Project 2.1 : Biobots Activity Detection

Hariharan Ramshankar
NC State University
Raleigh, NC
hramsha@ncsu.edu

Pooja Mehta
NC State University
Raleigh, NC
pmehta@ncsu.edu

Prathamesh Prabhudesai
NC State University
Raleigh, NC
ppprabhu@ncsu.edu

Abstract

Activity recognition from temporal data is an widely studied problem with far reaching applications especially in the fitness and health care industries. This report includes a comparative study between K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Bagged Trees approaches based on their F-1 scores and accuracy. The best performance is obtained by Fine KNN (Single Neighbor) with maximum F1 score of 0.84.

Keywords: activity detection, temporal data, bio-bots, bagging

1. Introduction

Insect bio-bots have a number of unique advantages like the ability to crawl into hidden spaces over synthetic robots. A swarm of such insects could revolutionize the search and rescue scenario. In the past, researchers at NC State University have exploited the random nature of cockroach movement for mapping collapsed structures. [1] The objective of this report is to compare the performance of various classifiers on the data received from the Inertial Measurement Units (IMU) on the bio-bots. In particular, the ability to distinguish between left and right movement is analyzed.

2. Dataset Description

Each sample in the dataset is of the form:

1. F_train.t - They are the times of the observation (N x 1 matrix)
2. F_train.f - They are the feature values (N x 42 matrix)
3. F_train.valid - A flag indicating if the sample is valid (N x 1 matrix)

The features computed over a 1 second time window with 75% overlap are:

1. Mean (of Accelerometer and Gyro values)
2. Variance (of Accelerometer and Gyro values)
3. Skewness (of Accelerometer and Gyro values)
4. Kurtosis (of Accelerometer and Gyro values)
5. Cross Correlation (between Accelerometer and Gyro)
6. Gyro Energy

The ground truth represents 4 different modes or classes: (0) Stationary, (1) moving in the middle of the arena, (2) moving Clockwise (CW) on the boundary, and (3) moving Counter-Clockwise (CCW) on the boundary.

3. Classification

Classification using various classifiers such as, Support Vector Machine, Logistic Regression, K-means Clustering, K-Nearest Neighbors etc was performed. The Classification Learner toolbox was used for training[2]. The best performing classifiers are described in this section.

3.1. K-Nearest Neighbors

K-Nearest Neighbors (KNN for short) is a *non-parametric, lazy learning* algorithm. It is *non-parametric* since it does not make any assumptions on the underlying data distribution. It is useful as in the real world, most of the time the data does not obey theoretical assumptions. The term *lazy* means there is no explicit training phase or it is very minimal. [3]

Simple KNN algorithm can be stated as follows: [4]

1. For each training example, $< x, f(x) >$, add the example to the list of training examples (training data).
2. Given a query instance, x_q to be classified,
   - Let $x_1, x_2, ..., x_k$ denote the k instances from training data that are nearest to x_q.
   - Return the class that represents the maximum of the k instances.

K is nothing but the number of neighbors taken into consideration while deciding the class for query instance. When $K = 1$, the predicted class of the element is the class of its closest sample. It is also known as **Fine KNN**. We tested Fine KNN with Euclidean, Cosine and Manhat-

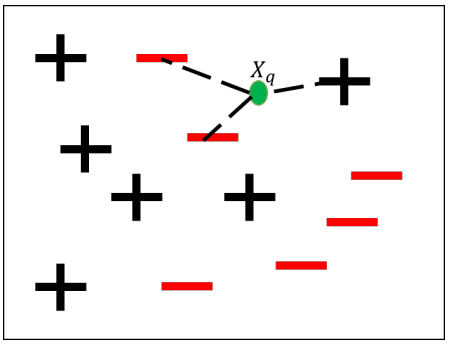

Figure 1. KNN Visualization: In this example, k=3. So to classify x_q it takes three nearest neighbors. It can be see that, it is closer to the two examples from the negative class, and hence will be classified as a negative example.

tan Distance measures. These distance measures are defined below. Distance is always calculated between two points and these points are defined by their feature vectors (features as co-ordinates). Consider an n-dimensional Cartesian space and two points $p = (p_1, p_2, p_3, ..., p_n)$ and $q = (q_1, q_2, q_3, ..., q_n)$.

- **Euclidean Distance:** The Euclidean distance is the length of the straight line connecting these two points.

$$d_{euclidean}(p, q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)}$$

.

- **Cosine Distance:** The cosine distance between these two points is given as:

$$d_{cos}(p, q) = 1 - \frac{p_1.q_1 + p_2.q_2 + ... + p_n.q_n}{\sqrt{p_1^2 + p_2^2 + ... + p_n^2}\sqrt{q_1^2 + q_2^2 + ... + q_n^2}}$$

- **City-block distance:**The City block distance or *Manhattan* distance between these two points, is the 4-neighborhood distance.

$$d_{cityblock} = \sum_{i=1}^{n} |p_i - q_i|$$

This distance is always greater than or equal to zero. The higher the similarity, the closer the value is to zero. Identical points have a distance of zero. Hence, we got the highest accuracy and F1 score for KNN using City Block Distance.

3.2. Support Vector Machine

Figure 2 shows a case for separable data, where two possible hyper-planes are shown. Both of these planes correctly separate the data and are viable options for a classifier. But, the dotted plane is a better option, since it is more generalized. SVM selects the hyperplane which has the largest minimum distance in the training examples.

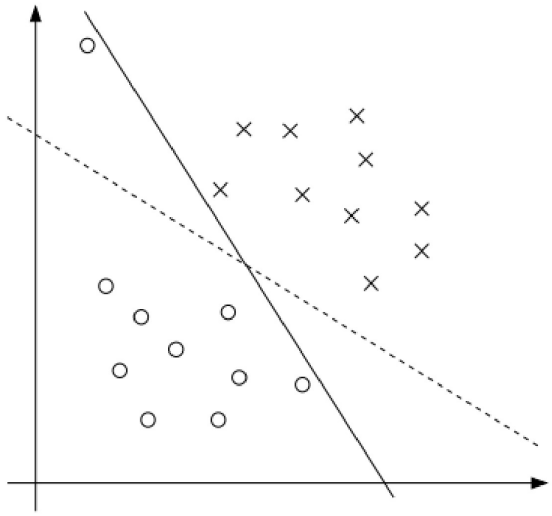

Figure 2. Possible for separable data

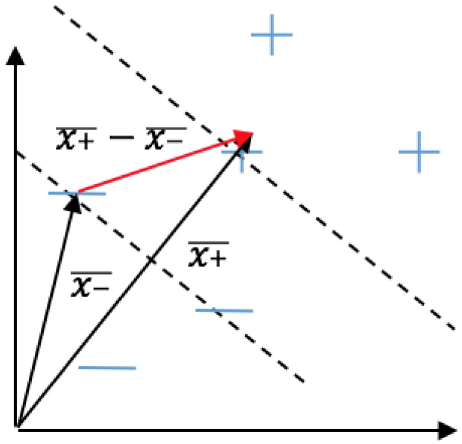

Figure 3. Possible for separable data

SVM is also known as a **maximum margin classifier**, since it maximizes the difference vector $\bar{x+} - \bar{x-}$, as seen in figure 5, in order to select the optimal hyper-plane.

In case of non-linear data, a kernel functional must be used to transform the features into a linear space. Data in this project, as is the case with most real world data, is non-linear. Testing included using the radial basis function as well as the polynomial kernel functions for feature space

transformation. Cubic polynomial gave the best results for the project data.

3.3. Bagged Trees

Bagged Trees is an Ensemble Learning method, where based on the number of learners, decision trees are created and an ensemble is formed. Maximum Voting is used to decide the class for query sample.

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. **Bagging** is used to reduce the variance and smoother decision boundaries. Bagging is useful in cases where the classifiers are unstable i.e a small change in the training set leads to drastic change in the result.

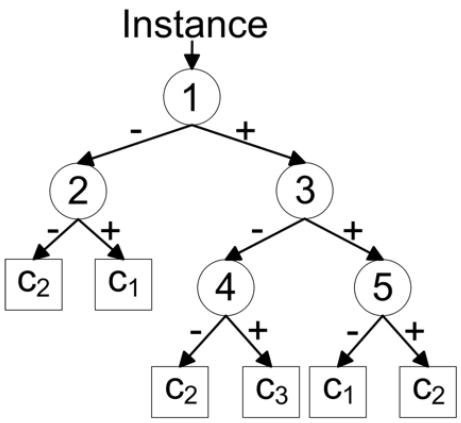

Figure 4. Decision Tree Example

Figure 4 shows the example of a basic decision tree. When at every split, if all the features are considered to make a decision, it is called as Bagged Tree. Algorithm for Bagged Trees can be stated as follows:

Training Phase:

- Initialize the parameters. $D = \phi$, the ensemble. L, the number of trainers.
- For $i = 1, 2, ..., L$ Build a decision tree D_i. Add it to the ensemble.
- Return D.

Classification Phase:

- Run $D_1, D-2, ..., D_L$ on query sample x_q.
- The class with maximum number of votes is chosen as the label for x_q.

4. Results

Results obtained by different classifiers described in section 3 are presented below. 10 Fold K-cross validation is used and the final results are obtained by averaging over the results obtained on validation sets. Performance of classifier is decided using *Precision, Recall and F-score* values.

4.1. Definitions: Precision, Recall, and F-Score

1. **Precision:** The number of correctly classified positive examples divided by the number of examples labeled by the system as positive. [6]

$$Precision = \frac{t_p}{t_p + f_p} \tag{1}$$

2. **Recall:** The number of correctly classified positive examples divided by the number of positive examples in the data. [6]

$$Recall = \frac{tp}{tp + fn} \tag{2}$$

3. F_β **Score:** a combination of the above.[6]

$$F_\beta - Score = \frac{(\beta^2 + 1)tp}{(\beta^2 + 1)tp + \beta^2 fn + fp} \tag{3}$$

$$F_\beta - Score = \frac{(\beta^2 + 1)Precision_M * Recall_M}{\beta^2 Precision_M + Recall_M} \tag{4}$$

4.2. Fine K-Nearest Neighbors

Table 1 shows the precision, recall and F1 score obtained for each individual class. The KNN algorithm used the *city-block* distance metric $k = 1$. Class 2 suffers a bit while the others have F1 scores above 0.8.

Class	Precision	Recall	F1 Score
0	0.8327	0.8536	0.8430
1	0.8777	0.8805	0.8791
2	0.8058	0.7913	0.7985
3	0.8523	0.8334	0.8427

Table 1. Fine KNN: average precision = **0.842125**, average recall = **0.8397**, F1 Score = **0.8409**

4.3. Cubic SVM

Table 2 shows the precision, recall and F1 score obtained for each individual class. *Cubic* kernel is used for feature transformation.

Class	Precision	Recall	F1 Score
0	0.8279	0.8429	0.8304
1	0.8014	0.8408	0.8206
2	0.6990	0.6300	0.6627
3	0.7761	0.7200	0.7470

Table 2. Cubic SVM: average precision = **0.7761**, average recall = **0.758425**, F1 Score = **0.7672**

4.4. Bagged Trees

Table 3 shows the precision, recall and F1 score obtained for each individual class. As can be seen, Class 2 has more false negatives than others hence giving the lowest recall.

Class	Precision	Recall	F1 Score
0	0.8303	0.8782	0.8535
1	0.7452	0.9051	0.8173
2	0.7906	0.4439	0.55685
3	0.8389	0.6	0.6996

Table 3. Bagged Trees: average precision = **0.80125**, average recall = **0.706575**, F1 Score = **0.7510**.

4.5. Summary

Table 4 shows the average precision, average recall, F1 scores and the accuracy for all the methods tested.

Classifier	Avg. Prec.	Avg. Rec.	F1 Score	Accuracy
Fine KNN	**0.8421**	**0.8397**	**0.8409**	**85.39%**
Cubic SVM	0.7761	0.7584	0.7672	78.94%
Bagged Trees	0.8013	0.7066	0.7510	77.90%

Table 4. Performance Comparison: Fine KNN, Cubic SVM, Bagged Trees

It is observed that the highest accuracy and F1 score was obtained with the Fine KNN classifier. Since the data used in this project is a time-series data,it is hard to model it using the theoretical distribution functions. Fine KNN is a non-parametric algorithm and does not make assumptions about the data distribution. Hence gives the best performance in all the classifiers tested.

The cubic SVM also performed well with an F1 score of 0.7672 and an accuracy near 79%. The bagged trees method was the least effective classifier for this problem with an F1 score of 0.7510 and an accuracy of around 78%.

Looking at the performance for individual classes, since there are more training samples for class 0 and class 1, we get higher accuracy for these two classes. Class 2 and Class 3 suffer because of less training samples and considerable overlap.

5. Conclusion

All the classifiers struggled to differentiate between Clockwise and Counter-Clockwise Motion of the bio-bots mainly because of the significant overlap between the feature vectors describing these classes. Fine KNN produced better results than the SVM and Bagged Trees methods despite taking far lesser time to train because of its lazy learning and non-parametric behavior.

References

[1] Alper Bozkurt, Edgar Lobaton, Mihail Sichitiu, "A Biobotic Distributed Sensor Network for Under-Rubble Search and Rescue", Computer, vol. 49, pp. 38-46, 2016, ISSN 0018-9162. 1

[2] MATLAB Classification Learner Toolbox 1

[3] A Blog explaining various Machine Learning Algorithms. KNN Detailed Explanation 1

[4] http://www.csee.umbc.edu/tinoosh/cmpe650/slides/K_Nearest_Neighbor_Algorithm.pdf 1

[5] https://www.cs.rit.edu/~rlaz/prec20092/slides/Bagging_and_Boosting.pdf

[6] M. Sokolova and G. Laplame, "A Systematic Analysis of Performance Measures for Classification Tasks", Information Processing and Management 45, 2009. 3

Project 2.2 : HMM for Biobots Activity Detection

Prathamesh Prabhudesai
NC State University
Raleigh, NC
ppprabhu@ncsu.edu

Pooja Mehta
NC State University
Raleigh, NC
pmehta@ncsu.edu

Hariharan Ramshankar
NC State University
Raleigh, NC
hramsha@ncsu.edu

Abstract

Activity recognition from temporal data is a widely studied problem with far reaching applications especially in the fitness and health care industries. In the first part of the project various classification methods such as K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Bagged Trees were implemented. This report includes implementation of Hidden Markov Models(HMM) as they are well known for temporal data classification. It was observed that the F1 score improved from 0.84 to 0.95.

Keywords: activity detection, temporal data, bio-bots, Hidden Markov Models, emission, transition

1. Introduction

Insect bio-bots have a number of unique advantages like the ability to crawl into hidden spaces over synthetic robots. A swarm of such insects could revolutionize the search and rescue scenario. In the past, researchers at NC State University have exploited the random nature of cockroach movement for mapping collapsed structures. [1] The objective of this report is to improve the classification of the data received from the Inertial Measurement Units (IMU) on the bio-bots. In particular, the ability to distinguish between left and right movement is analyzed.

2. Dataset Description

Each sample in the dataset is of the form:

1. F_train.t - They are the times of the observation (N x 1 matrix)
2. F_train.f - They are the feature values (N x 42 matrix)
3. F_train.valid - A flag indicating if the sample is valid (N x 1 matrix)

The features computed over a 1 second time window with 75% overlap are:

1. Mean (of Accelerometer and Gyro values)
2. Variance (of Accelerometer and Gyro values)
3. Skewness (of Accelerometer and Gyro values)
4. Kurtosis (of Accelerometer and Gyro values)
5. Cross Correlation (between Accelerometer and Gyro)
6. Gyro Energy

The ground truth represents 4 different modes or classes: (0) Stationary, (1) moving in the middle of the arena, (2) moving Clockwise (CW) on the boundary, and (3) moving Counter-Clockwise (CCW) on the boundary.

3. Classification

Classification using HMM was performed. The Statistics and Machine Learning Toolbox was used for implementing the various functions.[3].

3.1. Hidden Markov Model

Hidden Markov Model is statistical Markov Model used for time series data, with unobserved or **hidden** states. The name encodes *hidden* because of the assumption that the observation at time t is generated by some process whose state S_t is *hidden* from the observer. The second assumption is that the process satisfies the **Markov** property, i.e. given the state S_{t-1} , the next state S_t is independent of all other states prior to S_{t-1}. In this project, the 4 class labels correspond to the 4 states and the 42 feature vectors are the observations. Our goal is to find the sequence of biobots activities given the observations.

The most likely sequence can be estimated using the Viterbi algorithm. Baum-Welch algorithm is used to parameterize the model using the training data.

3.1.1 Baum-Welch Algorithm

The Baum-Welch algorithm is used to find the unknown parameters of a HMM. It makes use of the forward-backward

algorithm. The HMM is described by the parameters: $a_{i,j}$, π and $b_j(y_t)$. Given the states S_t with N values, the transition probabilities are:

$$a_{i,j} = P(S_t = j | S_t - 1 = i)$$

The initial state distribution is given by

$$\pi_i = P(S_1 = i)$$

And the probability of observation at time t for state j is

$$b_j(y_t) = P(Y_t = y_t | S_t = j)$$

The algorithm can be stated as follows:

1. Begin with some model u(random or preselected)
2. Use the Forward and Backward procedure to determine the temporal variables.
   - Forward Procedure: Let

     $$\alpha = P(Y_1 = y_1, ..., Y_t = y_t, S_t = i | \theta)$$

     Calculate this recursively

     $$\alpha_1(1) = pi_i b_i(y_1)$$
     $$\alpha_j(t+1) = b_j(y_i + 1) \sum_{i=1}^{N} \alpha_i(t) a_{ij}$$

   - Backward Procedure: Calculate $\beta_i(t)$ as:

     $$\beta_i(T) = 1$$
     $$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_i j b_j$$

3. Update the model that gives maximum expectation values:

   $$\pi_i * = \gamma_i(1)$$

   $$a_i j = \frac{\sum_{i=1}^{T-1} \xi_{ij}(t)}{\sum_{i=1}^{T-1} \gamma_{ij}(t)}$$

   where

   $$\gamma_i(t) = P(S_i = i | Y, \theta) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}$$

   $$\xi_{ij}(t) = P(S_i = i, S_{i+1} = j | Y, \theta) = \frac{\alpha_i(t)\beta_i(t+1)a_{ij}b_j(y_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i(t)\beta_i(t+1)a_{ij}b_j(y_{t+1})} \quad (1)$$

3.1.2 Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm for finding the sequence of hidden states that is most likely to result in a sequence of observations. This sequence is known as the **Viterbi path**. The algorithm is ubiquitous in the communication field and is also heavily used in speech to text systems. The spoken text is the the observed sequence and the hidden cause is the string of text. The viterbi algorithm finds the most likely string of text given the acoustic signal.

"Suppose we are given a hidden Markov model (HMM) with state space S , initial probabilities π_i of being in state i and transition probabilities $a_{i,j}$ of transitioning from state i to state j.

Say we observe outputs $y_1, \ldots, y_T$.The most likely state sequence $x_1, \ldots, x_T$ that produces the observations is given by the recurrence relations:

$$\begin{aligned} V_{1,k} &= \mathrm{P}\big(y_1 \mid k\big) \cdot \pi_k \\ V_{t,k} &= \max_{x \in S} \big(\mathrm{P}\big(y_t \mid k\big) \cdot a_{x,k} \cdot V_{t-1,x}\big) \end{aligned}$$

Here $V_{t,k}$ is the probability of the most probable state sequence $\mathrm{P}\big(x_1, \ldots, x_T, y_1, \ldots, y_T\big)$ responsible for the first t observations that have k as its final state.
The Viterbi path can be retrieved by saving back pointers that remember which state x was used in the second equation. Let $\mathrm{Ptr}(k, t)$ be the function that returns the value of x used to compute $V_{t,k}$ if $t > 1, orkift = 1$. Then:

$$\begin{aligned} x_T &= \arg\max_{x \in S}(V_{T,x}) \\ x_{t-1} &= \mathrm{Ptr}(x_t, t) \end{aligned}$$

Here we're using the standard definition of $argmax$. The complexity of this algorithm is $O(T \times |S|^2)$."[2]

3.2. K-Nearest Neighbors

K-Nearest Neighbors (KNN for short) is a *non-parametric, lazy learning* algorithm. It is *non-parametric* since it does not make any assumptions on the underlying data distribution. It is useful as in the real world, most of the time the data does not obey theoretical assumptions. The term *lazy* means there is no explicit training phase or it is very minimal. [4]

Simple KNN algorithm can be stated as follows: [5]

1. For each training example, $< x, f(x) >$, add the example to the list of training examples (training data).
2. Given a query instance, x_q to be classified,
   - Let $x_1, x_2, ..., x_k$ denote the k instances from training data that are nearest to x_q.
   - Return the class that represents the maximum of the k instances.

K is nothing but the number of neighbors taken into consideration while deciding the class for query instance. When $K = 1$, the predicted class of the element is the class of its closest sample. It is also known as **Fine KNN**. We tested Fine KNN with Euclidean, Cosine and Manhattan Distance measures. These distance measures are defined below. Distance is always calculated between two points and these points are defined by their feature vectors (features as co-ordinates). Consider an n-dimensional Cartesian space and two points $p = (p_1, p_2, p_3, ..., p_n)$ and $q = (q_1, q_2, q_3, ..., q_n)$.

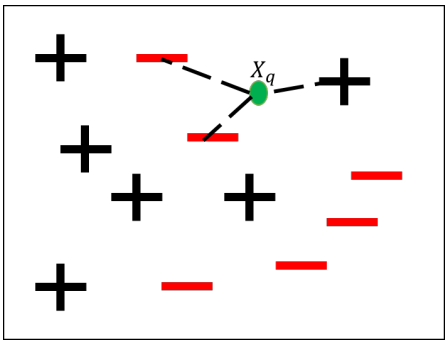

Figure 1. KNN Visualization: In this example, k=3. So to classify x_q it takes three nearest neighbors. It can be see that, it is closer to the two examples from the negative class, and hence will be classified as a negative example.

- **Euclidean Distance:** The Euclidean distance is the length of the straight line connecting these two points.

$$d_{euclidean}(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)}$$

.

- **Cosine Distance:** The cosine distance between these two points is given as:

$$d_{cos}(p,q) = 1 - \frac{p_1.q_1 + p_2.q_2 + ... + p_n.q_n}{\sqrt{p_1^2 + p_2^2 + ... + p_n^2}\sqrt{q_1^2 + q_2^2 + ... + q_n^2}}$$

- **City-block distance:**The City block distance or *Manhattan* distance between these two points, is the 4-neighborhood distance.

$$d_{cityblock} = \sum_{i=1}^{n} |p_i - q_i|$$

This distance is always greater than or equal to zero. The higher the similarity, the closer the value is to zero. Identical points have a distance of zero. Hence, we got the highest accuracy and F1 score for KNN using City Block Distance.

3.3. Implementation

Two methods were implemented to train the HMM. The first one involved Gaussian Mixture Model, but as there were not enough observations to capture the variability of the data, results were not good.

In the second method, output from a local classifier was used to estimate the HMM parameters and train the model. KNN local classifier was used as it gave the best performance in terms of F1 score, as tested in project 2.1. Detailed implementation is as follows:

1. Implement KNN classifier to predict each label using k-fold cross validation. This would give a sequence of time series data.

2. Using the output of KNN as the input sequence and the ground truth table of the training data as the states, estimate the transition and estimation matrices using 'hmmestimate' function.[3] This function calculates the Maximum Likelihood Estimate(MLE) of the transition and emission probabilities of a HMM for a sequence with known states. The estimated matrices are both of size 4x4. Input sequence is an output of the local classifier so has range from 0 to 3 (4 different classes) and even the states are also 4. Hence both the sizes are 4x4.

3. Improve the estimated transition and emission matrices using the 'hmmtrain' function.[3] To this function, we feed the output of the fine KNN classifier and use the Baum-Welch algorithm. The data is provided sequentially here as HMM is time dependent.

4. Determine the conditional probability that the model is in a particular state that generates the symbol, given that the sequence is generated. Use 'hmmdecode' using the estimated parameters produced in step 3.[3]

5. Generate the confusion matrix using 'confusionmat' to determine precision, recall along with F1 scores.[3]

3.4. Dealing with the Missing Data

In project 2.1, only the valid data (where all the 42 feature values are present) is used. In practice, because of various reasons such as communication loss, measurement device irregularity etc., the data (features) gets lost. The algorithm should be able to predict the states for such states where some or all the features are not available to describe it. It can be dealt two ways.

3.4.1 Interpolation

This is a time series data with 75% (considerable) overlap between two measurements. Due to the overlap and the short length (relatively) of the missing data sequences, we

can assume that there won't be sudden chnages in the states. This short span of missing features can be assumed to follow the linear nature and can be filled up with linear interpolation. After filling up all the gaps, implementation from section 3.3 can be followed. This assumption hampers the performance drastically.

3.4.2 Forward Algorithm based prediction

We train the Fine KNN classifier after removing the invalid data points, but save the indices for future use. The output of the KNN is then expanded to include the blank data points.To calculate the values for the missing points,we use the posterior distribution calculated till that point and multiply it with the transition matrix to obtain the probabilities for each class. Taking the maximum probability and assigning it as the predicted class, we fill blanks for the entire dataset. After filling the gaps, implementation from section 3.3 can be followed. This assumption produces far better results compared to the interpolation method described above, especially for classes 2 and 3.

4. Results

Since the HMM is time dependent, the input to the model has to be sequential. When the dataset is large, the model is tested with incremental training and validation sets (in terms of size of the data). For this project, since data was limited, 75-25(train-test percentages) and 50-50(train-test percentages) are used. Performance of classifier is decided using *Precision, Recall and F-score* values.

4.1. Definitions: Precision, Recall, and F-Score

1. **Precision:** The number of correctly classified positive examples divided by the number of examples labeled by the system as positive. [6]

$$Precision = \frac{t_p}{t_p + f_p} \tag{2}$$

2. **Recall:** The number of correctly classified positive examples divided by the number of positive examples in the data. [6]

$$Recall = \frac{tp}{tp + fn} \tag{3}$$

3. F_β **Score:** a combination of the above.[6]

$$F_\beta - Score = \frac{(\beta^2 + 1)tp}{(\beta^2 + 1)tp + \beta^2 fn + fp} \tag{4}$$

$$F_\beta - Score = \frac{(\beta^2 + 1)Precision_M * Recall_M}{\beta^2 Precision_M + Recall_M} \tag{5}$$

4.2. Fine K-Nearest Neighbors

Table 1 shows the precision, recall and F1 score obtained for each individual class. The KNN algorithm used the *city-block* distance metric $k = 1$. Class 2 suffers a bit while the others have F1 scores above 0.8.

Class	Precision	Recall	F1 Score
0	0.8327	0.8536	0.8430
1	0.8777	0.8805	0.8791
2	0.8058	0.7913	0.7985
3	0.8523	0.8334	0.8427

Table 1. Fine KNN: average precision = **0.842125**, average recall = **0.8397**, Macro F1 Score = **0.8409**

4.3. Hidden Markov Model

Table 2 shows the precision, recall and F1 score obtained for each individual class using 75% training data and 25% validation data using Viterbi algorithm.

Class	Precision	Recall	F1 Score
0	0.9464	0.9464	0.9463
1	0.9787	0.9559	0.9672
2	0.9190	0.9677	0.9427
3	0.9513	0.9695	0.9603

Table 2. HMM (75-25) Viterbi: average precision = **0.9489**, average recall = **0.9599**, Macro F1 Score = **0.9543**

Class 1 has a very high precision value of 0.9787 while class 2 has the lowest value at 0.9190. Recall shows slightly different characteristics. Class 3 comes out on top with a value of 0.9695 and class 0 has the least value of 0.9464. The highest F1 score of 0.9672 is achieved for class 1 with the lowest score of 0.9427 for class 2.

When compared to the results achieved by the Fine KNN classifier, we see that even the *lowest* scores generated by the HMM are *higher* than the *highest* scores achieved by the KNN classifier.

Table 3 shows the precision, recall and F1 score obtained for each individual class using 50% training data and 50% testing data split and the viterbi algorithm.

Class	Precision	Recall	F1 Score
0	0.8801	0.9313	0.9049
1	0.9640	0.9246	0.9438
2	0.9075	0.9282	0.9177
3	0.9500	0.9535	0.9518

Table 3. HMM (50-50) Viterbi: average precision = **0.9254**, average recall = **0.9344**, F1 Score = **0.9299**.

Class 1 has a high precision value of 0.9640 while class 0 has the lowest value at 0.8801. Recall shows slightly different characteristics.Class 3 comes out on top with a value of 0.9535 and class 1 has the least value of 0.9246. The highest F1 score of 0.9518 is achieved for class 3 with the lowest score of 0.9049 for class 0.

The scores achieved with a 50-50 split are lower than those achieved with a 75-25 split of the data. This makes sense, as there is lesser data to train with in the former case. Still, even with a 50-50 split, the HMM performs admirably well compared to the best classifier from our previous approach, i.e, Fine KNN.

Looking at the transition matrix we can glean some information about the behavior of the bio bot.

0.9617	0.0217	0.0010	0.0155
0.0107	0.9812	0.0038	0.0043
0.0114	0.0398	0.9489	0
0.0086	0.0129	0	0.9785

Table 4. 4x4 Transition Matrix.

The probabilities on the *principal diagonal* are all close to 1. This shows the tendency for the bio bot to remain in its current state between time steps. Also, 2 of the probabilities in the matrix are 0. This implies that the probability of transitioning from state 2 to 3 or from 3 to 2 is 0. From the description of the states in the dataset, this means that the bio bot doesn't change abruptly from Clockwise to Counter Clockwise motion or vice-versa according to the HMM.

4.4. Missing Data Estimation

Linear interpolation does produce good results for the classes 0 and 1 (classes with more data) but it fails to recognize classes 2 nd 3. It's because of the assumption that the data variability will be linear in nature. The F1 score observed was 0.4563.

Forward Algorithm based approach gives far better results. We observe a macro F1 score of 0.9281. This is similar to the results obtained by using the reduced set of data, showing the benefits of this approach over linear interpolation, curve fitting and other similar methods.

Class	Precision	Recall	F1 Score
0	0.8242	0.9671	0.9049
1	0.9676	0.8893	0.9438
2	0.9318	0.9152	0.9177
3	0.9410	0.9623	0.9518

Table 5. HMM (75-25) Forward Algorithm : Average precision = **0.9162**, Average recall = **0.9334**, F1 Score = **0.9247**.

4.5. Summary

It is observed that the accuracy and F1 scores improved after using HMM. This is because HMM includes temporal analysis of the data and time-series data is hard to model using only theoretical distribution functions such as KNN. Also, the structure of the HMM allowed us to fill in missing data points and retain high accuracy levels, which was not possible with a KNN.

Looking at the performance for individual classes, since there are more training samples for class 0 and class 1, we get higher accuracy for these two classes. Class 2 and Class 3 suffer because of less training samples and considerable overlap.

5. Conclusion

Both classifiers struggled to differentiate between Clockwise and Counter-Clockwise Motion of the bio-bots mainly because of the significant overlap between the feature vectors describing these classes. But with HMM, there was significant increase in accuracy for these two classes. HMM takes into account the time dependencies, thus providing a better anaylsis of the data.It was also able to identify the fact that transitioning from a clockwise movement to a counter clockwise movement has to happen through the stop state.

References

[1] Alper Bozkurt, Edgar Lobaton, Mihail Sichitiu, "A Biobotic Distributed Sensor Network for Under-Rubble Search and Rescue", Computer, vol. 49, pp. 38-46, 2016, ISSN 0018-9162.

[2] https://en.wikipedia.org/wiki/Viterbi_algorithm 1

[3] MATLAB Statistics and Machine Learning Toolbox 2

[4] A Blog explaining various Machine Learning Algorithms. KNN Detailed Explanation 1, 3

[5] http://www.csee.umbc.edu/tinoosh/cmpe650/slides/K_Nearest_Neighbor_Algorithm.pdf 2

[6] M. Sokolova and G. Laplame, "A Systematic Analysis of Performance Measures for Classification Tasks", Information Processing and Management 45, 2009. 2