Predicting and Classifying Drought Severity using LSTMs

Joshua Lemmon
jmlemmon@uwaterloo.ca
20575620
University of Waterloo
Waterloo, ON, Canada

Abstract

Weather events like drought can cause result in damaged crops and potential death. Predicting the occurrence and severity of drought events would allow for preemptive actions to greatly minimize potential damage. Machine learning methods have been shown in the past to be effective weather predictors when performing regression. This paper aims to apply machine learning models to the task of classifying drought severity using time-series data. A baseline Random Forest model is trained to be compared to a Time-Series Forest model and a Long Short Term Memory model to measure how effective time-series analysis is for drought classification. It is found that when comparing models trained on a balanced data set, the LSTM model performs the best in terms of all metrics; accuracy, precision, recall and F1 score.

Introduction

Drought is a major weather condition that is capable of causing heavy damage across the world. The US government reports that since 1980, droughts have caused $249.7 billion in damage (Smith 2020). These damages include crops, livestock and even human death due to intense heat waves. As a result of climate change, droughts have been increasing in both severity and frequency (Gray and Merzdorf 2019).

This paper aims to investigate the feasibility of using machine learning based methods to predict the severity of potential drought events. Given a set of features depicting various weather values such as wind speed an temperature, we want to find a model that can accurately predict whether a drought will happen, as well as how severe the drought will be. This is known in the field as time-series forecasting. If such a forecasting model can be found, it would provide great help in potentially mitigating the negative effects of an imminent drought.

This paper will explore the usage of machine learning based methods in predicting the severity of droughts based on a given feature vector of weather data. The main model that will be trained and evaluated will be a Long Short Term Memory Recurrent Neural Network (LSTM). Due to the temporal nature of weather patterns, it is reasonable to assume that an LSTM model based on time-series analysis would perform well in this prediction task.

The LSTM model will be compared to a baseline time-series forest (TSF) model, as a TSF model has been shown to be an effective method used for multivariate time-series classification (Deng et al. 2013). This comparison will show whether the more sophisticated LSTM model is capable of learning a better representation of the data. Furthermore, the LSTM will also be compared to a basic random forest model. This will be to determine whether the time-series analysis provides a noticeable benefit to the predictability of droughts when compared to a non-temporal model, and if so by how much.

The research question this paper will explore is: Will an LSTM model be able to predict drought severity with a higher F1 score when compared to a temporal TSF model and a non-temporal random forest model?

The LSTM model will be a 2-layer stacked LSTM, with a final softmax layer for use in determining the classification of the input vector. The LSTM model will be built using PyTorch. The TSF model will be built using pyts (Faouzi and Janati 2020), a library that specializes in time-series algorithms. The non-temporal random forest classifier will be built with sklearn. Both ensemble models will consist of 100 individual decision trees.

The data set that will be used is public, and can be found on Kaggle (Minixhofer 2019). The data set consists of time-series data from a number of anonymized counties from across the United States. Each instance of a county contains 90 days of weather data, and the data set contains data from years 2000-2020. Some features that are included in the data set are: precipitation, humidity, temperature and wind speed.

The data is preemptively split into train, validation and test sets. The training set encompasses the years 2000-2009, the validation set encompasses 2010-2011 and the test set encompasses 2012-2020. The data set contains six different possible labels: None, D0, D1, D2, D3, D4 and D5. These correspond to differing levels of drought severity, with None corresponding to no drought and D5 corresponding to the most severe level of drought. The data set is also heavily imbalanced towards samples where no drought is occurring, so precision, recall and F1 score will be used to evaluate

the models instead of pure classification accuracy. Confusion matrices will be maintained for each model to assist in the calculation and visualization of the model results.

Related Work

Weather prediction is a well researched application of time-series forecasting. Amei et al. have used a form of time-series analysis called autoregressive integrated moving average (ARIMA) models to predict the occurrence of large earthquakes (Amei Amei and Ho 2012). This method is similar in scope to the purpose of this paper, but Amei et al. only use an ARIMA model with no machine learning. In the same year, Agrawal et al. use a simple feed-forward Artificial Neural Network (ANN) to perform temperature prediction. Singh et al. perform weather forecasting using a simple Recurrent Neural Network (RNN), and compare its results with a linear SVM and a feed-forward ANN (Singh et al. 2019). Singh et al.'s method is closer to the methodology that this paper follows, however this paper is exploring the use of an LSTM, random forest and VAR model for classification, not regression.

(Khan et al. 2020; Salman et al. 2018; Kaneko, Nakayoshi, and Onomura 2019; Kratzert et al. 2018) all use some form of basic single layered or 2-layer stacked LSTM model in their papers. Khan et al. utilize an LSTM model to predict monthly average temperature and rainfall in the region of Bangladesh (Khan et al. 2020). They show that the LSTM model performs quite well when predicting temperature, and relatively well when predicting rainfall. Salman et al. compare their LSTM model to an ARIMA model on the task of predicting visibility level at an airport. They discover that the LSTM outperforms the ARIMA model in all experiments (Salman et al. 2018). The LSTM model used in both of these papers only consisted of a single layer. Kaneko et al. compare their 2-layer stacked LSTM model with the Japan Meteorology Agency's Meso Scale Model (MSM) and find that their model predicted rainfall more accurately than the MSM model in some cases (Kaneko, Nakayoshi, and Onomura 2019). In a similar vein, Kratzert et al. use a stacked 2-layer LSTM model to predict rainfall-runoff resulting from rain storms(Kratzert et al. 2018). This can be viewed as a parallel to drought prediction, as rainfall-runoff can be used to predict a different natural disaster in the form of flooding.

Contrary to the previous LSTM papers discussed, (Akbari Asanjan et al. 2018; Poornima and Pushpalatha 2019; Karevan and Suykens 2018; Chhetri et al. 2020) all use some form of modified LSTM model for their forecasting. Asjan et al. propose combining an LSTM model with the pre-existing Precipitation Estimation from Remotely Sensed Information using Artifical Neural Networks (PERSIANN) model into one unified framework (Akbari Asanjan et al. 2018). They show that their unified framework outperforms the baseline models in several metrics, including convective rainfall prediction. While the results of this paper are impressive, a similar method for drought prediction would not be possible as to my knowledge no such PERSIANN model exists for droughts. Poornima and Pushpalatha present a modified LSTM model called Intensified LSTM, which uses weighted linear units to mitigate the effects of a vanishing gradient on rainfall prediction (Poornima and Pushpalatha 2019). Poornima and Pushpalatha compare their Intensified LSTM with a number of models including an ARIMA, several RNN models and a non-weighted LSTM model, and show that the Intensified LSTM performs the best in terms of accuracy and Root Mean Square Error (RMSE). The Intensified LSTM shows marginal improvement over a base LSTM, however as it is yet to be seen if the vanishing gradient problem will effect the drought prediction, the Intensified LSTM method will not be used in this paper. Karevan et al. propose using a 2 layer spatio-temporal stacked LSTM for use in predicting temperature ranges in a variety of locations (Karevan and Suykens 2018). The first layer consists of 5 separate LSTM models, one for each location in the training set. The hidden states of each model are then each fed into the second layer before outputting a prediction. Since the data set for this project is split into several counties, this would seem like a promising method for the drought prediction. However the data set uses 28 different counties compared to the 5 used in the paper, so the resulting model used would have to be quite large. Chhetri et al. present a Bidirectional LSTM and Gated Recurrent Unit (BLSTM-GRU) hybrid model for use in a case study of monthly rainfall prediction in Simtokha, Bhutan(Chhetri et al. 2020). They then compare their model to a variety of models including a basic LSTM and an ANN, finding the BLSTM-GRU model to perform marginally better than the others, with a single-layer LSTM coming in at second place. This model was shown to perform quite well, however at a much higher model complexity. The LSTM used for comparison was also only a single layer, and based off the results of the previously discussed 2-layer LSTMs there is potential that a 2-layer LSTM could perform similarly with much less model complexity.

While the previously discussed papers all show that LSTMs perform quite well for weather forecasting, they were all being used for specifically regression-based weather forecasting. Furthermore, most of the discussed papers were dealing with temperature or rainfall prediction, and not drought severity. There is far less research on classification-based weather forecasting, compared to regression-based forecasting, however research shows it is still a valid line of research. As seen in Shi et al. an ensemble of SVMs can be used to classify a weather forecast. Furthermore, Gao et al. show that LSTM models can also be effective when used in classification-based weather forecasting as opposed to regression-based forecasting (Gao et al. 2019). This paper will focus on the task of weather classification through time-series analysis, as opposed to regression.

Methodology

This paper utilizes three different machine learning models; a random forest classifier, time-series forest classifier and an LSTM.

Random forest classifiers are an ensemble learning method. The specific ensemble method random forests are derived from is called bootstrap aggregating, commonly referred to as bagging (Breiman 2001). Bagging works by

building many weak learners and then combining their outputs together to determine an aggregated prediction. Here a weak learner means a model that as poor accuracy, but is marginally better than random guessing. In this case the weak learner refers to a single decision tree. Decision trees work by recursively selecting a feature from the data set, and then splitting the data set on that feature to build the next node in the tree. In a decision tree, each node represents a specific feature, and each edge represents a possible value of the feature. The splitting feature is selected based on the amount of information gain that would be achieved by splitting on said feature. Assuming there are k outcomes $c_1, ..., c_k$, the entropy value for a distribution can be calculated using the following equation (Gao 2020)

$$I(P(c1), ..., P(c_k) = -\sum_{i=1}^{k} P(c_i) \log_2(P(c_i)) \quad (1)$$

For testing a specific feature in the binary case, the number of positive and negative values for each value of that feature is considered. If there are p positive features and n negative features in the data set before splitting on the feature, then the prior entropy value can be calculated as

$$H_{before} = I(\frac{p}{p+n}, \frac{n}{p+n}) \quad (2)$$

Then for each value, $v_1, ..., v_k$, of the feature the expected entropy can be calculated using

$$H_{after} = \sum_{i=1}^{k} \frac{p_i + n_i}{p+n} \cdot I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}) \quad (3)$$

where p_i and n_i are the number of positive and negative values for the i^{th} value. The information gain can then be calculated as

$$InfoGain = H_{before} - H_{after} \quad (4)$$

The feature that results in the highest amount of information gain will be selected to be split on. While the above formulation works in the binary case, this project is performing multi-class classification with 6 different classes. As such, instead of using entropy the random forest will be using Gini impurity as the measure for feature splitting instead, since Gini impurity can easily be extended to the multi-class setting (Breiman et al. 1983). Given j classes, the Gini impurity can be calculated as

$$G(p) = \sum_{i=1}^{j} p_i \sum_{k \neq i}^{j} p_k = 1 - \sum_{i=1}^{j} (p_i)^2 \quad (5)$$

where p_i is the fraction of examples labeled as class i. The algorithm will pick a split that results in the smallest impurity value, i.e. the smaller the Gini impurity, the better a prediction will be. Then, using the Gini impurity, it stands to reason that the random forest should be a reasonable model to use for the drought classification. The non-temporal result can serve as a baseline to be compared to the time-series forest result to see how the inclusion of temporal data effects the prediction rate.

The time-series forest model is very similar to that of the non-temporal random forest model discussed above. One major difference is that the time-series forest engineers some temporal features to aid in the prediction process(Deng et al. 2013). For a given time interval over the data set, three features types are considered: mean, standard deviation and slope, referred to as f_1, f_2 and f_3 respectively. Over some time interval $[t1, t2]$, the feature types are calculated as follows:

$$f_1(t1, t2) = \frac{\sum_{i=t_1}^{t_2} v_i}{t_2 - t_1 + 1} \quad (6)$$

$$f_2(t_1, t_2) = \begin{cases} \sqrt{\frac{\sum_{i=t_1}^{t_2} (v_i - f_1(t_1, t_2))^2}{t_2 - t_1}}, t_2 > t_1 \\ 0, t_2 = t_1 \end{cases} \quad (7)$$

$$f_3(t1, t2) = \begin{cases} \hat{\beta}, t_2 > t1 \\ 0, t_2 = t_1 \end{cases} \quad (8)$$

where v_i is the value at time i and $\hat{\beta}$ is the slope calculated from the least squares regression line over the time interval (Deng et al. 2013). The other major difference is the splitting criteria used. While random forest just uses some form of information gain, time-series forest uses a mixture of information gain and distance, which the authors refer to as the *entrance* gain. The entrance gain is calculated as follows:

$$E = \delta Entropy + \alpha \cdot Margin \quad (9)$$

$\delta Entropy$ is the information gain, α is some small value that can break ties on similar entropy gain cases, and $Margin$ is calculated as

$$Margin = \min_{n=1,2,...,N} |f_k^n(t_1, t_2) - \tau| \quad (10)$$

where τ is a threshold value, and $f_k^n(t_1, t_2)$ is the k^th feature type found for the n^th interval instance. This entrance gain metric can also be modified to use the Gini impurity instead of entropy gain. For consistency between the random forest and time-series forest, the time-series forest will also use Gini impurity when training. As shown by (Deng et al. 2013), time-series forest with entrance gain performs quite well on multi-label classification tasks. Therefore it is likely that it will perform well on the drought classification task this project is aiming to solve.

LSTMs are a variant of the RNN model, which utilize various types of control gates in a cell to regulate what is passed through the model (Hochreiter and Schmidhuber 1997). LSTMs work by accepting input data x_0, which is the first data example. This data goes into the first LSTM cell, and from it the cell generates a cell state c and a hidden state h. c and h are both passed sequentially to the next cell in the current layer, but h can also be passed as input to the next LSTM layer if such a layer exists. Both c and h are modified by the cell using a variety of control gates. An example of an LSTM cell can be seen in figure 1. In the diagram, c_t, h_t, and x_t represent the cell state, hidden state and input example at time t respectively.σ represents a sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$, $tanh$ represents the hyperbolic tan function, $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$. w represents

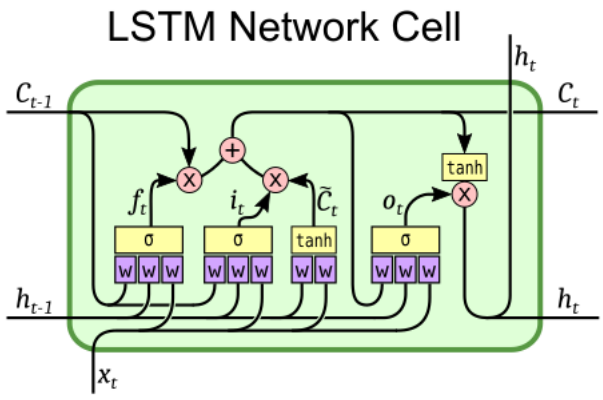

Figure 1: From (Kvita 2016), a diagram depicting how the various gates in an LSTM cell interact with the cell state.

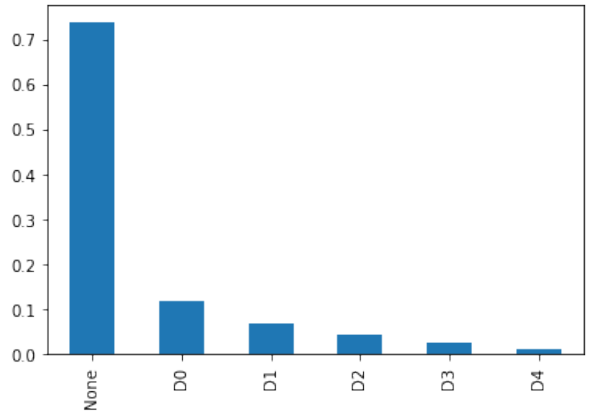

Figure 2: Graph showing the imbalance of labels in the data set (Minixhofer 2019).

the learned weights and biases, and contains both the input weights W, as well as the recurrent weights U. All operators seen in the blue circles stand for element-wise operations.

As seen in figure 1, each gate in the LSTM cell is denoted by a function; these functions being f_t, i_t, $\tilde{C}_t$, and o_t. Introduced by (Gers, Schmidhuber, and Cummins 2000), f_t is the forget gate, which allows the LSTM cell to forget prior information it has learned, if more important information that it must remember comes along. It takes the previous cell state, the previous hidden state and the current input data as input. i_t is the input gate and it determines how important new data is for updating the cell state. It uses the same input as the forget gate. $\tilde{C}_t$ is the cell update function, which, along with the input gate, regulates how the cell state is updated. This cell update takes the previous hidden state and the input data into consideration, but ignores the previous cell state. Finally, o_t is the output gate, and it is used to update the hidden state of the model. It takes the newly updated cell state, the previous hidden state and the input data as input. These gates and the state updates can be formulated as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{11}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{12}$$
$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{13}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{14}$$
$$c_t = c_{t-1} \odot f_t + i_t \odot \tilde{C}_t \tag{15}$$
$$h_t = o_t \odot \tanh(c_t) \tag{16}$$

where $\odot$ represents element-wise multiplication (Gers, Schmidhuber, and Cummins 2000). LSTMs excel at learning temporal data, so the model should perform quite well on the time-series drought classification task.

As previously mentioned, the data set being used is a public meteorological data set found on Kaggle[1]. It contains a wide array of weather features recorded from numerous counties across the United States between the years 2000 and 2020. All features are numerical, and some of the included features cover temperature, wind speed, surface pressure, and precipitation levels at various heights. Each example instance in the data set covers a period of 90 days for a single county, with a corresponding drought label. These labels are heavily imbalanced, as seen in figure 2. The data is pre-split into training, validation and test sets, with the following proportions; 47%, 10% and 43% respectively. Since the data set is originally given in JSON format, the first preprocessing step will be to convert it to a more useful tabular format using Pandas. Next, since each feature has different numerical ranges, they will be individually min-max scaled to normalize all data between 0 and 1. This will be done separately across the training, validation and test splits to prevent data leakage. Since the examples labeled with no drought dominate the data set, under-sampling them to a similar level as the other labels should help improve results.

Results

By under-sampling the "None" label examples, accuracy would be one viable way that the model can be evaluated. Accuracy can even be calculated individually for each label, i.e. to find the accuracy for label i you would only consider the test examples that have i as the true label. Evaluating accuracy in this manner could be indicative to any labels that the model might specifically struggle with.

Another effective method would be building a confusion matrix and using it to compute the precision, recall and F1 score. Precision represents the ratio of true positive prediction versus the combination of true positive (TP) and false positive (FP) predictions.

$$Precision = \frac{TP}{TP + FP} \tag{17}$$

Recall represents how many relevant predictions were made by looking at the ratio of true positives to a combination of true positives and false negatives (FN).

$$Recall = \frac{TP}{TP + FN} \tag{18}$$

[1] https://www.kaggle.com/cdminix/us-drought-meteorological-data

These metrics can then be combined into the F1 score, which is the harmonic mean of the precision and recall. The F1 score gives a more representative measure of model accuracy in multi-label classification than naive accuracy does.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{19}$$

While typically these metrics are used in a binary case, they can be extended to the multi-label case by extending the number of columns and rows in the confusion matrix. Then micro-averaging can be applied to average the precision and recall across all labels, as shown by (Zhang and Zhou 2014). For example finding the micro average for recall would be

$$MA_{recall} = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + \sum_{i=1}^{L} FN_i} \tag{20}$$

Both the random forest and time-series forest will initially use default hyper-parameter values, meaning they will have 100 trees, a minimum of 1 sample per leaf node, no maximum depth and will consider $\sqrt{N_{features}}$ when looking for the best split. These will not be tuned much during training as they are meant to be a baseline to compare the LSTM results to, but if initial results are poor some tuning will be done.

The LSTM consists of two recurrent layers and an output layer. There are 18 features in the data set, so the first recurrent layer will use 18 neurons, one for each feature, and will have an output size of 12. The second layer will have 12 neurons and have an output size of 6. The output layer will have 6 neurons, one for each class, and will use softmax activation to get a probability distribution across the different classes. The most probable class from this distribution will be taken as the prediction. Since there is no specific method to selecting layer sizes, the layer sizes were picked so that they gradually scale down in size from the number of features to the number of labels. These values will be tweaked as training is performed in order to optimize model performance. An ADAM optimizer (Kingma and Ba 2017) with a default learning rate (0.01) is used with a categorical cross-entropy loss function, as ADAM has been shown to be an effective optimizer and categorical cross-entropy is well suited to a multi-label classification problem.

In order to determine how balancing the data set affects the results of the models, the models were trained on the full, unbalanced training set, as well as an under-sampled training set where all classes appear with equal frequency. The system consists of a data pipeline that loads and pre-processes the training, validation and test sets. Due to the amount of time this pre-processing step takes, the pipeline saves the processed data for subsequent use, so the pre-processing step must only be done once. This data is then balanced if the appropriate flag is active. The balancing is achieved by finding the number of occurrences of the least frequent label, in this case 'D4', and randomly under-samples the other classes so that the frequency of each labels is equal. This ends up being 40000 examples for each label. The system then builds and trains the models sequentially in the order random forest, time-series forest and LSTM. Training and testing one set of models takes an upward of 10 hours, so time was a constraint for this project. Furthermore one technical challenge was debugging three different models at various stages of model building, as one model could finish the majority of a run and then crash due to an error. After applying a fix for the error I would then have to wait for the model to reach the same point to see if the error was fixed or not. As such, debugging the models took quite a bit more time than expected. In hindsight I should have used a small partition of each data set to more quickly iterate on the debugging process.

Random Forest Results

The random forest was trained to be used as a baseline to see how the use of the time-series aspect of the data affects the classification results. Since the data set only has a single label for each series, the random forest is trained using only the features from the final day of each series. Table 1 shows the results of applying the validation set to the trained random forest models using both the unbalanced and balanced data set.

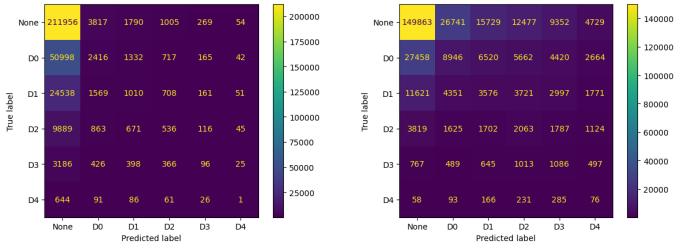

Table 1: Validation confusion matrices for the random forest model on the unbalanced (left) and balanced (right) training sets.

In both cases the model over estimates toward the 'None' drought label, but for the balanced data set the model estimates the other classes a bit more frequently. The averaged accuracy, precision, recall and F1 scores over each label can be seen below:

	Acc.	Prec.	Rec.	F1
Unbalanced	.892	0.186	0.239	0.176
Balanced	.839	0.245	0.209	0.208

As it can be seen, balancing the data set had a positive effect on the precision, recall and F1 scores but decreased the accuracy as expected. A similar effect can be seen on the test results, where the confusion matrices can be seen in table 2.

The averaged metrics over each label for applying the test set to the trained random forest models are as follows:

	Acc.	Prec.	Rec.	F1
RF	0.911	0.191	0.226	0.192
RF-Balanced	0.829	0.244	0.205	0.195

Where again it can be seen that balancing the data set causes a decrease in model accuracy but an increase in every other metric.

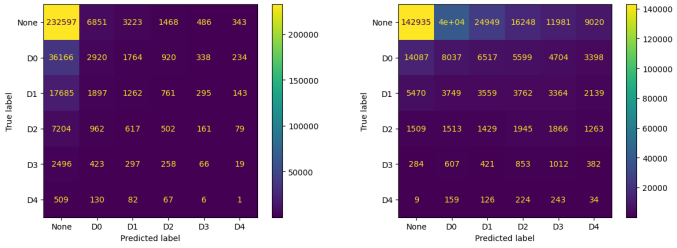

Table 2: Test confusion matrices for the random forest model on the unbalanced (left) and balanced (right) training sets.

Time Series Forest

Originally the time-series forest was meant to use the implementation from the package *pyts* (Faouzi and Janati 2020), but found that their implementation was incapable of using the data shape in my implementation; i.e. a list of week-long series with a label, which was 3-dimensional. As such, I implemented their idea for using aggregated window values directly into my system. Therefore, when training the TSF I compute the average, standard deviation and slope for each feature over every day in the series. This results in a feature vector of length $3 * n$, where n is the number of features. I calculate this averaged feature vector for each series and train a random forest model using these vectors, similarly to the *pyts* implementation.

Table 3 shows the confusion matrices for the time-series forest on the validation set.

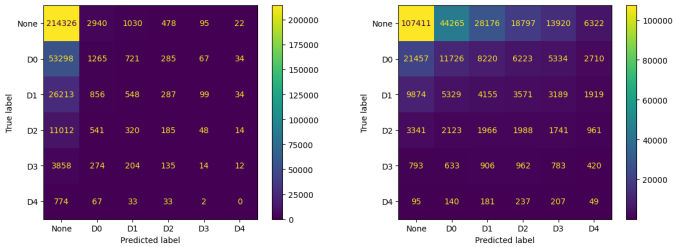

Table 3: Validation confusion matrices for the time-series forest model on the unbalanced (left) and balanced (right) training sets.

Similarly to the random forest, the time-series forest over predicts the 'None' label in the unbalanced data set, as the 'None' label makes up the vast majority of labels. In this case it never even predicts the 'D4' label, since it is the least frequent label. When the data set is balanced the time-series model predicts the other labels with much more frequency. The averaged validation metrics for the time-series forest can be seen bellow:

	Acc.	Prec.	Rec.	F1
TSF	0.892	0.173	0.212	0.153
TSF-Balanced	0.798	0.207	0.188	0.176

Interestingly, every metric for the time-series forest actually decreases when the data set is balanced, even though the non-'None' label examples are predicted more frequently.

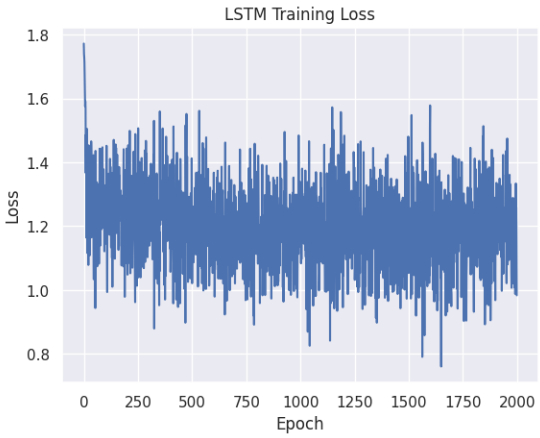

Figure 3: Unbalanced LSTM loss graph

Possible reasoning for this will be discussed in the Discussion section later. When evaluating the TSF models on the test set the results end up being quite similar. See table 4 for the confusion matrices.

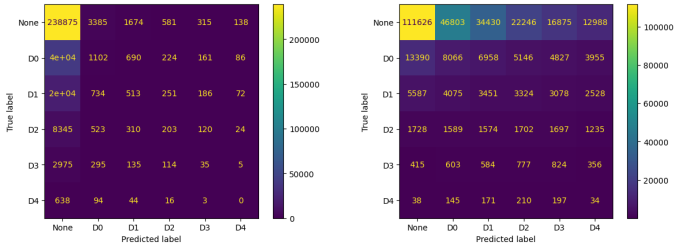

Table 4: Test confusion matrices for the time-series forest model on the unbalanced (left) and balanced (right) training sets.

The test results have the following average metrics:

	Acc.	Prec.	Rec.	F1
TSF	0.915	0.176	0.215	0.166
TSF-Balanced	0.796	0.209	0.188	0.164

As seen, balancing the data set causes the expected decrease in accuracy, as well as a decrease in recall, while precision is increased. The F1 scores for both are comparable. Reasoning for this disparity from the expected result of an increase in the non-accuracy metrics is discussed further in the Discussion section.

LSTM

As mentioned before, the time taken to train the model was quite long, with the LSTM taking approximately 6.5 hours to run for 2000 epochs. As such, the desired amount of hyper-parameter tuning was not fully achieved. It was found that including a small amount of dropout (0.1), increasing the hidden state size and increasing batch size all result in a marginal improvement in the metric values. The LSTM results shown in this section correspond to a hidden layer size

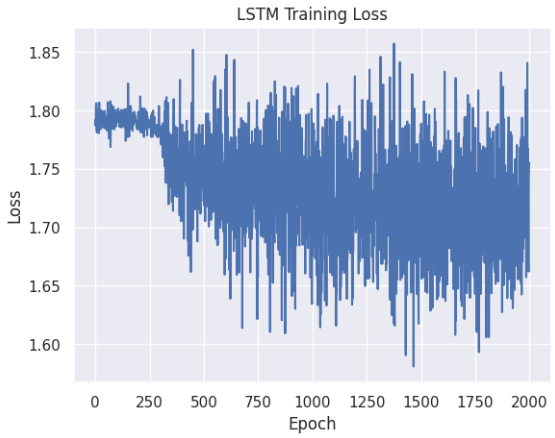

Figure 4: Balanced LSTM loss graph

of 128. A larger LSTM with a hidden size of 512 over 5000 epochs was attempted, but unfortunately the training has yet to converge after 13 hours, and will not be included in this report.

It was found that the training of the LSTM was very unstable, and the loss value frequently fluctuated across epochs, as can be seen in figures (3) and (4). Further experimentation involving different optimizers, learning rates and batch sizes would likely be required to improve training stabilization.

Since testing the complete validation set after each epoch would dramatically increase training time, a random sample of size 10000 is taken and used across all training epochs for consistency. After each epoch this sample is used to evaluate the model on all of the performance metrics.

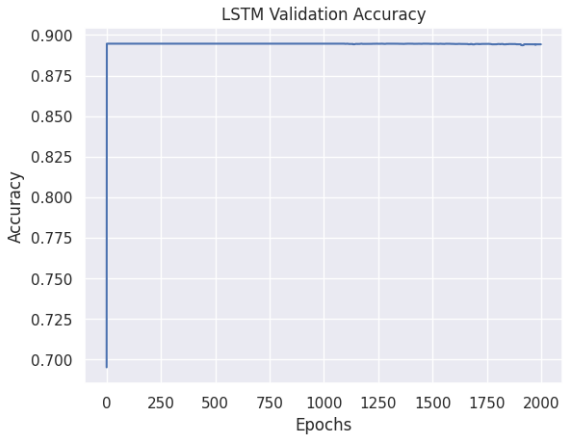

Figure 5: Unbalanced validation accuracy.

Figures (5) and (6) show the validation accuracy plots for both the unbalanced and balanced data sets. In the case of the unbalanced data, the LSTM converges to an accuracy value of 89% very quickly with little deviation. This is likely due to the overabundance of the 'None' label in the unbalanced data, so the LSTM quickly learns to only predict 'None'.

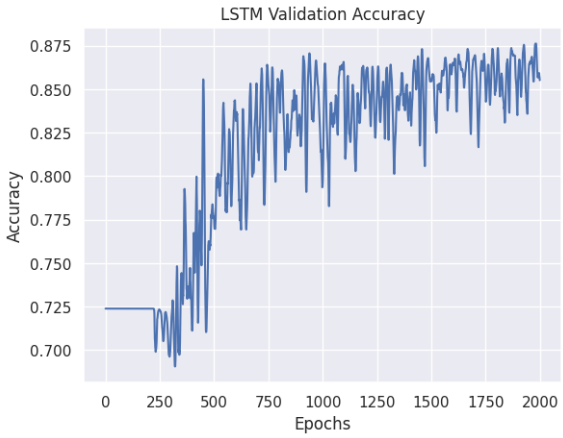

Figure 6: Balanced validation accuracy.

The balanced data set gradually increases and converges to a value of 86%, as it learns to predict across all labels.

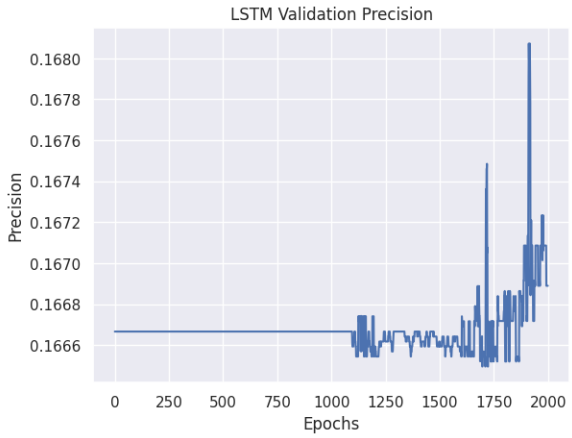

Figure 7: Unbalanced validation precision.

The unbalanced and balanced precision plots can be seen in figures (7) and (8) respectively. Again it can be seen that there is only a small deviation from the initial value for the unbalanced data set, as it starts completely flat at 0.1667 for 1000 epochs, before increasing slightly and converging around a value of 0.1670. It is not clear exactly why it took so long for a change to be seen in the precision value. The value of the balanced data set also stays static at a value of 0.1700, before steadily increasing around epoch 250. There is a large drop of 0.04 at the end of the balanced plot, but it seems like the model starts to increase again, and it seems likely the value would converge back to 0.2900 if more training epochs had been used.

A similar trend can be found in the recall plots seen in figures (9) and (10). The unbalanced data set levels out with small fluctuations, converging to a value of approximately 0.124. The balanced data generally increases across the training phase, with some local oscillations and spikes

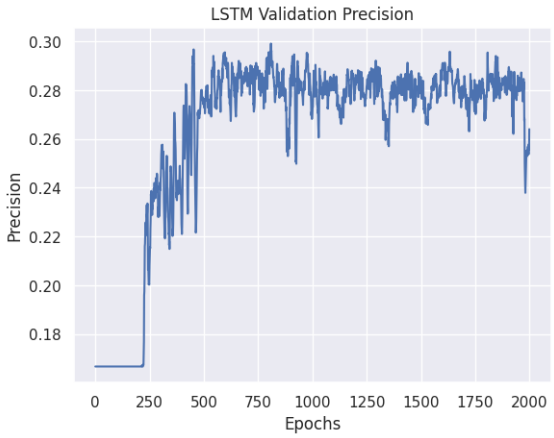

Figure 8: Balanced validation precision.

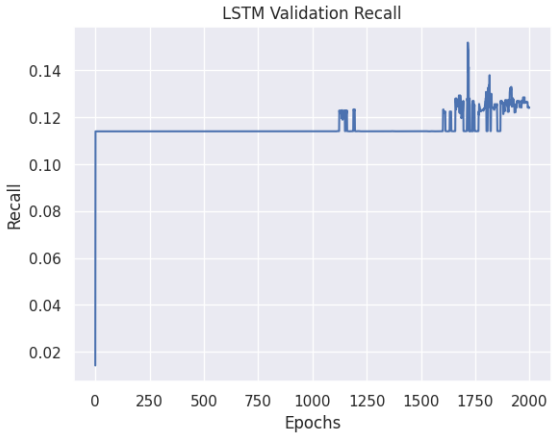

Figure 9: Unbalanced validation recall.

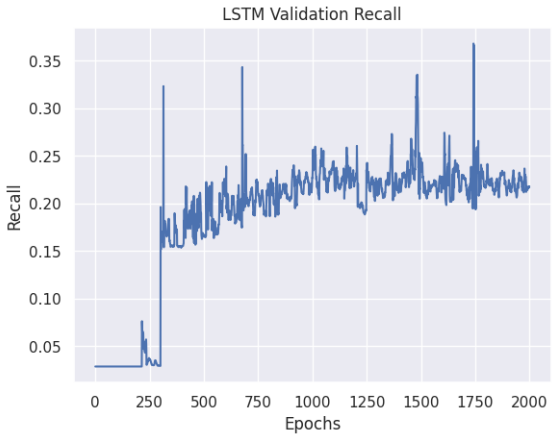

Figure 10: Balanced validation recall.

occurring occasionally, and seems to begin converging to a value around 0.22.

Finally the F1 scores can be observed in figures (11) and

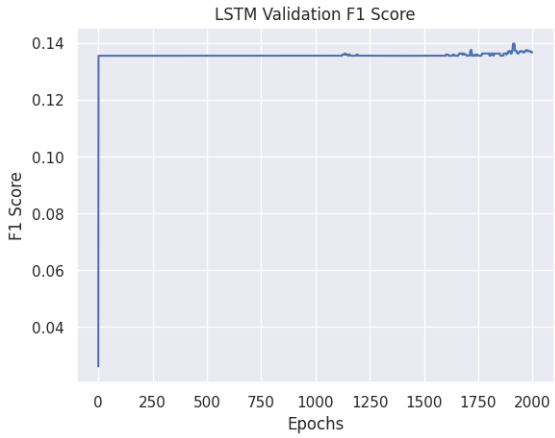

Figure 11: Unbalanced validation F1 score.

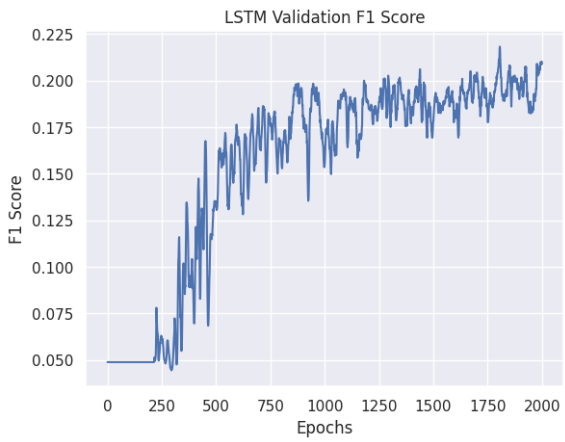

Figure 12: Balanced validation F1 score.

(12). Due to the precision and recall values leveling out, the unbalanced F1 score also converges within a few epochs. As expected, the balanced F1 score follows the same trend as the balanced precision and recall values, showing a steady upwards trend with some local fluctuations.

Based off the plot results for the balanced data set, it seems like all four performance metrics would continue increasing given more training epochs. As such, it seems for this task an LSTM needs many training epochs to perform well, especially due to its training instability.

Confusion matrices for the unbalanced and balanced LSTMs can be observed in table (5). Here it is easy to see that the unbalanced LSTM only ever learns to predict the 'None' label, with an occasional 'D1' prediction. As such, it maintains similar accuracy to the unbalanced random forest and time-series forest models, but its performance in the other metrics are quite a bit worse than both. The balanced LSTM model however maintains the best accuracy out of all of the balanced models by approximately 3%, while also having the best precision, and F1 score out of all the evaluated models. The various LSTM metric values can be seen

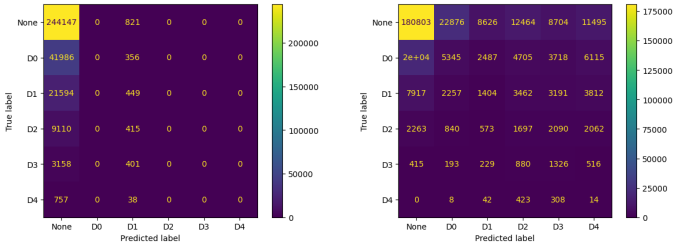

Table 5: Test confusion matrices for the LSTM models on the unbalanced (left) and balanced (right) training sets.

in table (6), along with the RF and TSF metric values. The bold values correspond to the highest value for each metric.

Model	Acc.	Prec.	Rec.	F1
RF	0.911	0.191	**0.226**	0.192
RF-Balanced	0.829	0.244	0.205	0.195
TSF	**0.915**	0.176	0.215	0.166
TSF-Balanced	0.796	0.209	0.188	0.164
LSTM	0.91	0.167	0.157	0.150
LSTM-Balanced	0.863	**0.249**	0.212	**0.206**

Table 6: Comparison of average test results across all models.

In a direct comparison, it seems that the LSTM performs very poorly on an unbalanced data set, as it has the worst precision, recall and F1 values out of all other models. However it can be said that when the data set is balanced then an LSTM is the best performing model in general, as it results in the highest F1 score, which correlates to the best model in a multi-label domain, like the drought data set being used. Interestingly, the TSF model actually performs worse when the data set is balanced, and in general the RF models out perform the TSF models. Even though it is seen that the LSTM model outperforms all other models on the balanced data set in terms of F1 score, it is only a marginal improvement. It is feasible that increasing the size of the LSTM's hidden layers or increasing the number of training epochs would result in a further improvement, but further experimentation would be required. It is also possible that an F1 score of 0.2 is the best that can be achieved on the given data, and this idea is further explored in the upcoming Discussion section.

Discussion

This section will discuss the implications and possible reasoning for the results seen in the Results section.

LSTM Performance

It can be seen that the LSTM is the best overall model when used with balanced data, and results in a marginal improvement over the non-temporal RF model. The LSTM model manages to outperform all other models in terms of micro-averaged F1 score, and maintains the highest accuracy when compared to the unbalanced models. This shows that the LSTM model is able to differentiate between the various drought levels more effectively, while maintaining the highest accuracy without overfitting. However, even though the LSTM model is the best overall model, it still seems to begin to plateau in every metric, and it is unlikely that parameter tuning would be able to improve the model much more past the plateau point. Further experiments were conducted on the LSTM using more training epochs and a larger hidden cell size, the results of which can be seen in appendix A. From this, it can be seen that increasing the complexity of the LSTM does not guarantee an improved performance on the data set. This results suggests that there is an issue that lies in the data set itself, and that an average F1 score of 0.200 could be the best score that can be attained using the data set as it is.

Time-Series Forest Results

A surprising result found from this experiment is that the TSF models perform the worst in terms of micro-averaged F1 score. The RF models out-performed the TSF models using only the feature values from the final time step in each series. This is likely due to the TSF model being very sensitive to small changes in the feature values. It seems like aggregating each feature over a series into its mean, standard deviation, and slope causes some information in the series to be lost. This aggregation could be causing the TSF models to pick poor split points that hinders its ability to learn the patterns of the data set. Perhaps utilizing different aggregation features could lead to better results, such as variance, the average difference between days, or the difference between the minimum and maximum values.

Limitations of the Data Set

As mentioned previously, the results seen could be a direct consequence of limitations derived from the data set itself. While the data set contains a number of features, it is known that the weather is a very complex phenomenon. The data set used only contains 18 weather features measured in individual U.S. counties, with no information about weather in surrounding counties. It is very probable that only 18 features taken from a single area is not enough information to make accurate predictions about whether a drought will happen, or the severity of it if it does. Recently the original data set found on Kaggle (Minixhofer 2019) was updated to include non-temporal soil information for the counties as well, increasing the number of usable features by 31. This additional information could help increase the classification scores of the models used, or allow a secondary model to be trained on soil data to be used in conjunction with the time-series models. The data set, and by extension the models, also treats each county as independent from all other counties, which likely imposes a large limitation upon the ability to predict potential droughts. Weather events are almost never found to be localized within a single county, they often encompass a wide area. Due to this, it is probable that the performance of the models can be improved by considering weather conditions in surrounding counties as well. If county A and county B are adjacent, and county A is experiencing drought, then

it is more likely that county B will experience some level of drought as well.

Another limitation that can be considered is the length of the time series. Each time series contains 7 time steps, one for each day of a week, and the drought label is applied to the whole week. It is possible that only considering a week at a time is not a sufficient length of time for the LSTM models to meaningfully learn drought predictors. It would likely be more effective to look at data over a period of several weeks, and to use a sliding window to increase the length of the series. In other words when classifying a series S_t at time t, the model also considers $S_{t-1}, S_{t-2}, ..., S_{t-k}$, where k is the number of weeks in the past the model will look. This will result in the model learning drought patterns over a longer period of time, which would likely improve performance. This can be combined with the previous suggestion for using adjacent county data, such that when predicting on county C at time t, the model takes into consideration the adjacent counties at time $s < t$. This would help account for possible weather migrations.

Future Work

While this project has shown some promising results in the usage of LSTMs for prediction and classifying drought severity, there are multiple avenues of further research that can be done to improve the results. The first avenue would be improving and expanding the data set to include more features that might indicate drought. As mentioned in the Discussion section, this would involve using the updated soil data, or by including weather features from adjacent counties in the feature vector of the county being predicted on. Another avenue would be changing the length of the time series used. Instead of using a single week, a time series can be constructed from the data that incorporates several weeks at a time. For example, when training instead of using only S_4 to predict on S_4, can use $S_{1,2,3,4}$ which include the feature values of the previous three series. This would then cause the model to predict using a time series that encompasses an entire month. A third avenue of research could be further parameter tuning on a deeper LSTM. The LSTM used in the project only consisted of 2 layers, as in general that is sufficient. However, drought classification is a very complex problem, and it is likely that making the LSTM deeper could better help it learn patterns over a time series. For example, the time series used had 7 time steps, so perhaps an LSTM with 7 layers would map each day to a layer and be able to learn a deeper understanding of how the weather changes on a daily basis.

Conclusion

Droughts are a complex weather phenomenon that can cause untold damage if an area is not prepared for it. If a method was found to accurately predict and classify the severity of drought events, then precautions could be taken to help mitigate the damage caused. Weather prediction has been a popular topic in the field of machine learning for years, however previous literature mainly focused on the act of predicting weather through regression methods. This paper explored the application of machine learning models to the problem of predicting drought severity as a classification task, as opposed to a regression task. Three different models were used for classification; a random forest, a time-series forest, and a long short term memory model. The RF model acted as a baseline to compare the results of the time-series models, as it was hypothesized that utilizing the temporal nature of weather would improve model performance. A data set of independent U.S. county weather features was used, and the time series in the data set consist of seven days each. This data set was highly imbalanced towards the 'None' drought label, so an undersampling technique was applied to ensure each class was equally represented. The primary metric of model performance was the F1 score micro-averaged across all classes, and the averaged accuracy, precision and recall were also measured. It was found that the LSTM performs the best out of models on the balanced data set in terms of accuracy, precision and F1 score by a marginal amount. While it seems that the model performance is limited by the data set, some future avenues of research were discussed to mitigate this and improve results. Overall, it seems that using LSTMs for classifying drought severity is a promising solution to a complex issue.

References

[Akbari Asanjan et al. 2018] Akbari Asanjan, A.; Yang, T.; Hsu, K.; Sorooshian, S.; Lin, J.; and Peng, Q. 2018. Short-term precipitation forecast based on the persiann system and lstm recurrent neural networks. *Journal of Geophysical Research: Atmospheres* 123(22):12,543–12,563.

[Amei Amei and Ho 2012] Amei Amei, W. F., and Ho, C.-H. 2012. Time series analysis for predicting the occurrences of large scale earthquakes. *International Journal of Applied Science and Technology* 2(7).

[Breiman et al. 1983] Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. J. 1983. Classification and regression trees.

[Breiman 2001] Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

[Chhetri et al. 2020] Chhetri, M.; Kumar, S.; Pratim Roy, P.; and Kim, B.-G. 2020. Deep blstm-gru model for monthly rainfall prediction: A case study of simtokha, bhutan. *Remote Sensing* 12(19).

[Deng et al. 2013] Deng, H.; Runger, G.; Tuv, E.; and Vladimir, M. 2013. A time series forest for classification and feature extraction.

[Faouzi and Janati 2020] Faouzi, J., and Janati, H. 2020. pyts: A python package for time series classification. *Journal of Machine Learning Research* 21(46):1–6.

[Gao et al. 2019] Gao, M.; Li, J.; Hong, F.; and Long, D. 2019. Day-ahead power forecasting in a large-scale photovoltaic plant based on weather classification using lstm. *Energy* 187:115838.

[Gao 2020] Gao, A. 2020. Lecture 7 notes - decision trees. `https://learn.uwaterloo.ca/d2l/le/content/633098/viewContent/3399341/View?ou=633098`.

[Gers, Schmidhuber, and Cummins 2000] Gers, F.; Schmidhuber, J.; and Cummins, F. 2000. Learning to forget: Continual prediction with lstm. *Neural computation* 12:2451–71.

[Gray and Merzdorf 2019] Gray, E., and Merzdorf, J. 2019. Earth's freshwater future: Extremes of flood and drought – climate change: Vital signs of the planet.

[Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

[Kaneko, Nakayoshi, and Onomura 2019] Kaneko, R.; Nakayoshi, M.; and Onomura, S. 2019. Rainfall Prediction by a Recurrent Neural Network Algorithm LSTM Learning Surface Observation Data. In *AGU Fall Meeting Abstracts*, volume 2019, GC43D–1354.

[Karevan and Suykens 2018] Karevan, Z., and Suykens, J. A. K. 2018. Spatio-temporal stacked LSTM for temperature prediction in weather forecasting. *CoRR* abs/1811.06341.

[Khan et al. 2020] Khan, M. M. R.; Siddique, M. A. B.; Sakib, S.; Aziz, A.; Tasawar, I. K.; and Hossain, Z. 2020. Prediction of temperature and rainfall in bangladesh using long short term memory recurrent neural networks. *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*.

[Kingma and Ba 2017] Kingma, D. P., and Ba, J. 2017. Adam: A method for stochastic optimization.

[Kratzert et al. 2018] Kratzert, F.; Klotz, D.; Brenner, C.; Schulz, K.; and Herrnegger, M. 2018. Rainfall–runoff modelling using long short-term memory (lstm) networks. *Hydrology and Earth System Sciences* 22(11):6005–6022.

[Kvita 2016] Kvita, J. 2016. Visualizations of rnn units. `https://kvitajakub.github.io/2016/04/14/rnn-diagrams/`.

[Minixhofer 2019] Minixhofer, C. 2019. United states drought & meteorological data.

[Poornima and Pushpalatha 2019] Poornima, S., and Pushpalatha, M. 2019. Prediction of rainfall using intensified lstm based recurrent neural network with weighted linear units. *Atmosphere* 10(11).

[Salman et al. 2018] Salman, A. G.; Heryadi, Y.; Abdurahman, E.; and Suparta, W. 2018. Weather forecasting using merged long short-term memory model (lstm) and autoregressive integrated moving average (arima) model. *Journal of computer science* 14(7):930–938.

[Singh et al. 2019] Singh, S.; Kaushik, M.; Gupta, A.; and Malviya, A. K. 2019. Weather forecasting using machine learning techniques. *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)*.

[Smith 2020] Smith, A. B. 2020. 2010-2019: A landmark decade of u.s. billion-dollar weather and climate disasters: Noaa climate.gov.

[Zhang and Zhou 2014] Zhang, M., and Zhou, Z. 2014. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering* 26(8):1819–1837.

Appendices

A - Increasing Training Length and Complexity for LSTM

A further test was done to see how the complexity and training time of the LSTM affects it's performance. An LSTM with a hidden size of 512 was trained over 5000 epochs on the balanced data set. The results can be seen in the following figures. As it can be seen, increasing the complexity and training time of the LSTM model still results in similar performance as the less complex LSTM model with only 128 hidden units. Interestingly, the precision metric seems to begin to decrease past 2000 training iterations. The exact reasoning for this is unknown, but it is likely that the false positive rate is increasing, since the recall metric does not see a similar decrease. The LSTM achieved average test metric values of 0.863, 0.253, 0.210 and 0.189, for accuracy, precision, recall and F1 score respectively. This shows that even with an increased complexity, the LSTM does not always outperform the baseline when looking at the micro-averaged results.

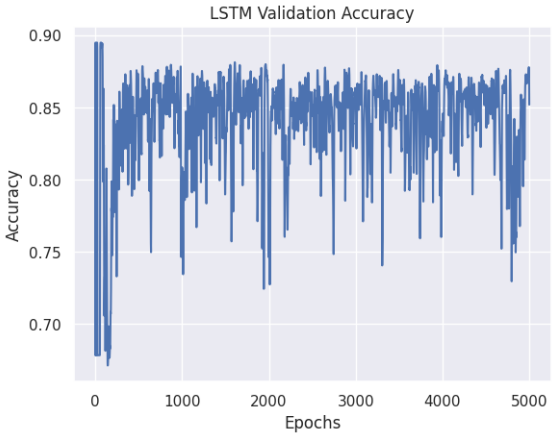

Figure 13: Balanced validation accuracy on the large LSTM model.

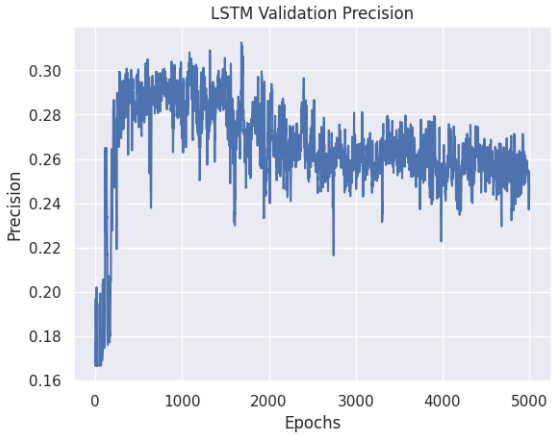

Figure 14: Balanced validation precision on the large LSTM model.

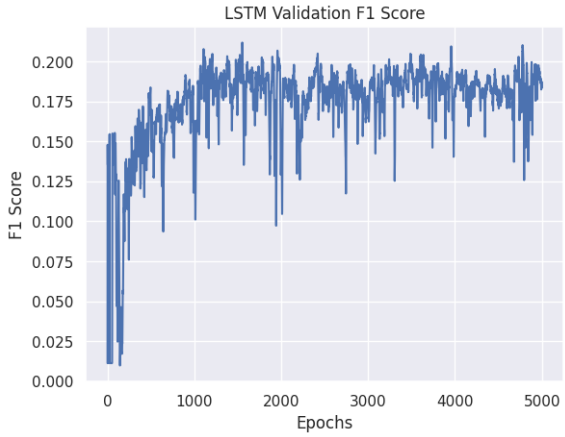

Figure 16: Balanced validation F1 score on the large LSTM model.

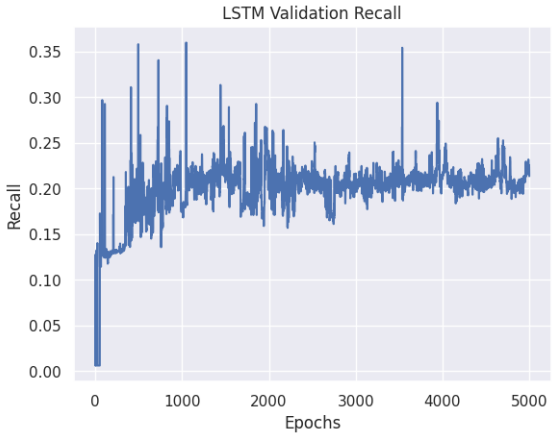

Figure 15: Balanced validation recall on the large LSTM model.