Team 3 Proj-C: Applying Bidirectional LSTM to Terrain Identification Problem

Shilpa Kancharla
Department of Computer Science
North Carolina State University
Raleigh, NC
skancha@ncsu.edu

Haitao Jiang
Department of Statistics
North Carolina State University
Raleigh, NC
hjiang24@ncsu.edu

I. Methodology

We approached this dataset by observing the measurements taken from the accelerometer and gyroscope, as well as the timestamp that each measurement was taken at. Since the sampling rates for the input data are at 40 Hz while the sampling rate for the labels is 10 Hz, we extrapolate 4 labels to match the sampling rate of the input data. We consider this up-sampling. We chose to do this instead of down-sampling because we believed it would generate more training instances for our model. In the preliminary data preprocessing stage, we created a dataframe of the inertial measurements and a dataframe of the extrapolated labels. To clarify, this extrapolated dataframe is roughly four times larger than the original length of the associated labels. We note the length of the input inertial data and the extrapolated label data, and we subtract the length of the extrapolated label dataframe from the input inertial dataframe. Essentially, extra data points at the end of the input files are discarded. Moreover, we use `StandardScaler` from `sklearn.preprocessing` to scale the values of the input data to make it robust to outliers and reduce bias.

Since we are dealing with sequential data, we attempted to model this problem as a time series classification problem. In order to convert our given sequential data into a time series point that could be fed into a model which handles sequential input data, we used the the method of sliding windows [1]. An illustration of this technique is shown in Figure (1). We reshape the information we have by fixed windows that will provide the model with the most complete information possible at a given time point from the recent past in order to achieve an accurate prediction. We utilized each subject and trial file given in the training set as an input instance as opposed to merging all the training data together. We employed overlapping windows to capture the continuous nature of the data. After this, we concatenated the windowed data from each file. The final shape of the dataset is `(total windows, 30, 6)` for a single input file. We use a window size of 30, step size of 1, and 6 measurement attributes to generate this shape.

We understood that a key characteristic of this dataset was that it is imbalanced. In order to address this, we assign

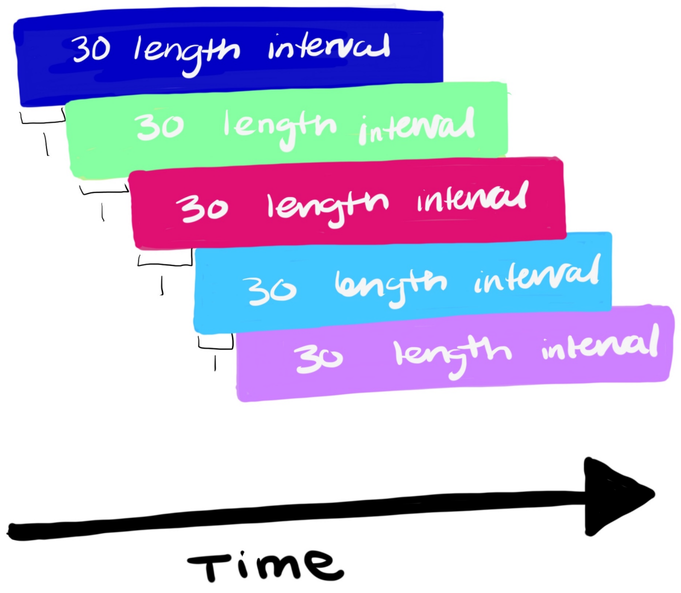

Fig. 1. Sliding window technique illustrated with an interval length of 30 and step size of 1.

different weights to each class. In the case of this dataset, we found that the number of observations labeled as class `0` far exceeded observations labeled as `1`, `2`, or `3`. For our analysis, we chose to use the weight calculated by the `class_weight` in the `sklearn` package. That is, for each observation of class `0`, we multiply all the changes in the entire network by a small number because there are a relatively rich number of class `0` observations. As for other classes, their multipliers are greater than `1` so the cost of misclassification with in these group costs more.

Recurrent neural networks (RNNs) were developed for sequential inputs. They exhibit temporal dynamic behavior because their internal state has the ability to keep some information from the previous observations. Long short-term memory (LSTM) [2] is one of the most popular neural network architectures used in the field of deep learning, composed of an input gate, an output gate, and a forget gate in each unit. In acceleration and gyroscopic data regarding human activity recognition, an RNN model with a LSTM layer and dense layer was used to perform the classification. The model learns to extract features from sequences of observations and how to map the internal features to different human activity types [2]. Bidirectional LSTMs are an extension of regular LSTMs. While a standard LSTM looks at only previous context, a bidirectional LSTM considers data in both directions, forward and reverse. Bidirectional LSTMs can provide additional context to the network and can result in more efficient learning of the problem given. We believe that we could use a such structure to solve a similar problem with terrain identification instead of different activity types.

II. MODEL TRAINING AND SELECTION

A. Model Training

For training, we used input files except `subject 002_01` and `subject 001_08`, which are reserved as validation set, and `subject 006_03` and `subject 007_01`, which are reserved for our test set in order to evaluation the performance of our bidirectional LSTM. Upon applying the sliding window technique described in our methodology, the shape of training input was `(1126710, 30, 6)` and and the training label shape was `(1126710, 1`. The shape of our validation input was `(122696, 30, 6` and the validation output had a shape of `(122696, 1`. Finally, the shape of our test set was `(91260, 30, 6` and the test output had a shape of `(91260, 1`. We used the `sklearn` library from Python, and specifically used the `class_weight` function to assign 0.32587, 5.86071, 4.42792, and 1.869729 to class `0, 1, 2,` and `3` respectively. Additionally, we used one-hot encoding on the label values for the training, validation, and test sets. The `OneHotEncoder` function from `sklearn.preprocessing` was implemented to accomplish this.

The architecture of our network is summarized in Table (I). It contains three layers: one bidirectional LSTM layer, and two fully-connected dense layers with 125 and 4 nodes. Note that we applied the dropout technique after the bidirectional LSTM layer. As for the activation functions, we choose the ReLU function in the first dense layer, and the softmax function in the output layer. In this model, there are $163,879$ parameters in total to be trained. To build our model, we used the `Keras` library. Specifically, we used the `Sequential`, `Bidirectional`, `LSTM`, `Dropout`, and `Dense` functions. To compile our model, we employed categorical cross-entropy as our loss function and Adam as our optimizer. We also used the metrics of loss, accuracy, precision, recall, and the F1-score at every epoch during training to monitor phenomena such as overfitting.

B. Model Selection

Initially, our model had three hyperparameters to be tuned. They are the dropout rates, L1 regularization multiplier, and L2 regularization multiplier. Based on preliminary tests, we decided to use [0.1, 0.5, 0.9] as dropout rates, and [$2^{(-5:-8)}$] as multipliers for regularizers. For the sake of training time, we set the tuning epochs as 8 (however, even with using a lower

TABLE I
STRUCTURE OF BIDIRECTIONAL LSTM MODEL

Layer (type)	Output Shape	Parameter No.
Bidirectional	(None, 250)	132000
Dropout	(None, 250)	0
Dense	(None, 125)	31375
Dense	(None, 4)	504

epoch value, the hyperparameter search took approximately 22 hours). There are 48 different combinations given grids above, and we will use following metric to decide which of them is the best:

$$L(r, l_1, l_2) = \frac{1}{5} \sum_{i=n-5}^{n} F_1(r, l_1, l_2),$$

where the j^{th} of $F_1()$ function is the j^{th} F1 score calculated from the training set. So literally, we are using the mean of the last five f1 scores as the indicator because we have some preliminary experience of a bumpy curve. After finding the ideal combination given the parameter grid, we can first do some visualized validation by plotting the predicted label sequence vs actual label sequence. A final model using these parameters with more (15) epochs will be trained using all training data, then we can predict results for the testing files using our final model. The optimal hyperparameters found through this search are as follows: the dropout rate is 0.1, the L1-regularization multiplier is 2^{-6} and the L2-regularization multiplier is 2^{-8}. Upon applying this regularization to our model, we found our loss and accuracy on the test data to be 0.6662 and 76%, respectively. A summary of the classification metrics can be found in Table (II).

TABLE II
CLASSIFICATION REPORT FOR BIDIRECTIONAL LSTM WITH L1 AND L2 REGULARIZATION

Label	Precision	Recall	F1-Score
0	0.56	0.69	0.62
1	0.68	0.72	0.70
2	0.69	0.70	0.69
3	0.63	0.48	0.54

Upon further investigation of why adding terms of L1 and L2 regularization may depreciate the performance of RNNs, we found that L1/L2 regularizing the RNN cells also compromises the cells' ability to learn and retain information through time. While using L1 and/or L2 penalties, the recurrent weights can help with exploding gradients, this approach can also limit the model to a single point attractor at the origin, where any information inserted into the model dies out exponentially fast. This prevents the model from learning efficiently [3]. We experimented to see how our model would perform with no L1/L2 penalties, and increased the dropout rate to 0.5 from 0.1 since this is the only kind of regularization we are using. We elaborate on the results of this decision in the next section.

III. EVALUATION

In our final model, the only regularization we applied was a dropout rate of 0.5. We assessed our model performance after 10 epochs by examining the plots of loss, accuracy, precision, recall, and F1-score (shown in Figure (2), Figure (3), Figure (4), Figure (5), and Figure (6) respectively) on the training and validation test sets. Specifically, the loss and accuracy plots of our training and validation sets can be seen in Figure (2) and Figure (3). After 10 epochs of our bidirectional LSTM model, it was found that the training set had a loss of 0.0246 and an accuracy of 98.4%. Moreover, the validation set had a loss of 3.4266 and an accuracy of 62.26%. Despite our validation set receiving a higher loss and lower accuracy in comparison to our training loss and accuracy, we also perform an evaluation on our withheld test data.

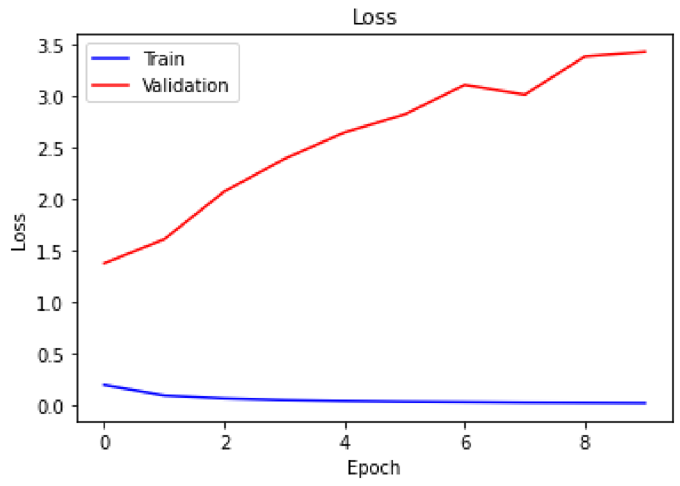

Fig. 2. Loss on training and validation sets

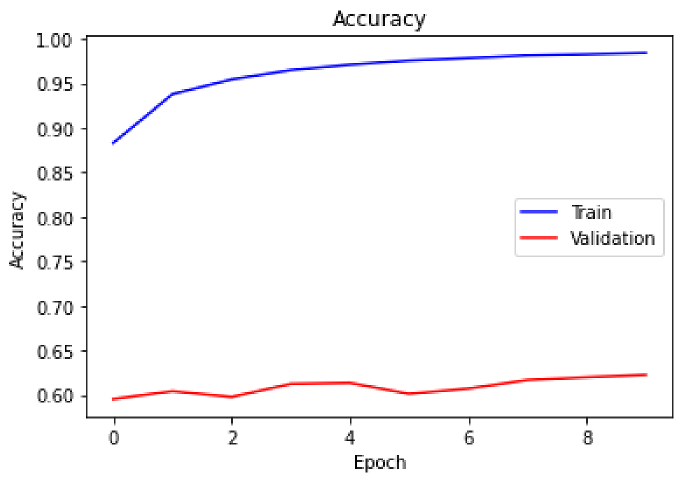

Fig. 3. Accuracy on training and validation sets

The classification report for the model performance on the test data is given below in Figure (7). While we expected that the accuracy and other classification metrics for the validation

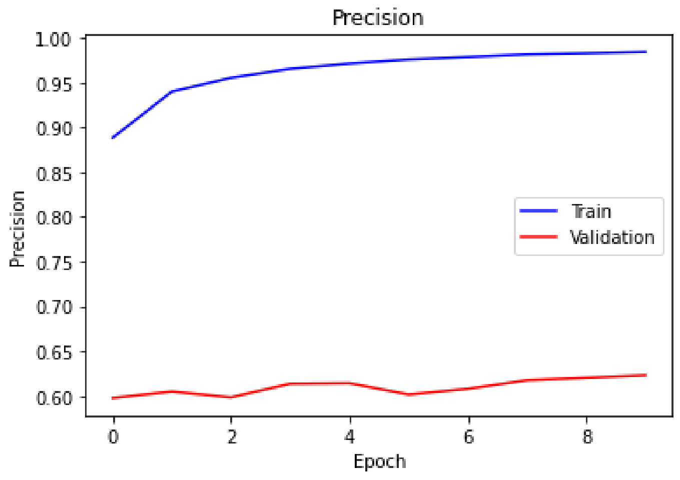

Fig. 4. Precision on training and validation sets

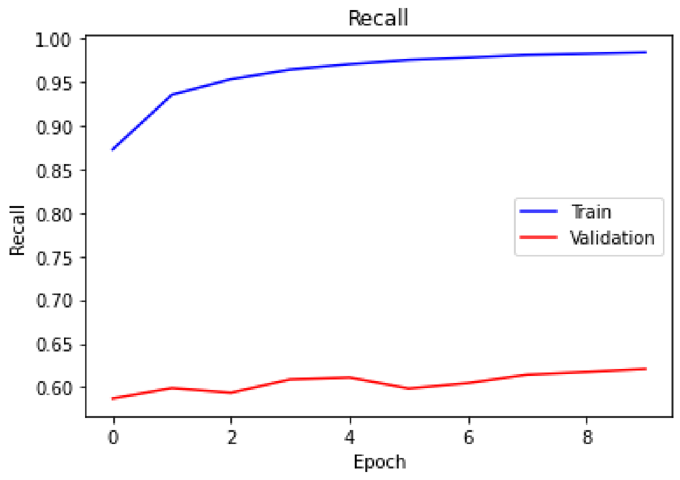

Fig. 5. Recall on training and validation sets

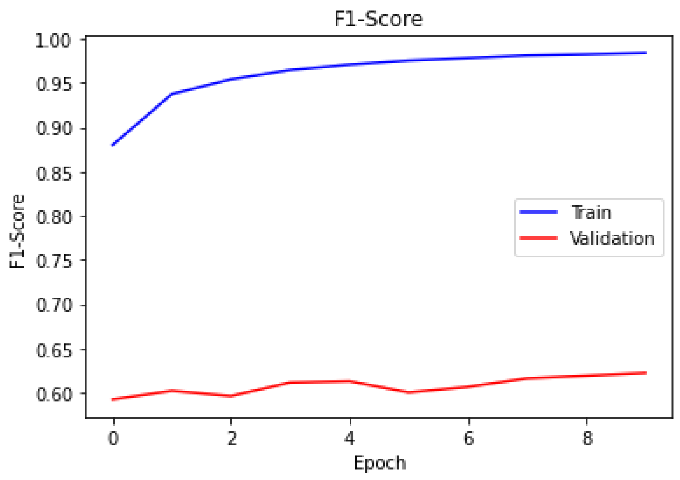

Fig. 6. F1-score on training and validation sets

set would be lower than that of the training set, we were intrigued by how much higher the classification metrics for the test data were than for the validation data. We are curious to investigate if the validation set classification metrics we received were due to the nature of the data of the particular two subjects used to create the the validation set, or if there were some issues with overfitting in our trained model. However, the significantly higher classification metrics for our test set indicate against overfitting.

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96     65623
           1       0.84      0.97      0.90      2892
           2       0.91      0.99      0.95      3745
           3       0.91      0.81      0.86     19000

    accuracy                           0.93     91260
   macro avg       0.90      0.93      0.92     91260
weighted avg       0.93      0.93      0.93     91260
```

Fig. 7. Classification report for the withheld test data

From Figure (7), we found evaluation metrics of precision, recall, and accuracy of our final model. In particular, it was seen that the precision and recall, as well as the F1-score, were higher for labels 0 and -1 compared to labels 1 and 3. This contrast may have been due to the use of weighting as our model training technique, or just the nature of the given classification problem.

We wanted to visualize our results on the withheld test dataset in an additional way. We compare a subsection of the actual output data (shown in Figure (8)) to a subsection of the predicted output (shown in Figure (9)).

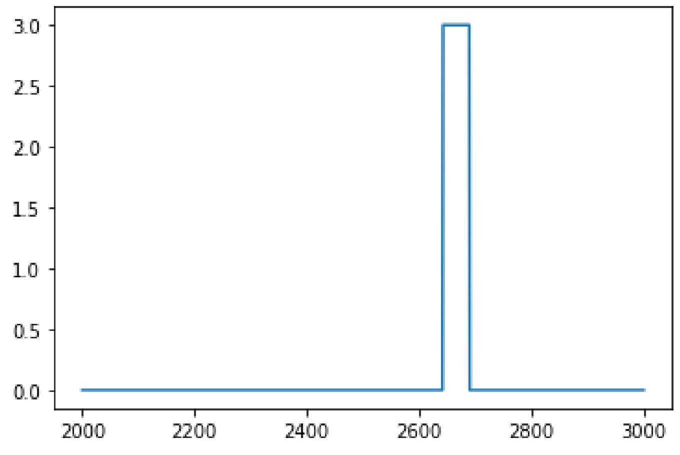

Fig. 8. Subsection of actual data.

The length of our model output is still roughly equivalent to the length of the input data. As a post-processing step, we look at every four outputs and take the most frequently occurring item in that subset of the output using the `mode` function from `scipy`. The x-axis in Figure (8)) are the indices of the dataframe of the actual data from `subject 006_03`

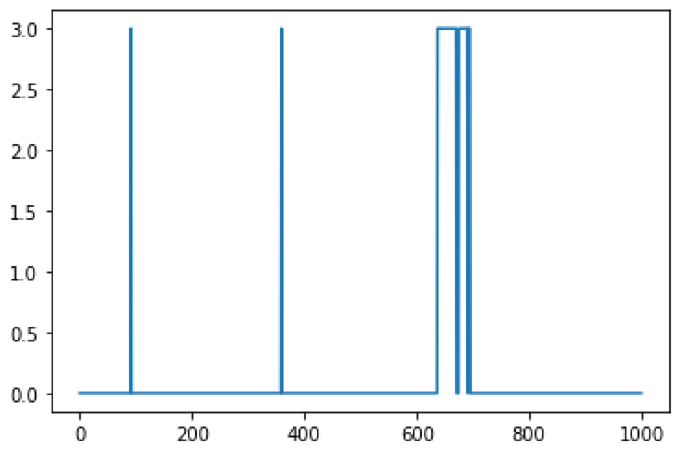

Fig. 9. Subsection of predicted data that corresponds to the subsection of actual data in Figure (8).

indexed from 2000 to 3000 using the `.iloc` command from the `Pandas` framework. The x-axis in Figure (9)) differs as we index a `numpy` array structure from 2000 to 3000. The y-axis is supposed to represent the labels of 0, 1, 2, and 3. Nonetheless, these two plots show the same subsection of data. When we compare these two figures, we see that with the exception of a few points which are zero that are incorrectly predicted as 3, the majority of the predicted values are consistent with the actual labels. Lastly, we provide a final plot of the predictions for Subject 9 which are our model predicted in Figure (10)). This figure shows a subsection of the predictions indexed from 2000 to 3000.

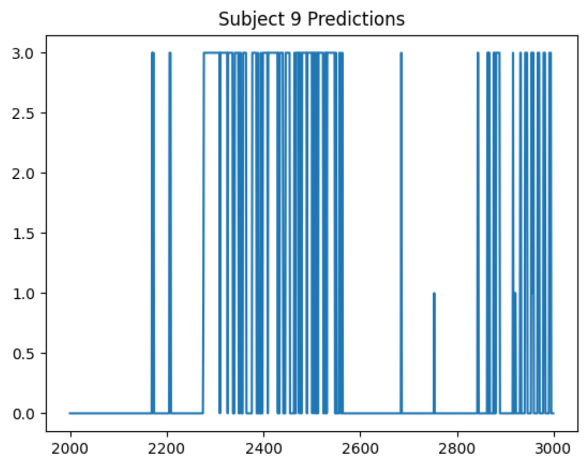

Fig. 10. Subsection of predicted data for Subject 9.

References

[1] Sefidmazgi, A., Sefidmazgi, M. (2019). *Causality Analysis in Climate Time Series Using Windowed Regression* (ResearchGate).

[2] Gers, F. (2001). *Long short-term memory in recurrent neural networks* (Doctoral dissertation, Verlag nicht ermittelbar).

[3] Pascanu, R., Mikolov T., Bengio Y. (2013). *On the difficulty of training recurrent neural networks* (arXiv.org).