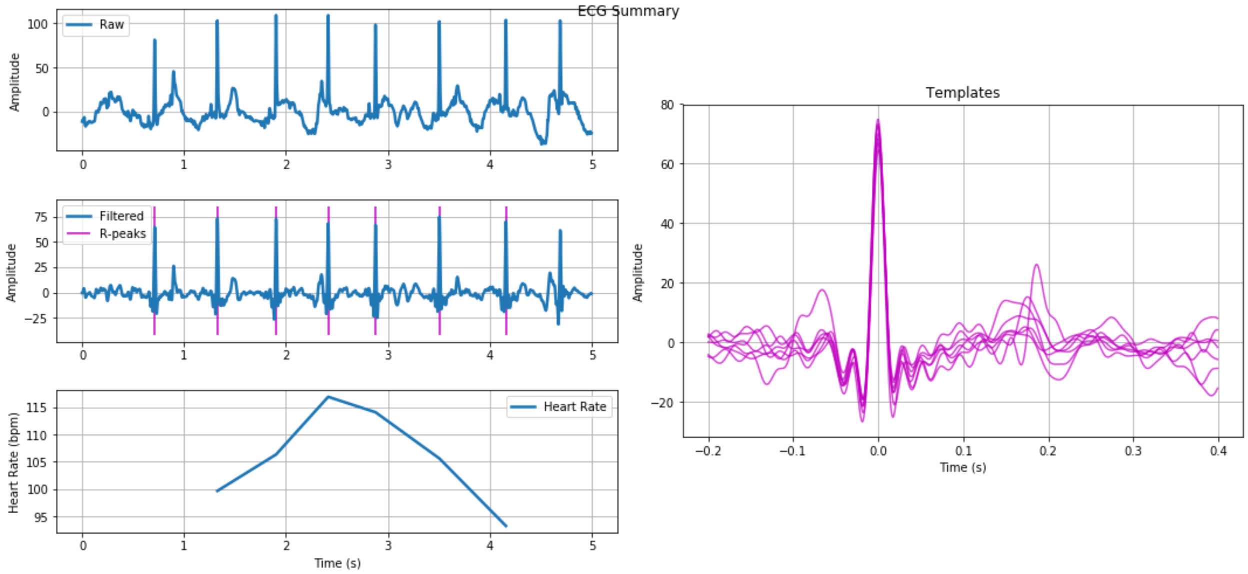
Plot a High P Amplitude ECG Singal

```
In [11]: out_a_highp = ecg.ecg(signal=select_random(highp_signals), sampling_rate=sampling_freq, show=True)
```



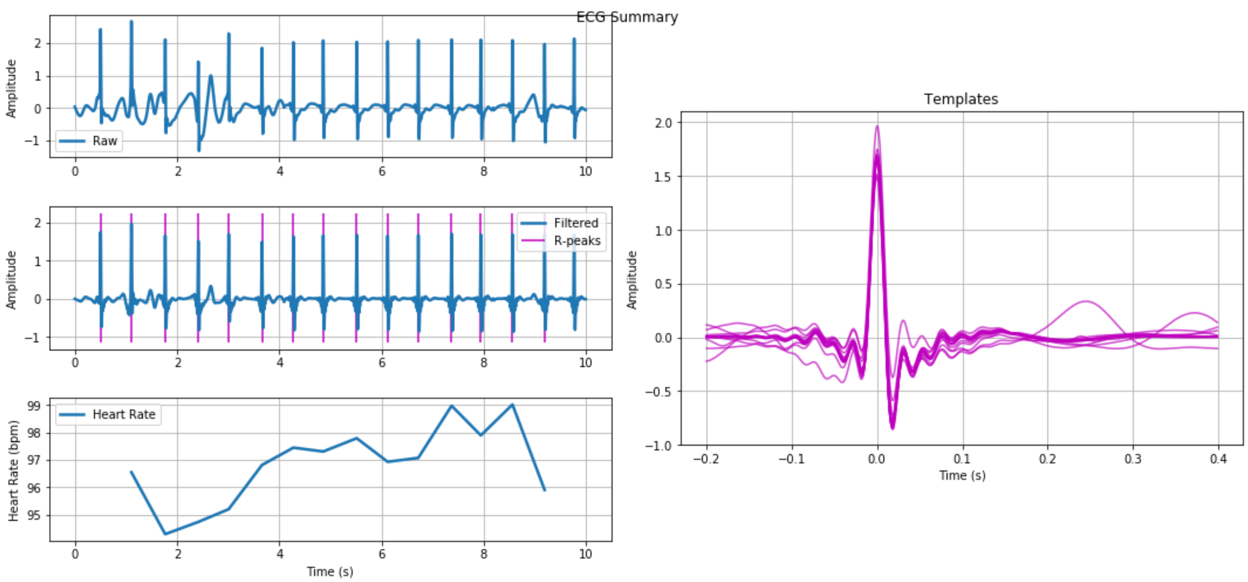
Plot a SA ECG Singal

In [12]:
```
out_a_sa = ecg.ecg(signal=select_random(sa_signals), sampling_rate=sampling_freq, show=True)
```



Plot a ST ECG Singal

In [13]:
```
out_a_st = ecg.ecg(signal=select_random(st_signals), sampling_rate=sampling_freq, show=True)
```



In [14]:
```
plt.rcParams['figure.figsize'] = [6, 4]
```

Scale signal amplitudes

In [15]:
```
# scaler = MinMaxScaler(feature_range=(-1, 1))
scaler = StandardScaler()
def scale_singals(signals):
    return [scaler.fit_transform(signal.reshape(-1, 1)).flatten() for signal in signals]

con_signals_scaled = scale_singals(con_signals)
highp_signals_scaled = scale_singals(highp_signals)
sa_signals_scaled = scale_singals(sa_signals)
st_signals_scaled = scale_singals(st_signals)
```

Extract R-R intervals

In [16]:
```
def cal_r_peaks(signal, sampling_rate=sampling_freq):
    rpeaks, = ecg.hamilton_segmenter(signal=signal, sampling_rate=sampling_rate)
    rpeaks, = ecg.correct_rpeaks(signal=signal, rpeaks=rpeaks, sampling_rate=sampling_rate, tol=0.05)
    templates, rpeaks = ecg.extract_heartbeats(signal=signal, rpeaks=rpeaks, sampling_rate=sampling_ra
te, before=0.2, after=0.4)
    return rpeaks

def get_r_peaks(signals):
    return Parallel(n_jobs=6)(delayed(cal_r_peaks)(signal) for signal in signals)
```

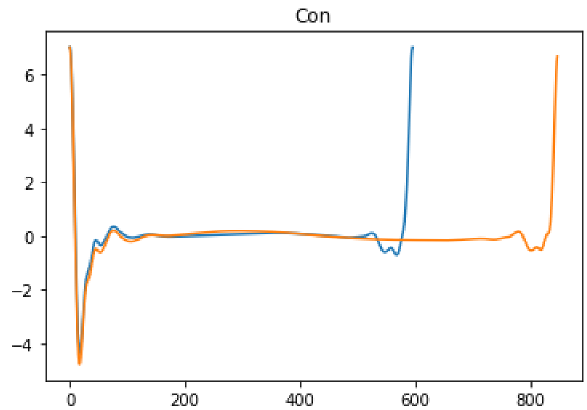
In [17]:
```
con_r_peaks = get_r_peaks(con_signals_scaled)
highp_r_peaks = get_r_peaks(highp_signals_scaled)
sa_r_peaks = get_r_peaks(sa_signals_scaled)
st_r_peaks = get_r_peaks(st_signals_scaled)
```

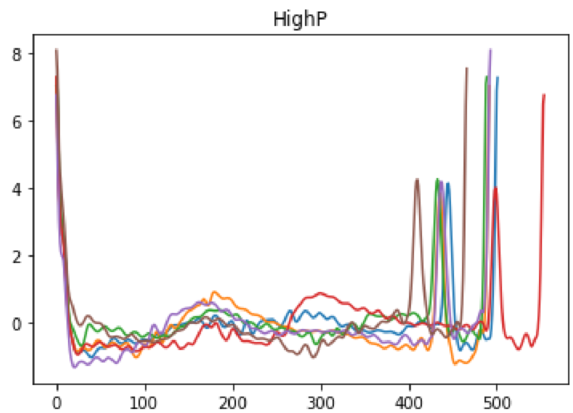
In [18]:
```
def split_list(alist, indices):
    splitted_list = []
    for start, end in zip(indices, indices[1:]):
        sublist = alist[start:end+1]
        splitted_list.append(sublist)
    return splitted_list

def extract_rr_intervals(signals, r_peaks):
    rr_itvs = []
    for idx, signal in enumerate(signals):
        if r_peaks[idx] is None:
            continue
        rr_itvs.append(split_list(signal, r_peaks[idx].tolist()))
    return rr_itvs
```

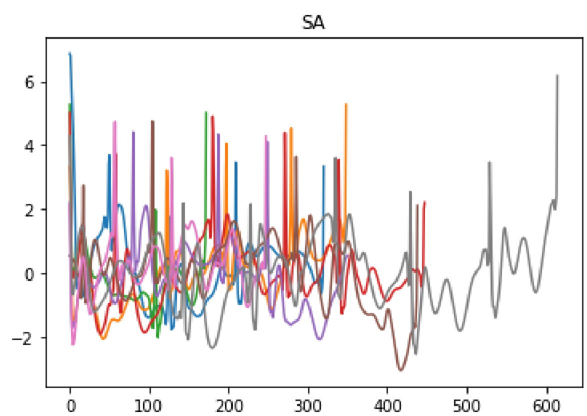
In [19]:
```
con_rr_itvs = extract_rr_intervals(con_signals_scaled, con_r_peaks)
highp_rr_itvs = extract_rr_intervals(highp_signals_scaled, highp_r_peaks)
sa_rr_itvs = extract_rr_intervals(sa_signals_scaled, sa_r_peaks)
st_rr_itvs = extract_rr_intervals(st_signals_scaled, st_r_peaks)
```

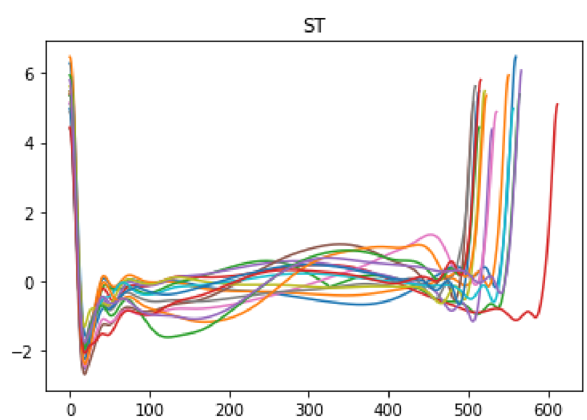
In [20]:
```
plt.rcParams['figure.figsize'] = [6, 4]
def plot_rr_intervals(rr_intervals, anomaly):
    for rr in rr_intervals[1:-1]:
        plt.plot(rr)
    plt.title(anomaly)
    plt.show()
```

In [21]:
```
plot_rr_intervals(select_random(con_rr_itvs), 'Con')
plot_rr_intervals(select_random(highp_rr_itvs), 'HighP')
plot_rr_intervals(select_random(sa_rr_itvs), 'SA')
plot_rr_intervals(select_random(st_rr_itvs), 'ST')
```



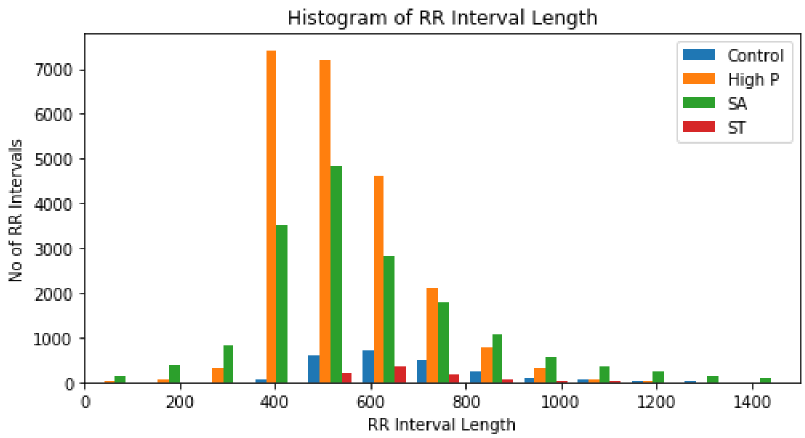




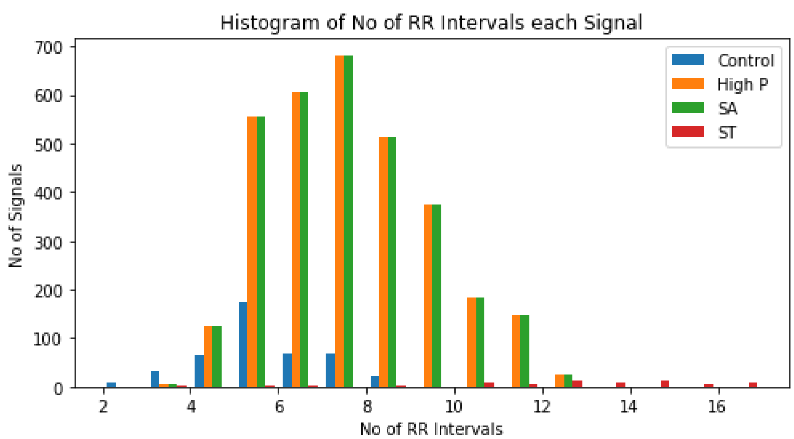


R-R intervals are normalized to a specific length

In [22]:
```
plt.rcParams['figure.figsize'] = [8, 4]
plt.hist([get_len_sublists(con_rr_itvs, level=2), get_len_sublists(highp_rr_itvs, level=2), \
          get_len_sublists(sa_rr_itvs, level=2), get_len_sublists(st_rr_itvs, level=2)], 30, label=ano
malies)
plt.legend(loc='upper right')
plt.xlabel('RR Interval Length')
plt.ylabel('No of RR Intervals')
plt.title('Histogram of RR Interval Length')
plt.xlim(0, 1500)
plt.show()
```



In [23]:
```
plt.rcParams['figure.figsize'] = [8, 4]
plt.hist([get_len_sublists(con_rr_itvs), get_len_sublists(highp_rr_itvs), \
          get_len_sublists(highp_rr_itvs), get_len_sublists(st_rr_itvs)], 15, label=anomalies)
plt.legend(loc='upper right')
plt.xlabel('No of RR Intervals')
plt.ylabel('No of Signals')
plt.title('Histogram of No of RR Intervals each Signal')
plt.show()
```



In [24]:
```
rr_normalized_size = 1000
max_rr_intervals = 18

def linear_interpolation(rr_interval):
    rr_interp = interp.interp1d(np.arange(rr_interval.size), rr_interval)
    return rr_interp(np.linspace(0,rr_interval.size-1, rr_normalized_size))

def pre_pad_sequence(sequence):
    max_length = max_rr_intervals
    seq_length = sequence.shape[0]
    if seq_length == max_length:
        return sequence
    elif seq_length < max_length:
        dim = max_length - seq_length
        zeros_seq  = np.zeros((max_length - seq_length, rr_normalized_size))
        return np.concatenate([zeros_seq, sequence])
    else:
        return sequence[seq_length - max_length:]

def normalize_rr_intervals(signal_rr_intervals):
    signal_rr_itvs_normed = []
    for signal in signal_rr_intervals:
        rr_itvs_normed = []
        if len(signal) == 0:
            continue
        for rr in signal:
            rr_itvs_normed.append(linear_interpolation(rr))
        rr_itvs_normed_padded = pre_pad_sequence(np.array(rr_itvs_normed))
        signal_rr_itvs_normed.append(np.array(rr_itvs_normed_padded))
    return np.array(signal_rr_itvs_normed)
```

In [25]:
```
con_rr_itvs_normed = normalize_rr_intervals(con_rr_itvs)
highp_rr_itvs_normed = normalize_rr_intervals(highp_rr_itvs)
sa_rr_itvs_normed = normalize_rr_intervals(sa_rr_itvs)
st_rr_itvs_normed = normalize_rr_intervals(st_rr_itvs)
```

R-R Interval Dimension Reduction with Autoencoder

In [26]:
```
X = np.concatenate([con_rr_itvs_normed, highp_rr_itvs_normed, sa_rr_itvs_normed, st_rr_itvs_normed])
y = np.concatenate([[[1, 0, 0, 0]] * len(con_rr_itvs_normed), \
                    [[0, 1, 0, 0]] * len(highp_rr_itvs_normed), \
                    [[0, 0, 1, 0]] * len(sa_rr_itvs_normed), \
                    [[0, 0, 0, 1]] * len(st_rr_itvs_normed)])
X, y = shuffle(X, y, random_state=seed)
```

In [27]:
```
#Split train, evaluation, and test sets
eval_split_pos = int(len(y) * .7)
test_split_pos = int(len(y) * .8)
X_train, y_train = X[:eval_split_pos], y[:eval_split_pos]
print('X_train.shape', X_train.shape)
X_eval, y_eval = X[eval_split_pos:test_split_pos], y[eval_split_pos:test_split_pos]
print('X_eval.shape', X_eval.shape)
X_test, y_test = X[test_split_pos:], y[test_split_pos:]
print('X_test.shape', X_test.shape)

# Reshape rr sequences into unordered rr intervals and remove zero padded rr rows
X_train_AE = X_train.reshape(-1, rr_normalized_size)
print('X_train.reshape', X_train_AE.shape)
all_zeros_rows_train = (X_train_AE==0).all(1)
X_train_AE = X_train_AE[~all_zeros_rows_train]
print('X_train_AE.shape', X_train_AE.shape)

X_eval_AE = X_eval.reshape(-1, rr_normalized_size)
print('X_eval.reshape', X_eval_AE.shape)
all_zeros_rows_eval = (X_eval_AE==0).all(1)
X_eval_AE = X_eval_AE[~all_zeros_rows_eval]
print('X_eval_AE.shape', X_eval_AE.shape)

X_test_AE = X_test.reshape(-1, rr_normalized_size)
print('X_test.reshape', X_test_AE.shape)
all_zeros_rows_test = (X_test_AE==0).all(1)
X_test_AE = X_test_AE[~all_zeros_rows_test]
print('X_test_AE.shape', X_test_AE.shape)
```

```
('X_train.shape', (4506, 18, 1000))
('X_eval.shape', (644, 18, 1000))
('X_test.shape', (1288, 18, 1000))
('X_train.reshape', (81108, 1000))
('X_train_AE.shape', (30079, 1000))
('X_eval.reshape', (11592, 1000))
('X_eval_AE.shape', (4267, 1000))
('X_test.reshape', (23184, 1000))
('X_test_AE.shape', (8694, 1000))
```

In [28]:
```
encoding_dim = 500
#Create 2-layer AE
input_rr = Input(shape=(rr_normalized_size,))
encoded = Dense(1024, activation='relu')(input_rr)
encoded = Dense(encoding_dim, activation='relu')(encoded)
decoded = Dense(1024, activation='relu')(encoded)
decoded = Dense(rr_normalized_size)(encoded)

autoencoder = Model(input_rr, decoded)
encoder = Model(input_rr, encoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')
autoencoder.fit(X_train_AE, X_train_AE, epochs=25, batch_size=64, \
                validation_data=(X_eval_AE, X_eval_AE))
```

```
Train on 30079 samples, validate on 4267 samples
Epoch 1/25
30079/30079 [==============================] - 2s 67us/step - loss: 0.0795 - val_loss: 0.0328
Epoch 2/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0273 - val_loss: 0.0307
Epoch 3/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0270 - val_loss: 0.0243
Epoch 4/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0213 - val_loss: 0.0318
Epoch 5/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0208 - val_loss: 0.0176
Epoch 6/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0156 - val_loss: 0.0382
Epoch 7/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0181 - val_loss: 0.0174
Epoch 8/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0162 - val_loss: 0.0172
Epoch 9/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0137 - val_loss: 0.0167
Epoch 10/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0127 - val_loss: 0.0148
Epoch 11/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0137 - val_loss: 0.0177
Epoch 12/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0148 - val_loss: 0.0131
Epoch 13/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0128 - val_loss: 0.0160
Epoch 14/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0102 - val_loss: 0.0161
Epoch 15/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0126 - val_loss: 0.0144
Epoch 16/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0100 - val_loss: 0.0182
Epoch 17/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0101 - val_loss: 0.0152
Epoch 18/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0121 - val_loss: 0.0125
Epoch 19/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0111 - val_loss: 0.0122
Epoch 20/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0105 - val_loss: 0.0277
Epoch 21/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0112 - val_loss: 0.0137
Epoch 22/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0108 - val_loss: 0.0125
Epoch 23/25
30079/30079 [==============================] - 2s 53us/step - loss: 0.0088 - val_loss: 0.0124
Epoch 24/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0086 - val_loss: 0.0119
Epoch 25/25
30079/30079 [==============================] - 2s 52us/step - loss: 0.0118 - val_loss: 0.0127
```

Out[28]:
```
<keras.callbacks.History at 0x7fa47c431890>
```

In [29]:
```
#Indices to reinsert zero padded rows
from operator import itemgetter
from itertools import groupby

def __get_continuous_ranges(indices):
    ranges = []
    for k, g in groupby(enumerate(indices), lambda (i,x):i-x):
        group = map(itemgetter(1), g)
        ranges.append((group[0], group[-1]))
    return ranges

zeros_indices_train = np.nonzero(all_zeros_rows_train)[0]
zeros_indices_eval = np.nonzero(all_zeros_rows_eval)[0]
zeros_indices_test = np.nonzero(all_zeros_rows_test)[0]

zero_ranges_train = __get_continuous_ranges(zeros_indices_train)
zero_ranges_eval = __get_continuous_ranges(zeros_indices_eval)
zero_ranges_test = __get_continuous_ranges(zeros_indices_test)
```

In [30]:
```
#Encode rr intervals
X_train_AE_encoded = encoder.predict(X_train_AE)
print('X_train_AE_encoded.shape', X_train_AE_encoded.shape)
X_eval_AE_encoded = encoder.predict(X_eval_AE)
print('X_eval_AE_encoded.shape', X_eval_AE_encoded.shape)
X_test_AE_encoded = encoder.predict(X_test_AE)
print('X_test_AE_encoded.shape', X_test_AE_encoded.shape)

#Reinsert zero padded rows
for arange in zero_ranges_train:
    X_train_AE_encoded = np.insert(X_train_AE_encoded, arange[0], np.zeros((arange[1] - arange[0] + 1,
 encoding_dim)), 0)
print('X_train_AE_encoded.shape', X_train_AE_encoded.shape)

for arange in zero_ranges_eval:
    X_eval_AE_encoded = np.insert(X_eval_AE_encoded, arange[0], np.zeros((arange[1] - arange[0] + 1, e
ncoding_dim)), 0)
print('X_eval_AE_encoded.shape', X_eval_AE_encoded.shape)

for arange in zero_ranges_test:
    X_test_AE_encoded = np.insert(X_test_AE_encoded, arange[0], np.zeros((arange[1] - arange[0] + 1, e
ncoding_dim)), 0)
print('X_test_AE_encoded.shape', X_test_AE_encoded.shape)

# Reshape to rr sequences
X_train_dim_reduced = X_train_AE_encoded.reshape((-1, max_rr_intervals, encoding_dim))
print('X_train_dim_reduced.shape', X_train_dim_reduced.shape)
X_eval_dim_reduced = X_eval_AE_encoded.reshape(-1, max_rr_intervals, encoding_dim)
print('X_eval_dim_reduced.shape', X_eval_dim_reduced.shape)
X_test_dim_reduced = X_test_AE_encoded.reshape(-1,max_rr_intervals, encoding_dim)
print('X_test_dim_reduced.shape', X_test_dim_reduced.shape)
```

```
('X_train_AE_encoded.shape', (30079, 500))
('X_eval_AE_encoded.shape', (4267, 500))
('X_test_AE_encoded.shape', (8694, 500))
('X_train_AE_encoded.shape', (81108, 500))
('X_eval_AE_encoded.shape', (11592, 500))
('X_test_AE_encoded.shape', (23184, 500))
('X_train_dim_reduced.shape', (4506, 18, 500))
('X_eval_dim_reduced.shape', (644, 18, 500))
('X_test_dim_reduced.shape', (1288, 18, 500))
```

Anomaly Classification with LSTM

In [31]:
```
# create the model
model = Sequential()
model.add(LSTM(64, input_shape=(max_rr_intervals, encoding_dim)))
model.add(Dropout(0.3))
model.add(Dense(4, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train_dim_reduced, y_train, epochs=20, batch_size=32, class_weight = {0:5., 1:1., 2:1., 3:
10.}, \
        validation_data=(X_eval_dim_reduced, y_eval))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 64)                144640
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 260
=================================================================
Total params: 144,900
Trainable params: 144,900
Non-trainable params: 0
_________________________________________________________________
None
Train on 4506 samples, validate on 644 samples
Epoch 1/20
4506/4506 [==============================] - 3s 745us/step - loss: 1.2068 - acc: 0.6869 - val_loss:
0.5085 - val_acc: 0.8137
Epoch 2/20
4506/4506 [==============================] - 3s 610us/step - loss: 0.7501 - acc: 0.7985 - val_loss:
0.4174 - val_acc: 0.8478
Epoch 3/20
4506/4506 [==============================] - 3s 614us/step - loss: 0.5153 - acc: 0.8546 - val_loss:
0.4607 - val_acc: 0.8106
Epoch 4/20
4506/4506 [==============================] - 3s 613us/step - loss: 0.3992 - acc: 0.8844 - val_loss:
0.3052 - val_acc: 0.8944
Epoch 5/20
4506/4506 [==============================] - 3s 617us/step - loss: 0.3034 - acc: 0.9154 - val_loss:
0.2312 - val_acc: 0.9193
Epoch 6/20
4506/4506 [==============================] - 3s 613us/step - loss: 0.2350 - acc: 0.9292 - val_loss:
0.1820 - val_acc: 0.9270
Epoch 7/20
4506/4506 [==============================] - 3s 619us/step - loss: 0.2198 - acc: 0.9399 - val_loss:
0.1737 - val_acc: 0.9224
Epoch 8/20
4506/4506 [==============================] - 3s 615us/step - loss: 0.1606 - acc: 0.9527 - val_loss:
0.1042 - val_acc: 0.9627
Epoch 9/20
4506/4506 [==============================] - 3s 620us/step - loss: 0.1284 - acc: 0.9629 - val_loss:
0.0992 - val_acc: 0.9674
Epoch 10/20
4506/4506 [==============================] - 3s 611us/step - loss: 0.1068 - acc: 0.9703 - val_loss:
0.1681 - val_acc: 0.9379
Epoch 11/20
4506/4506 [==============================] - 3s 621us/step - loss: 0.1198 - acc: 0.9636 - val_loss:
0.0906 - val_acc: 0.9627
Epoch 12/20
4506/4506 [==============================] - 3s 618us/step - loss: 0.1214 - acc: 0.9627 - val_loss:
0.0607 - val_acc: 0.9860
Epoch 13/20
4506/4506 [==============================] - 3s 614us/step - loss: 0.0932 - acc: 0.9738 - val_loss:
0.0572 - val_acc: 0.9783
Epoch 14/20
4506/4506 [==============================] - 3s 621us/step - loss: 0.0566 - acc: 0.9851 - val_loss:
0.0453 - val_acc: 0.9829
Epoch 15/20
4506/4506 [==============================] - 3s 613us/step - loss: 0.0548 - acc: 0.9834 - val_loss:
0.0392 - val_acc: 0.9860
Epoch 16/20
4506/4506 [==============================] - 3s 621us/step - loss: 0.0564 - acc: 0.9834 - val_loss:
0.0732 - val_acc: 0.9689
Epoch 17/20
4506/4506 [==============================] - 3s 614us/step - loss: 0.0451 - acc: 0.9905 - val_loss:
0.1497 - val_acc: 0.9317
Epoch 18/20
4506/4506 [==============================] - 3s 618us/step - loss: 0.1904 - acc: 0.9510 - val_loss:
0.1025 - val_acc: 0.9705
Epoch 19/20
4506/4506 [==============================] - 3s 611us/step - loss: 0.1224 - acc: 0.9707 - val_loss:
```

```
0.0578 - val_acc: 0.9876
Epoch 20/20
4506/4506 [==============================] - 3s 624us/step - loss: 0.0970 - acc: 0.9727 - val_loss:
0.0350 - val_acc: 0.9938
```

Out[31]: <keras.callbacks.History at 0x7fa4a014f610>

In [32]:

```
y_preds = np.argmax(model.predict(X_test_dim_reduced), axis=1)
y_trues = np.argmax(y_test, axis=1)

print(classification_report(y_trues, y_preds, target_names=anomalies))
print('Confusion Matrix:\n')
print(confusion_matrix(y_trues, y_preds, labels=[0, 1, 2, 3]))
```

```
             precision    recall  f1-score   support

    Control       0.99      0.99      0.99        77
     High P       1.00      0.94      0.97       660
         SA       0.93      0.99      0.96       537
         ST       1.00      0.86      0.92        14

avg / total       0.97      0.97      0.97      1288

Confusion Matrix:

[[ 76   0   1   0]
 [  0 622  38   0]
 [  0   3 534   0]
 [  1   0   1  12]]
```