In [3]:

```
%%javascript
IPython.OutputArea.prototype._should_scroll = function(lines) {
    return false;
}
```

Project Summary

Objective - To detect the 'V' beats in Electrocardiogram (ECG)

Steps: -Read the train and test dataset -Extract the signal ,R peaks,Beat Types -Segment the signal around R peaks using moving window of size 180(90 left and 90 right of R peak) -Extract the RR interval features and wavlet coefficints using wavelet decomposition -Create dataframe using features and labels -Standardize dataset to have zero mean and unit variance.

To deal with imbalance in the dataset ,performed stratified Kfold cross validation on training data to avoid overfitting. Oversampling with SMOTE is not performed as it didnt improve model performance.

-Performed stratified Kfold cross validation with weighted classes using SVM model with C paramter values= [0.001,0.01,0.1,1,10]

-Trained the SVM model with C=1 which gave following best scores (least variance and highest average values across 5 folds)

Log loss score# (1) mean: 0.001808918327451133 (2)variance: 1.3437300689043264e-06 F1 score# (1) mean: 0.99967585089141 (2)variance: 4.202905783986499e-07 Accuracy score# (1) mean: 0.9994974874371859 (2)variance: 1.0100755031438243e-06 Precision score# (1) mean: 1.0 (2)variance: 0.0 Recall score# (1) mean: 0.9993527508090615 (2)variance: 1.6757260606821902e-06

-Used the trained model to predict V beats of test data.

Future Improvements

I have included only RR and wavelet coefficient features for modelling and it showed very good performance.

Some important features like QRS width,QRS amplitude etc can be considered for improving the detection. Also filtering the raw ecg signal before segmentation will remove the noise and improve detection.

References

As am not much familiar with signal processing domain,I referred several research papers to do this project.
1.https://www.hindawi.com/journals/cmmm/2018/1380348/
(https://www.hindawi.com/journals/cmmm/2018/1380348/)
2.https://pdfs.semanticscholar.org/271a/4037bc46272b86a5861d7e19fe80fd3cb498.pdf
(https://pdfs.semanticscholar.org/271a/4037bc46272b86a5861d7e19fe80fd3cb498.pdf)
3.https://www.computer.org/csdl/pds/api/csdl/proceedings/download-article/07838258/pdf?
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJjc2RsX2FwaSIsImF1ZCI6ImNzZGxfYXBpX2Rvd25
NCtVyoY6b0LyDFp_Ifj-ajuKVj531CSAPoAqo
(https://www.computer.org/csdl/pds/api/csdl/proceedings/download-article/07838258/pdf?
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJjc2RsX2FwaSIsImF1ZCI6ImNzZGxfYXBpX2Rvd25

NCtVyoY6b0LyDFp_Ifj-ajuKVj531CSAPoAqo) 4.https://arxiv.org/pdf/1005.0957.pdf (https://arxiv.org/pdf/1005.0957.pdf) 5.https://www.sciencedirect.com/science/article/pii/S1746809418301976 (https://www.sciencedirect.com/science/article/pii/S1746809418301976)

In [4]:

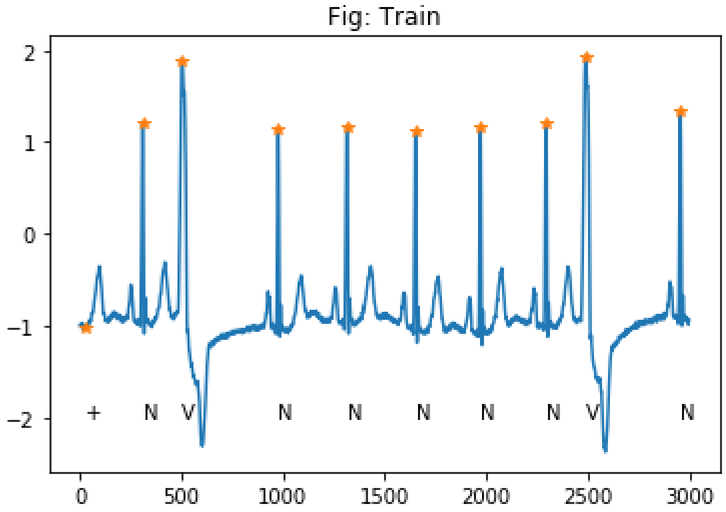
```
#Supporting functions:
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import wfdb
import matplotlib.pyplot as plt
from scipy import signal as ss
import numpy as np
import pywt
from sklearn import decomposition
from sklearn.decomposition import PCA, IncrementalPCA
from sklearn import svm
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler,RobustScaler
from sklearn.model_selection import StratifiedKFold,train_test_split,cross_val_score
from collections import OrderedDict
import pandas as pd
from sklearn.metrics import log_loss,auc,precision_score, recall_score, f1_score, ro
label_dict = {0: 'V', 1: 'N'}
training_file_path = './database/train/a4'
testing_file_path = './database/test/b1'


def read_ecg(file_path,isoutput=False):
    """
    output: ecg files, get signal, annotated peaks, annotated types
    input: ecg file id
    """
    signals, fields = wfdb.rdsamp(file_path)
    if not isoutput:
        annotation = wfdb.rdann(file_path, 'atr')
    else:
        annotation = wfdb.rdann(file_path, 'test')
    ecg_sig = signals[:,0]
    ecg_type = annotation.symbol
    ecg_peak = annotation.sample
    return ecg_sig, ecg_type, ecg_peak

def plot_ecg(ecg_sig, ecg_type, ecg_peak, title='Fig: Train', npeak=10, len_sig=300(
    """
    demo plot ecg signal with annotated peaks, annotated types
    """
    _, ax = plt.subplots()
    for i in range(0, npeak):
        if state == 'test':
            ax.annotate(ecg_type[i], xy=(ecg_peak[i], -1))
        else:
            ax.annotate(ecg_type[i], xy=(ecg_peak[i], -2))
    ax.plot(ecg_sig[0:len_sig])
    ax.plot(ecg_peak[0:npeak], ecg_sig[ecg_peak[0:npeak]], '*')
    ax.set_title(title)

def plot_output(ecg_sig, ecg_type, ecg_peak, title='Fig: Train', npeak=10, len_sig=
    """
    plot output ecg signal with annotated peaks, annotated types
    """
    _, ax = plt.subplots()
    for i in range(0, npeak):
        ax.annotate(ecg_type[i], xy=(ecg_peak[i+1], -2))
```

```
    ax.plot(ecg_sig[0:len_sig])
    ax.plot(ecg_peak[1:npeak], ecg_sig[ecg_peak[1:npeak]], '*')
    ax.set_title(title)
```

```
/Users/stephygeorge/.virtualenvs/dl4cv/lib/python3.6/site-packages/skl
earn/externals/six.py:31: DeprecationWarning: The module is deprecated
in version 0.21 and will be removed in version 0.23 since we've droppe
d support for Python 2.7. Please rely on the official version of six
(https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

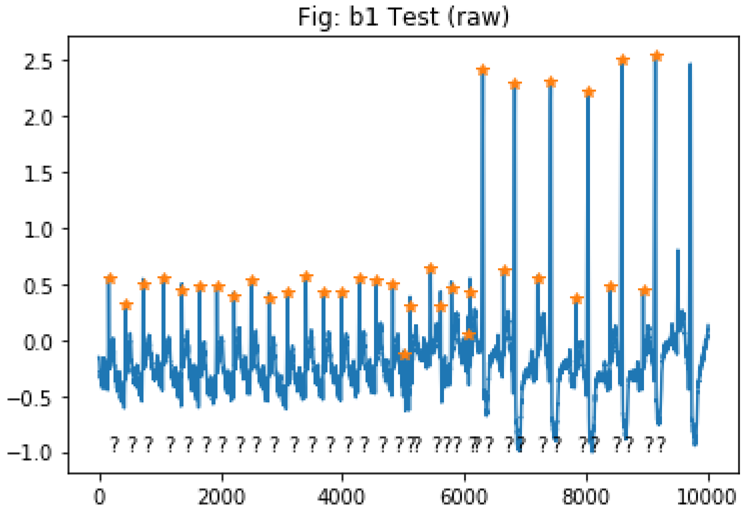
In [5]:

```
#What's included in the training database?
#1. The training database contains a group of ECG signals and annotations:
#2. In each annotation, the type of each beat is annotated on its peak (N, V, ...),
#   For example:

trn_ecg_sig, trn_ecg_type, trn_ecg_peak = read_ecg(training_file_path,isoutput=False
plot_ecg(trn_ecg_sig, trn_ecg_type, trn_ecg_peak)
```



In [6]:

```
#What's included in the testing database?
#1. The testing database contains a group of ECG signals. However,
#2. Type of each beat is unknown (?, ?, ...). For example:
# 'b2 test file'
# tst_ecg_sig, tst_ecg_type, tst_ecg_peak = read_ecg(testing_file_path1,isoutput=Fa
# plot_ecg(tst_ecg_sig, tst_ecg_type, tst_ecg_peak, title='Fig: b2 Test (raw)', len_

'b1 test file'
tst_ecg_sig, tst_ecg_type, tst_ecg_peak = read_ecg(testing_file_path,isoutput=False)
plot_ecg(tst_ecg_sig, tst_ecg_type, tst_ecg_peak, title='Fig: b1 Test (raw)',npeak=3
```



Segmentation of signal around R peaks

In [7]:

```
def create_segments(signal, r_peaks, beat_types, win_L, win_R):
    """
    Uses a moving window to segment the signal around R peaks
    :param signal: raw_signal
    :param r_peaks: r peak positions
    :param beat_types: N or V or ?
    :param win_L: length of left segment window :90
    :param win_R: length of right segment window :90
    :return: segments and labels
    """
    # Prepare containers
    beats, labels = [], []
    # Prepare the moving window
    for idx, annot in enumerate(list(zip(beat_types, r_peaks))):
        beat_type = annot[0]  # "N", "V", ... etc.
        r_peak_pos = annot[1]  # The R peak position
        if beat_type in("N","V","?"):
            if beat_type == "N":
                id = 1
            elif beat_type == "V":
                id = 0
            else:
                id = -1
            if r_peak_pos >= 0:
                if r_peak_pos < win_L:
                    # nxt_R_peak_pos = r_peaks[idx + 1]
                    beats.append(signal[0:win_L + win_R])
                    labels.append(id)
                elif (r_peak_pos > win_L and r_peak_pos < (len(signal) - win_R)):
                    beats.append(signal[r_peak_pos - win_L: r_peak_pos + win_R])
                    labels.append(id)

    # Turn into arrays
    beats = np.array(beats)
    labels = np.array(labels)

    return beats, labels
```

Extract features from beats :RR interval features and Wavelet coefficients using wavelet decomposition

In [8]:

```
def get_RR_intervals(r_peaks,idx):
    """
    Gets pre and post RR lengths of each beat
    :param r_peaks: list of R wave peaks
    :param idx: index
    :return: pre_RR and post_RR lengths
    """

    # Pre_R and Post_R
    if idx == 0:
        pre_R = 0
        post_R = r_peaks[1] - r_peaks[0]

    if idx > 0 and idx < len(r_peaks) - 1:
        pre_R =  r_peaks[idx] - r_peaks[idx - 1]
        post_R = r_peaks[idx + 1] - r_peaks[idx]


    if idx == len(r_peaks) - 1:
        pre_R = r_peaks[-1] - r_peaks[-2]
        # post_R = post_R
        post_R = r_peaks[idx] - r_peaks[idx - 1]


    return pre_R, post_R


def get_wavlet_coeffs(beat, family, level):
    """
    Computes wavlet transform on each beat extract coefficients
    :param beat: signal segments
    :param family: db1
    :param level: 3
    :return: coefficients
    """
    wave_family = pywt.Wavelet(family)
    coeffs = pywt.wavedec(beat, wave_family, level=level)
    return coeffs[0]


def get_features(beats, r_peaks):
    """
    Gets RR intervals and wavlet coefficients of each beat
    :param beats: signal segments
    :param r_peaks: list of R wave peaks
    :return:
    """
    rr_coeff_list = list()
    print("3a.Get RR features.")
    print("3b.Get wavelet transform coeffs.")
    for idx,item in enumerate(list(zip(beats, r_peaks))):
        beat=item[0]
        # get RR intervals for each beat
        pre_R, post_R = get_RR_intervals(r_peaks,idx)

        #get wavelet transform coefficients
        wave_coeffs = get_wavlet_coeffs(beat, 'db1', 3)

        lst_beat_row = ()
```

```
        lst_beat_row = [
            ("pre-RR", pre_R), ("post-RR", post_R)
        ]
        for i, coeff in enumerate(wave_coeffs):
            lst_beat_row.append(("wave_coeff{}".format(i), coeff))

        beat_row_dict = OrderedDict(lst_beat_row)

        rr_coeff_list.append(beat_row_dict)
    return rr_coeff_list
```

Create dataset with the extracted features

In [9]:

```
def create_dataset(ecg_sig, r_peaks, beat_type, win_L, win_R):
    """
    Create a dataframe after segmenting beats and extracting features
    :param ecg_sig: ecg_signal
    :param r_peaks: R peak list
    :param beat_type: type of beat(N/V/?)
    :param win_L: left segment window
    :param win_R: right segment window
    :return:
    """
    # Prepare containers
    signals, labels = [], []

    # Convert signal into labeled fragments
    print('2.Segment signal into beats.')
    signal, label = create_segments(ecg_sig, r_peaks, beat_type, win_L, win_R)
    # create segments and labels list
    signals.append(signal)
    labels.append(label)
    # Convert to one huge numpy.array
    signals = np.vstack(signals)
    labels = np.vstack(labels)

    print('3.Extract features from beats.')
    features_list = get_features(signals, r_peaks)
    print('4.Create dataset using features.')
    train_df = pd.DataFrame(features_list)
    train_df['type'] = labels.T
    print("Dataframe with features and labels:")
    print("--" * 40)
    print(train_df.head())
    print(train_df.shape)

    return train_df
```

In [10]:

```
#Data Preprocessing steps
def oversample_data(df, labels):
    """
    oversample train data using SMOTE
    """
    print('6.Oversampling imbalanced dataset after normalising dataset.')
    oversamp = SMOTE(ratio='minority', random_state=None, k_neighbors=5, m_neighbors
                     svm_estimator=None, n_jobs=1)
    df, labels = oversamp.fit_sample(df, labels)
    print("Oversampled row count : {}".format(df.shape[0]))
    print("Number of Normal beats after oversampling: {}".format(len(labels[labels =
    print("Number of V beats after oversampling    : {}".format(len(labels[labels ==
    return df, labels


def normalise(df):
    """
    Standardize to zero mean and unit variance -(z-score)
    :param df: train/test dataset
    :return:
    """
    std = RobustScaler()
    x = df.values
    x_scaled = std.fit_transform(x)
    # df_temp = pd.DataFrame(x_scaled, columns=num_cols, index = df.index)
    # df[num_cols] = df_temp
    return x_scaled
```

Preprocess dataset

Preprocessing steps summary:

1.Drop missing value rows-

```
-There were no missing values in both test and train dataset</li>
```

2.Standardize data to have zero mean and unit variance as it helps the model to train faster

In [11]:

```
def process_dataset(df,state='train'):
    """
    -Drop any missing values
    -standardize input

    :param df: train/test dataset
    :return:
    """
    # drop if any missing values
    df.dropna(inplace=True)
#     print(df.shape)

    # Classes are imbalanced
    if state == 'train':
        print("Number of Normal beats : {}".format(df[df['type'] == 1].shape[0]))
        print("Number of V beats     : {}".format(df[df['type'] == 0].shape[0]))
        print('Classes are imbalanced')

    df = df.copy()
    y = df['type'].values
    df.drop('type', axis=1, inplace=True)

    # Normalised to zero mean and unit variance
    print("5.Standardize dataset to zero mean and unit variance.")
    df = normalise(df)
    print("Dataset after normalisation :")
    print("--" * 40)
    print(df[5,:])
    print("Shape of Data after normalisation : {}" .format(df.shape))
    res = [df,y]

    return res


def prepare_data(state='train'):
    """
    -Read datasets
    -Prepare and preprocess dataframe
    :param state: train/test
    :return: df,labels
    """
    #create dataset from the signal data
    win_L=90
    win_R=90
    if state == 'train':
        print('1.Reading training data.')
        ecg_sig,ecg_type,ecg_peak = read_ecg(training_file_path,isoutput=False)
        plot_ecg(ecg_sig,ecg_type,ecg_peak,state='train')
    else:
        print('1.Reading testing data.')
        ecg_sig,ecg_type,ecg_peak  = read_ecg(testing_file_path,isoutput=False)
        plot_ecg(ecg_sig,ecg_type,ecg_peak, title='Fig:b1 Test (raw)',npeak=35, len_

    df = create_dataset(ecg_sig,ecg_peak,ecg_type,win_L,win_R)
    df,labels = process_dataset(df,state)
    return df,labels
```

In [12]:

```
def crossvalidate(model, df, labels):
    '''
    Crossvalidate train data using StratifiedKFold with 5 splits

    :param model: svm_model
    :param df: train/test
    :param labels: train_labels
    :return:
    '''

    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    scoring = ['neg_log_loss', 'f1','accuracy','precision','recall']

    scores = cross_validate(model, df, labels, cv=skf, scoring=scoring)
    print('Log loss score# (1) mean: {} (2)variance: {}'.format(-np.mean(scores['tes
                                                                 np.var(scores['test_
    print('F1 score# (1) mean: {} (2)variance: {}'.format(np.mean(scores['test_f1'])
    print('Accuracy score# (1) mean: {} (2)variance: {}'.format(np.mean(scores['test
    print('Precision score# (1) mean: {} (2)variance: {}'.format(np.mean(scores['tes
                                                                 np.var(scores['test_
    print('Recall score# (1) mean: {} (2)variance: {}'.format(np.mean(scores['test_r
                                                                  np.var(scores['test
```

In [13]:

```
def train():
    """
     Train SVM model using balanced class weights and C=1 after cross validation
     Test the model
    :return:
    """
    class_weights ={}
    print('Preparing training data.....')
    df_train, train_labels = prepare_data(state='train')

    # find class weights
    for c in range(len(label_dict)):
        class_weights.update({c: len(train_labels) / float(np.count_nonzero(train_la

#     print(class_weights)
    '''
    Oversampling was not performed as the model performed better with balanced weigh
    '''
#     df_train_re, train_labels_re = oversample_data(df_train, train_labels)



    print('Training SVM...')
    #Crossvalidated with set of C values
    c_svm=[0.001,0.01,0.1,1,10]
    for c in c_svm:
        # svm_model = svm.SVC(C=c,gamma='auto',
        #                       class_weight = 'balanced',probability=True,
        #                       random_state=42)
        model = svm.SVC(C=c, gamma='auto',
                            class_weight=class_weights, probability=True,
                            random_state=42)
        print('SVM cross validation scores when C: {}'.format(c))
        print("--" * 40)
        # crossvalidate(svm_model, df_train_re, train_labels_re)
        crossvalidate(model, df_train, train_labels)
        print("--" * 40)

    #Create model with C=1
    svm_model = svm.SVC(C=1, gamma='auto',
                        class_weight=class_weights, probability=True,
                        random_state=42)
    svm_model.fit(df_train, train_labels)
    return svm_model
```

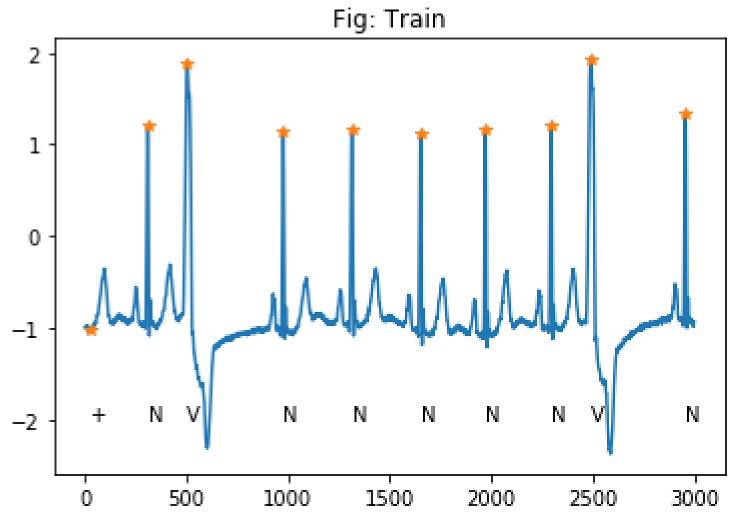
In [14]:

```
trained_model = train()
```

```
Preparing training data.....
1.Reading training data.
2.Segment signal into beats.
3.Extract features from beats.
3a.Get RR features.
3b.Get wavelet transform coeffs.
4.Create dataset using features.
Dataframe with features and labels:
----------------------------------------------------------------------
----------
   pre-RR  post-RR  wave_coeff0  wave_coeff1  wave_coeff2  wave_coeff3
\
0       0      277    -2.619831    -2.566798    -2.365272    -1.941008
1     277      194    -0.982878    -1.193243    -1.739483    -2.237993
2     194      474    -2.854944    -2.831963    -2.844337    -2.605688
3     474      338    -2.672864    -2.612760    -2.390021    -1.979899
4     338      336    -2.731200    -2.676399    -2.494319    -2.068287

   wave_coeff4  wave_coeff5  wave_coeff6  wave_coeff7  ...  wave_coeff
14  \
0    -1.712966    -2.368808    -2.687006    -2.701148  ...     -2.6781
67
1    -2.473106    -2.556191    -2.545584    -2.474874  ...     -2.3652
72
2    -2.207941    -1.896814    -2.428912    -2.839034  ...     -2.9468
68
3    -1.700592    -2.236225    -2.715290    -2.717058  ...     -2.6516
50
4    -1.842013    -2.400628    -2.840801    -2.869086  ...     -2.9645
45

   wave_coeff15  wave_coeff16  wave_coeff17  wave_coeff18  wave_coeff1
9  \
0     -2.763020     -2.780697     -2.718826     -2.701148     -2.56856
5
1     -3.399416     -3.851964     -4.118897     -4.399972     -4.52548
3
2     -2.964545     -2.969848     -3.012275     -2.925654     -2.83196
3
3     -2.741807     -2.750645     -2.718826     -2.692309     -2.60215
3
4     -3.008739     -3.021114     -3.010507     -2.994597     -2.86024
7

   wave_coeff20  wave_coeff21  wave_coeff22  type
0     -2.411234     -2.211476     -1.873833     1
1     -4.578516     -4.826004     -5.377547     0
2     -2.736503     -2.501390     -2.308704     1
3     -2.522603     -2.342291     -2.110714     1
4     -2.722361     -2.503158     -2.181424     1

[5 rows x 26 columns]
(1987, 26)
Number of Normal beats : 1543
Number of V beats     : 444
Classes are imbalanced
5.Standardize dataset to zero mean and unit variance.
```

```
Dataset after normalisation :
--------------------------------------------------------------------------------
[ 0.12244898 -0.00826446 -0.43966547 -0.52350699 -0.50598802 -0.39194139
  0.0753012  -0.07503234 -0.60787402 -0.6294964  -0.71902269 -0.37974684
 -0.38018018 -0.84528302 -0.53715499 -0.54928018 -0.55514706 -0.14285714
 -0.09929078 -0.02702703  0.1031746   0.22280472  0.26029216  0.49538867
  0.63076923]
Shape of Data after normalisation : (1987, 25)
Training SVM...
SVM cross validation scores when C: 0.001
--------------------------------------------------------------------------------
Log loss score# (1) mean: 0.008306947330012119 (2)variance: 8.017456276695462e-06
F1 score# (1) mean: 0.9967469047464281 (2)variance: 5.301664077338547e-06
Accuracy score# (1) mean: 0.9949723364296229 (2)variance: 1.2613216075326486e-05
Precision score# (1) mean: 1.0 (2)variance: 0.0
Recall score# (1) mean: 0.9935254066322028 (2)variance: 2.0932991750467078e-05
--------------------------------------------------------------------------------
SVM cross validation scores when C: 0.01
--------------------------------------------------------------------------------
Log loss score# (1) mean: 0.00494872578620965 (2)variance: 4.681595234920692e-06
F1 score# (1) mean: 0.9980519480519481 (2)variance: 2.5299375948725315e-06
Accuracy score# (1) mean: 0.9969849246231156 (2)variance: 6.060453018863212e-06
Precision score# (1) mean: 1.0 (2)variance: 0.0
Recall score# (1) mean: 0.996116504854369 (2)variance: 1.0054356364093487e-05
--------------------------------------------------------------------------------
SVM cross validation scores when C: 0.1
--------------------------------------------------------------------------------
Log loss score# (1) mean: 0.0028424469915679165 (2)variance: 2.8027016014151805e-06
F1 score# (1) mean: 0.9990265002420594 (2)variance: 1.6859427733483718e-06
Accuracy score# (1) mean: 0.9984924623115579 (2)variance: 4.040302012575454e-06
Precision score# (1) mean: 1.0 (2)variance: 0.0
Recall score# (1) mean: 0.9980582524271846 (2)variance: 6.702904242728961e-06
--------------------------------------------------------------------------------
SVM cross validation scores when C: 1
--------------------------------------------------------------------------------
Log loss score# (1) mean: 0.001808918327451133 (2)variance: 1.3437300689043264e-06
```

```
F1 score# (1) mean: 0.99967585089141 (2)variance: 4.202905783986499e-0
7
Accuracy score# (1) mean: 0.9994974874371859 (2)variance: 1.0100755031
438243e-06
Precision score# (1) mean: 1.0 (2)variance: 0.0
Recall score# (1) mean: 0.9993527508090615 (2)variance: 1.675726060682
1902e-06
-----------------------------------------------------------------------
----------
SVM cross validation scores when C: 10
-----------------------------------------------------------------------
----------
Log loss score# (1) mean: 0.0018089183274511333 (2)variance: 1.34373006
89043264e-06
F1 score# (1) mean: 0.99967585089141 (2)variance: 4.202905783986499e-0
7
Accuracy score# (1) mean: 0.9994974874371859 (2)variance: 1.0100755031
438243e-06
Precision score# (1) mean: 1.0 (2)variance: 0.0
Recall score# (1) mean: 0.9993527508090615 (2)variance: 1.675726060682
1902e-06
-----------------------------------------------------------------------
----------
```



In [15]:

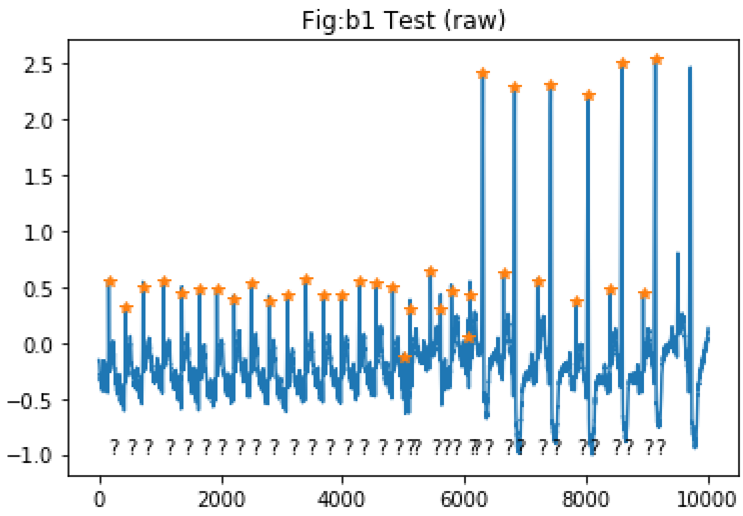
```
def test_model(model):
    """
    Predict and plot V beats using trained SVM model
    :param model: trained svm model
    :return:
    """
    print('Testing SVM...')
    print("--" * 40)

    print('Preparing testing data.....')
    df_test, _ = prepare_data(state='test')
    # df_test_re, test_labels_re = oversample_data(df_test, test_labels)
    print('Predicting beat types using trained SVM...')
    predictions = model.predict(df_test)
    ecg_sig, ecg_type, ecg_peak = read_ecg(testing_file_path,isoutput=False)
    pred_type=['V' if i == 0 else '?' for i in predictions]
    plot_ecg(ecg_sig, pred_type, ecg_peak, title='Fig: b1 Test (V detected)', npeak=
#     wfdb.wrann('b2', 'test', ecg_peak[0:len(ecg_peak) - 1], pred_type, write_dir=
    wfdb.wrann('b1', 'test', ecg_peak, pred_type, write_dir='./database/test/')
```

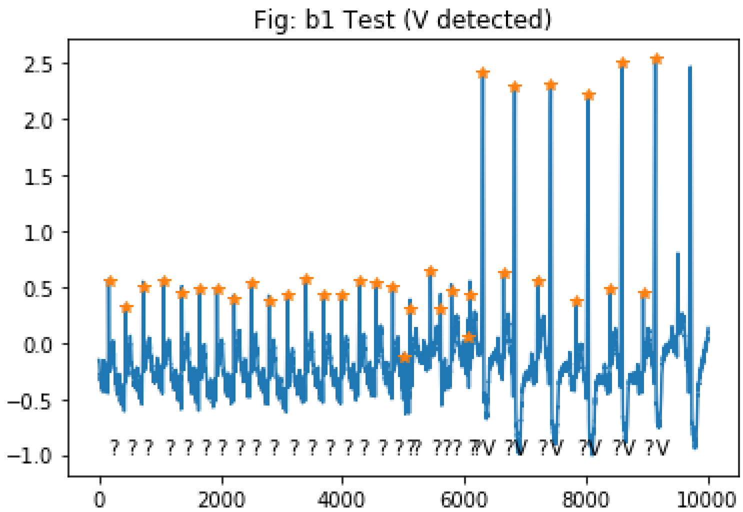
In [16]:

```
test_model(trained_model)
```

```
Testing SVM...
----------------------------------------------------------------------
-----------
Preparing testing data.....
1.Reading testing data.
2.Segment signal into beats.
3.Extract features from beats.
3a.Get RR features.
3b.Get wavelet transform coeffs.
4.Create dataset using features.
Dataframe with features and labels:
----------------------------------------------------------------------
-----------
   pre-RR  post-RR  wave_coeff0  wave_coeff1  wave_coeff2  wave_coeff
3  \
0       0      273    -0.565685    -0.647003    -0.958130    -1.13844
2
1     273      293    -1.140210    -1.382394    -1.453104    -1.49199
5
2     293      335    -1.094248    -1.138442    -1.226830    -1.29930
9
3     335      305    -0.814941    -0.869741    -0.958130    -1.07657
0
4     305      285    -1.134906    -1.244508    -1.272792    -1.38946
5

   wave_coeff4  wave_coeff5  wave_coeff6  wave_coeff7  ...  wave_coef
f14  \
0    -1.058892    -1.159655    -1.073035    -0.972272  ...     -0.648
770
1    -1.543261    -1.384162    -1.308148    -1.198546  ...     -0.814
941
2    -1.258650    -1.025305    -0.972272    -0.882116  ...     -0.618
718
3    -0.910400    -0.731856    -0.689429    -0.648770  ...     -0.480
833
4    -1.408910    -1.101319    -1.051821    -0.899793  ...     -0.595
737

   wave_coeff15  wave_coeff16  wave_coeff17  wave_coeff18  wave_coeff
19  \
0     -0.523259     -0.418961     -0.307591     -0.167938     -0.0919
24
1     -0.714178     -0.595737     -0.484368     -0.337643     -0.1679
38
2     -0.602809     -0.546240     -0.496743     -0.358857     -0.2704
68
3     -0.461387     -0.401283     -0.353553     -0.256326     -0.2421
84
4     -0.537401     -0.364160     -0.357089     -0.293449     -0.0424
26

   wave_coeff20  wave_coeff21  wave_coeff22  type
0     -0.022981     -0.049497     -0.201525    -1
1     -0.125511      0.097227     -0.003536    -1
2     -0.113137     -0.125511      0.028284    -1
```

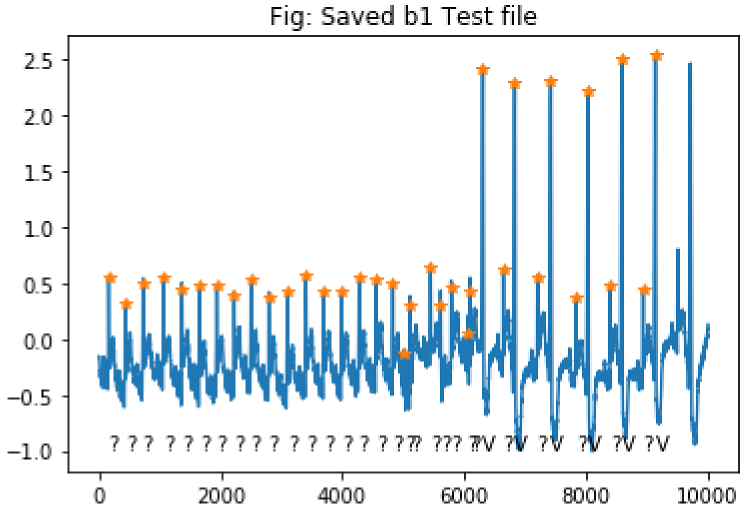
```
3      -0.114905      -0.010607      0.045962     -1
4       0.019445       0.000000      0.148492     -1

[5 rows x 26 columns]
(2097, 26)
5.Standardize dataset to zero mean and unit variance.
Dataset after normalisation :
---------------------------------------------------------------------
-----------
[-0.44262295 -0.45901639 -0.48       -0.63636364 -0.80612245 -0.67241
379
 -1.13768116 -1.19105691 -1.25       -0.78030303 -0.84459459 -0.97906
977
 -1.04176904 -0.70574163 -0.72372372 -0.39197531 -0.64985163 -0.50750
751
 -0.57391304 -0.63384615 -0.58227848 -0.4691358  -0.4617737  -0.47706
422
 -0.80981595]
Shape of Data after normalisation : (2097, 25)
Predicting beat types using trained SVM...
```





In [17]:

```
'Plot the output signal with detected v beats from saved test file'

tst_ecg_sig, tst_ecg_type, tst_ecg_peak = read_ecg(testing_file_path,isoutput=True)
plot_ecg(tst_ecg_sig, tst_ecg_type, tst_ecg_peak, title='Fig: Saved b1 Test file',np
```



In [ ]: