Support Vector Machine and Long Short-Term Memory Network Classification of Time Series Data

John Bannan
Department of Electrical and Computer Engineering, Temple University
smith@temple.edu

Introduction: This report will discuss the results of applying two machine learning techniques, Support Vector Machines (SVM), and Long Short-Term Memory Networks. The task in this project was to apply a non-neural and a neural network approach to classifying time series data and classify events occurring in the data. The dataset provided included a training, development, and evaluation data set which included raw amplitude data for the signal and the time of each sample as well as associated labels for each event class and duration. The eval file did not have labels associated with it and was tested blindly. The training data set included 10 folders with 1000 data signal files, the development data had 2 folders with 1000 signal files, and finally the evaluation file contained 2 folders of 1000 signal files. Moreover, looking at the data we can see the data set contains 5 classes (0,1, 2, 3, 4) four signify an event has occurred (1- 4).

Preprocessing: To make the signal files readable for the models, there was some preprocessing that had to be done to the data. First some preprocessing was done to the label data as it was contained in the comments of each file and only the duration and start and stop time of the data was given. Thus, to correctly label each sample, an array of the labels for each sample was created by taking the start and stop times for each event class. A sliding window approach was then applied to the data, with length of 11 samples. The window would slide across all the data of a given file, then the amplitude of the samples in the window would be averaged together and a feature vector would be created. The samples in the window were then normalized between a range of 0 and 1. This was to ensure that all windows of every signal were normalized to the sample range and uniform between files. This average would then be classed based on the middle data point. I had considered using majority vote of the samples in the window to classify the data; however, I used the former method as it was simpler and faster.

Support Vector Machine with Stochastic Gradient Descent Optimizer (SVM): For the classification of labels for the non-neural network approach I chose to use a Support Vector Machine with a Stochastic Gradient Descent optimizer. The SGD sklearn library with a linear SVM for a couple of reasons. One reason was because of the sklearn's limitations for certain algorithms in using online learning. Unlike my approach with the LTSM, which I used sequential learning, in the SVM I used incremental learning. Online learning or incremental learning updates the model with new information as it is presented, this is commonly used when new data is expected to be streaming into the system. The sklearn library does not provide an easy way to update the parameters of its systems, such as RandomForest and the non-SGD SVM, once the data has been fitted to the system. In fact, fitting any data using the. fit command, according to the sklearn documentation, will overwrite the previous model, so training the data files with the sklearn library, at least, would be the same as if I had just fitted the model to the last data file trained. There is a warm_start attribute in the Sklearn library, that will save some parameters of your previous model, but if the data is too different the parameters of the previous model will be overwritten. The other solution that I could come up with was to have the entire training set trained at once, but this was obviously a sub optimal solution and was not feasible. So, for this case a model was chosen that could be updated as I trained it on more data sets. The model was updated using the partial_fit command, which was a command that was not available to the other sklearn classifiers such as RandomForest, which allowed the SVM to use SGD to optimize and update its parameters.

A SVM utilizes support vectors to classify data. Support vectors machines work by taking data points and creating a hyper plane between the classes. We then maximize the margin of the classifier to find the best hyperplane. The hyper plane is then used to classify data points. We can maximize this margin by using a loss function, in this case the hinge loss function, seen below in Figure 1.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Figure 1. Hinge Loss Function

We can see from this loss function, the cost remains zero if the predicted class and the actual class are the same sign, while if they are different there is a loss. These partial derivatives of the loss function with respect to the weights can then be taken and can be updated as the model is trained. This SVM loss function is typically used in binary classification, however, the model can be used for multiclass problems. The error rate was computed using computing 1 - accuracy_score function in the sklearn library.

Random Forest (RNF): After attempting to use SVM, I tried to use Random Forest (RNF) with the warm start attribute set to true, I figured that the data should not vary too much between data sets and the results from the SVM, were very poor, so RNF may be a better option. The preprocessing was the same as the SVM. I used the Sklearn RNF forest library to update the model. Random Forest uses a multitude of decision trees in order to come to a conclusion. In my algorithms since, fit would train the model, I instead updated the number of estimators each time in order to account for new data.

Long Short-Term Memory Network (LTSM): LTSM's are a type of recurrent neural network (RNN) that contain memory cells that each contain an input gate, forget gate and an output gate. This allows the memory cell to store its past calculations and information. A figure of the arichtecutre can be seen in Figure 2.

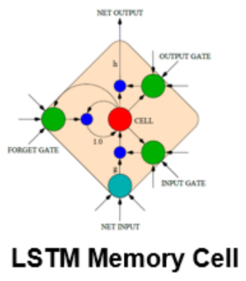
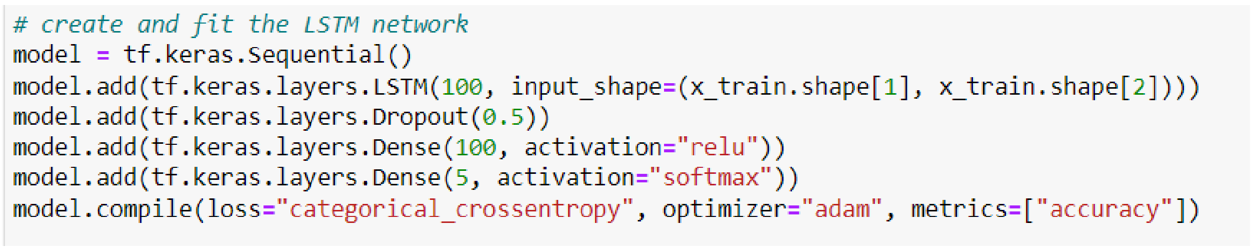


Figure 2. LSTM Memory Cell

The windowing and preprocessing of the data for the LTSM was the same as the other classifiers. Using the keras library, I was able to create sequential layers to the LSTM model. The model can be seen in Figure 3.



```
# create and fit the LSTM network
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(100, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(100, activation="relu"))
model.add(tf.keras.layers.Dense(5, activation="softmax"))
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Figure 3. LTSM Model

In Figure 2. There are 4 layers to the model which include an LSTM layer with 100 memory cells, a dropout layer, and two dense layers that condenses the system to the 5 classes. The dropout layer is used to prevent over fitting. The loss function used here categorical_crossentropy, as used in many multiclass models. The activation function is set to “softmax” here for the same reason. The model was updated as new training data was made available.

Results: Using the SVM Data I computed the error for each file and then averaged the error across all files for the data in the table. The data appears to have a very low error rate; however, I attribute this to the zero class, which is very likely to occur. Furthermore, I looked at my output data for this classifier and found that most data files were being labeled as zeros or rather all of them, at least at the few I looked at. This made me very suspicious that maybe all of the files were just classified as class zero and thus the higher error rate. This may be also due to the classification I did on the windows, using the middle point of the labels, instead of majority vote. Ultimately, I do not see the SVM as an effected classifier because of this, despite the low error rate. Both the train and Dev data sets performed about the same with around a 0.075 error rate. However, this also could be because there are so many zeros, that the data is very unbalanced. The RNF data shows almost double error rate compared to SVM, by table 1 alone it would be said that SVM would be a better classifier, however, again due to SVM output files I do not think this is the case. The csv files from the RNF seemed more consistent with the actual files, at least with the small batch that I visually scanned. The LSTM data will be filled in and discussed once completed.

	Data Set Error		
Algorithm	**Train**	**Dev Test**	**Eval**
SVM	0.07463	0.07575	N/A
LSTM	TBD	TBD	N/A
RNF	0.13979	0.14086	N/A

Table 1. Data Set Error Rates

Conclusions: The project was very tough and pushed my computer to its computational limits. Overall, the LTSM, while having low error rates, does not seem to work as intended and after viewing the output files of the some of the LTSM files and them all labeled as zero. The RNF seemed to do a good job, however, that remains to be seen until the eval data is scored.