Posture Reconstruction

Aayush Gupta
2017125
aayush17125@iiitd.ac.in

Bhavya Srivastava
2017037
bhavya17037@iiitd.ac.in

Raghav Gupta
2017178
raghav17178@iiitd.ac.in

Abstract

This research looks at various techniques for the classification of univariate posture reconstruction data. The dataset used is freely available on Kaggle. We have tried to convert the problem of classifying time series data into a standard classification problem. The techniques used for classification include SVM, Linear Regression, Logistic Regression, Neural Networks, and K-NN. We propose the use of K-NN with k=1(Number of neighbors) to classify the data. Based on the results of the experiments, K-NN achieves an accuracy of 99.61% on the test set and 100% on the training set. Therefore classification methods like K-NN can be used to classify the data without the need for accounting its time-series nature.

1. Introduction

Time series sequences are often encountered in classification and regression tasks today. Some examples of time-series data include prediction of stock prices, weather prediction, sales prediction. Time-series are divided into two categories. Univariate time series data depends on a single variable, while multivariate data has two independent variables. Univariate problem is generally used for prediction of the next number in the sequence, whereas multivariate data is primarily used for a classification task. The goal of this paper is to use univariate data for the classification task. The dataset chosen is posture reconstruction that includes data about the positioning of various sensors around the body. Our classification problem is to identify the posture of a person using information from these sensors at a particular time. The proposed method for this task is the use of a K-NN classifier with k=1(Number of neighbors). This method achieves an accuracy of 99.6%, which is comparable to the accuracy achieved by other state-of-the-art solutions that are much more complex.

1.1. Dataset

The dataset has 164860 instances, and eight attributes (including the output class). Thus, there are 7 features:

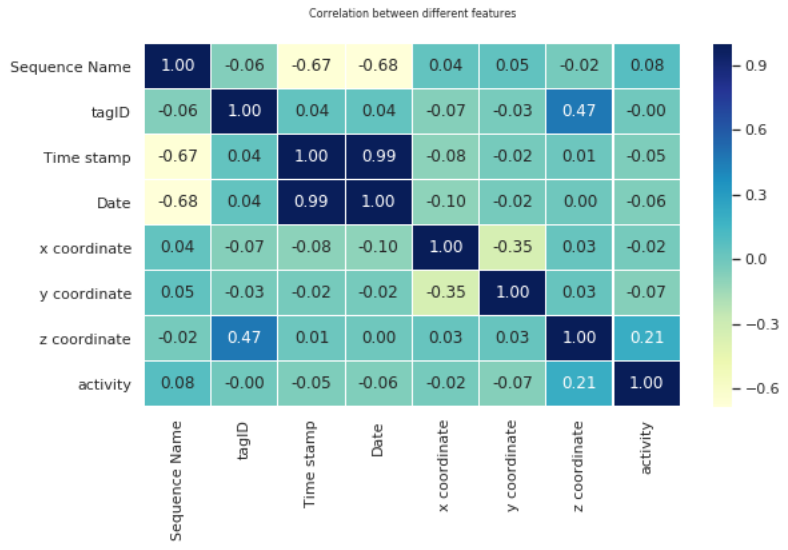


Figure 1. Heatmap

a) Sequence Name A01, A02, A03, A04, A05, B01, B02, B03, B04, B05, C01, C02, C03, C04, C05, D01, D02, D03, D04, D05, E01, E02, E03, E04, E05 (Nominal) - A, B, C, D, E = 5 people.
b) Tag identificator 010-000-024-033,020-000-033-111,020-00 0-032-221,010-000-030-096 (Nominal) - ANKLE_LEFT = 010-000-024-033 - ANKLE_RIGHT = 010-000-030-096 - CHEST = 020-000-033-111 - BELT = 020-000-032-221
c) Timestamp (Numeric) – All unique
d) Date FORMAT = dd.MM.yyyy HH:mm:ss:SSS (Date)
e) X coordinate of the tag (Numeric) f) Y coordinate of the tag (Numeric)
g) Z coordinate of the tag (Numeric)

The number of output classes is 11, which are as follows:
Walking, falling, lying down, lying, sitting down, sitting, standing up from lying, on all fours, sitting on the ground, standing up from sitting, standing up from sitting on the ground.

2. Methodology

2.1. Dataset Cleaning and Normalisation

The dataset contains many categorical fields like Date, sequence name, Tag identifier. We have used label encoders to convert these fields into numerical values and then applied normalization. The dataset is complete, and there are no missing values in it. First, we have encoded the features of the dataset, which do not have a numerical value, and are considered as objects by numpy. As a result, the features: Sequence Name, TagID and Date were converted into numerical values. In addition to this, the output field was encoded too.
After the encoding process was complete, we proceeded with Normalizing the data. To normalize each column, we mapped the data from [-1,1] by subtracting mean and dividing by standard deviation.
We used correlation to determine if two or more classes collides with each other. Based on this, it can be seen that no two classes collide with each other. Also, the conclusion can be drawn that the Z coordinate is the most co-related feature for the label activity. The heatmap is given in Figure 1. In addition to the heatmap, the feature importance was calculated using the functions of decision trees of Sklearn library, which is plotted in figure 2. The Time Stamp feature has low value. Therefore elucidating the impression that Time stamp is not at all important for the model. Hence, we have omitted the Time stamp feature from our dataset. In addition to this, the date feature which is actually time and date is the most important feature for the model. The outliers were calculated using the zscore method. There were very fewer outliers (around 100) which were cleaned before training the data. To further reduce the effect of outliers we are using RMSLE methods to calculate error.

2.2. Classification Techniques

We have used various classification techniques that model the time-series data as a standard classification problem either by normalizing the date variable or by altogether dropping it. Dropping the date variable doesn't have much effect on K-NN, but it reduces the accuracy of other models. The classification algorithms used include SVM, Logistic Regression, Random Forests, Decision Trees, Ridge and Lasso, K-NN, MLP., ElasticNet, Gradient Boosting Classifier.

2.3. Evaluation

For comparing the performance of different techniques, we have used F1-scores, accuracy, precision, confusion matrix, and generated the classification reports for all the algorithms. Accuracy is the most common evaluation metric used to measure the performance of a machine learning model. It is calculated by dividing the number of correctly classified samples by the total number of samples in the concerned dataset.

$$Accuracy = (TP + TN)/(FP + FN + TP + TN)$$

In the above formula, TP, TN, FP, FN are the number of samples which were classified as True Positive, True Negative, False Positive and False Negative respectively. Another popular evaluation metric we have used is Precision. For calculating precision of each of the 11 classes in our dataset, we have first calculated the total number of true positive predictions using our various models, which was then divided by the sum of true positives and false positives samples.

$$Precision = TP/(TP + TN)$$

Similarly, we calculated the recall of each of the 11 classes for each model used. The only difference between precision and recall is that in recall, the false negative samples are considered, instead of the false positive ones.

$$Recall = TP/(TP + FN)$$

Using precision and recall, we calculated the F1 score for each of the 11 classes. F1 score is calculated as 2 times the product of precision and recall, divided by the sum of precision and recall.

$$F1Score = 2 * precision * recall/(precision + recall)$$

To generate the statistics showing the precision, recall and F1 score, we have used the classification_report function of SKlearn. In addition to the above mentioned evaluation metrics, confusion matrices were generated to get an idea about the performance of the model. In a confusion matrix, the entry at position (i, j) represents the number of samples which were actually belonging to class "i", but were predicted to be in class "j" by our model. Generally, if a model performs well, the diagonal entries of it's confusion matrix would have relatively higher values than the rest of the entries. ROC curves have also been printed to identify the prediction of an algorithm for different classes. The ROC curve tells us how much the model is capable of differentiating between the classes. It is basically the graph of the True Positive Rate (Y axis) vs The False Positive Rate (X axis). The true positive rate is also called the Sensitivity of the model, while the False Positive rate is 1 - Specificity of the model. The AUC (Area under the Curve) gives us a measure of the degree of separability. The higher the AUC is, the better the model able to distinguish between the 11 classes, hence the performance is higher.

3. Related Work

A paper called Classification of posture reconstruction with uni-variate time series data type[1] presented in the

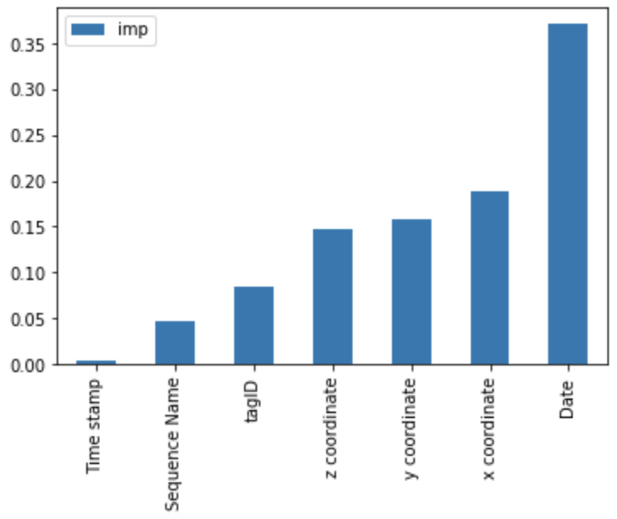
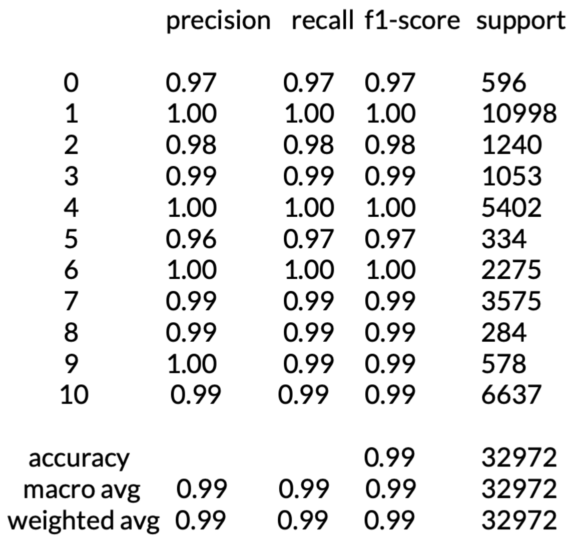


Figure 2. Feature Importance

2018 international conference on SIET proposed the classification method. K-NN should be used for this problem. This is because of it's simplicity. K-NN is also able to achieve error results compared to methods that are much more complex.
The data is first fed to the model. The next step is to determine the closest sequence based on Timestamp. After we get the successive value for the data, we compute the distance to every neighbour using the euclidean distance. Then we vote the class category based on k value and then we rank the resultant from the nearest neighbour to the most farthest neighbour.
There are two rules in classifying the new unlabeled data.Those are majority voting based on k value and similarity score summing. The class or category gets one vote for each instance of that class in the neighborhood sample set k. Next is that the new data samples are classified into the class with the largest number of votes. In similarity scores summing,each class gets indistinguishable score from the summation of the comparability level of the class instance in the environmental sample set. Then, the sample of new data is classified into the class with the highest degree of the similarity.
The authors determined that the number of k should be five based on similarity measurement by Euclidean distance. They have used a time series column for calculating the nearest neighbors. They achieved an accuracy of .995 using this technique.
The research also still could't measure how high the accuracy degree of the k-NN in classification of posture reconstruction. But regardless of the accuracy degree, k-NN is declared to be able to solve uni variate time series classification.

Model	Accuracy
Gradient Boosting	92.1%
Random Forest	92%
Decision Trees	85%
KNN	99.44%
MLP	77.19%
Logistic Regression	41%
Ridge Classifier	40.22%
SGD Classifier	28.99%
LassoCV	5.86%
BayesianRidge	4.81%
Lars	4.81%
Lasso	4.76%

Table 1. Accuracies for different models



```
              precision  recall  f1-score  support

   0            0.97      0.97    0.97      596
   1            1.00      1.00    1.00      10998
   2            0.98      0.98    0.98      1240
   3            0.99      0.99    0.99      1053
   4            1.00      1.00    1.00      5402
   5            0.96      0.97    0.97      334
   6            1.00      1.00    1.00      2275
   7            0.99      0.99    0.99      3575
   8            0.99      0.99    0.99      284
   9            1.00      0.99    0.99      578
  10            0.99      0.99    0.99      6637

    accuracy                      0.99      32972
   macro avg     0.99      0.99   0.99      32972
weighted avg     0.99      0.99   0.99      32972
```

Figure 3. Classification Report for K-NN

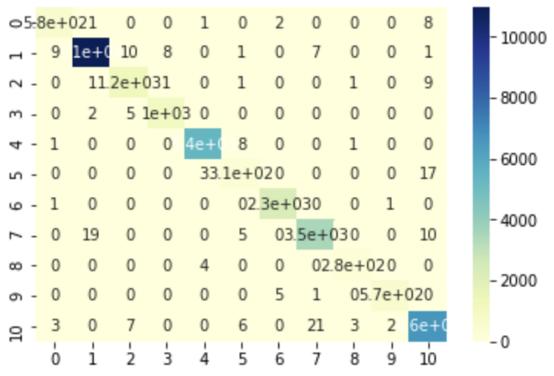


Figure 4. Confusion matrix for K-NN

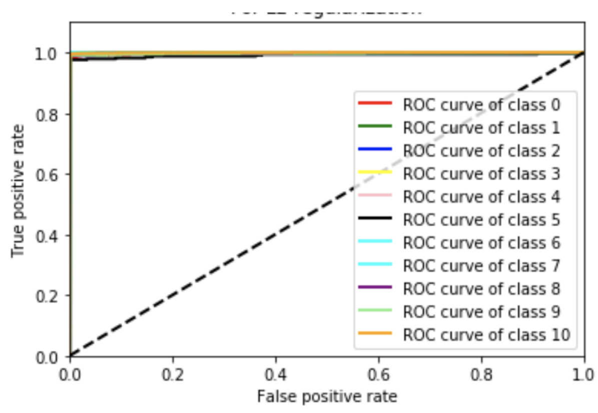


Figure 5. ROC curve for K-NN

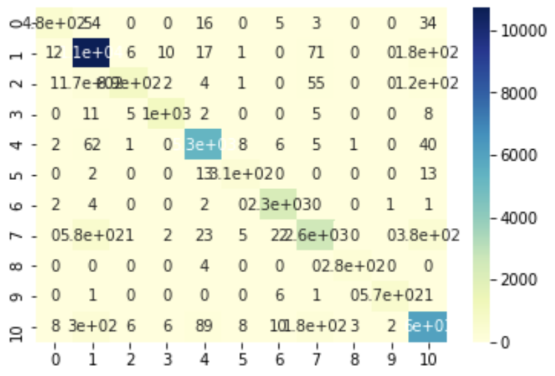
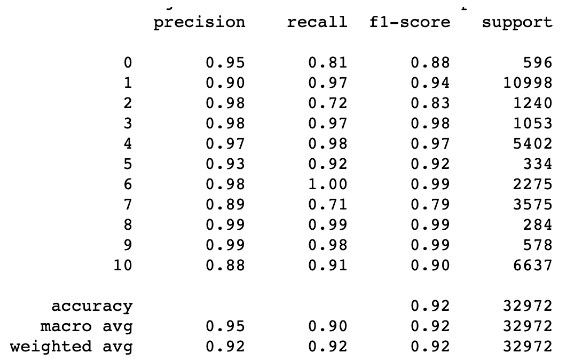


Figure 6. Confusion matrix for Gradient Boosting



```
              precision    recall  f1-score   support

           0       0.95      0.81      0.88       596
           1       0.90      0.97      0.94     10998
           2       0.98      0.72      0.83      1240
           3       0.98      0.97      0.98      1053
           4       0.97      0.98      0.97      5402
           5       0.93      0.92      0.92       334
           6       0.98      1.00      0.99      2275
           7       0.89      0.71      0.79      3575
           8       0.99      0.99      0.99       284
           9       0.99      0.98      0.99       578
          10       0.88      0.91      0.90      6637

    accuracy                           0.92     32972
   macro avg       0.95      0.90      0.92     32972
weighted avg       0.92      0.92      0.92     32972
```

Figure 7. Classification report for Gradient Boosting

4. Results

We achieved a maximum accuracy of 99.44% using the K-NN models with k=5. Increasing or lowering the value of k resulted in lowering of the accuracy and precision. The reason K-NN works so well in this particular classification problem, is that it automatically performs uni-variate time-series analysis of the dataset while training. The classification report for the K-NN model is given in Figure 3, and the confusion matrix is given in Figure 4. This is the best solution we could come up with. There were some other models which we trained on the dataset, but they could not perform as well as the K-NN. The ROC curve for K-NN is can be seen in Figure 5. The curves for all classes is almost a straight line at y=1. The reason for this is the high accuracy achieved using this model. Some models gave downright disappointing results. For example, the Bayesian Ridge, LassoCV, Lasso and Lars classifiers gave test accuracy 4.81, 5.86, 4.76 and 4.81% respectively. We had expected the MLP classifier (Neural Network) to give good results, but it was not able to perform up to the mark. After tuning the number of hidden layers, along with the number of neurons in each layer, it could give us a maximum accuracy of 77.19%. After MLP, we built a Logistic Regression classifier, but it soon became evident that it was no better than our Neural Network. In fact, it performed quite badly, and gave a disappointing 41% accuracy. Some other classifiers we used which did not perform very good were the Ridge and SGD Classifier. They gave an accuracy of 40.22% and 28.99% on the test set respectively.

One of the models which came very close to K-NN in terms of performance was the Gradient Boost Classifier. This classifier is basically an ensemble algorithm involving decision trees. It takes several decision trees, and combines them to form a single pruned tree. At first, when we kept the number of iterations as 100, we got an accuracy of 92%, which was pretty impressive. But we soon found out that as we keep increasing the number of iterations, this model tends to overfit on the training data. After setting the number of iterations to 500, we observed that the model gave a training accuracy larger than 99%, while the testing accuracy fell dramatically. We conducted a couple of more tests, and concluded that the training accuracy went up and testing accuracy went down as we increased the number of iterations. Thus, even with the best possible tuning of the parameters of the Gradient Boost Classifier, it gave a maximum of 92% accuracy.

The classification report and the confusion matrix for Gradient Boost classifier are present in figure 5 and 6. From the figure we can see that gradient boosting tends to perform classification of some classes better than the others. Some classes have F1 score of 0.99, whereas one of the class has an F1 score of 0.79. The reason for this is probably boosting, that focuses on some classes more than others based on intermediate accuracy results. The average accuracy of all these classes combine to give an accuracy of 92%. The decision tree classifier also gave an impressive accuracy of 85%, and naturally, the Random Forest classifier gave an accuracy which was a little better: 92%.

5. Conclusion

Gradient boosting gave an accuracy of 92.1% which is decent enough. But the problem with Gradient boosting is that it overfits very quickly. The default iterations were 100. Increasing the iterations to 500 resulted in training accuracy of 98.5% while the testing accuracy decreased sharply. Based on the results, we conclude that K-NN is the best algorithm for this problem. In this case, we used k=5. We can, therefore, infer that the K-NN classifier can be used for uni-variate time series problems, even without account for the time-series nature of the data, to achieve high accuracy. The accuracy of other models is given in Table 1.

References

[1] Rikatsih, N., Supianto, A.A. (2018). *Classification of Posture Reconstruction with Univariate Time Series Data Type*, 2018 International Conference on Sustainable Information Engineering and Technology (SIET), 322-325.