Machine Learning II
Final Project
Professor Amir Jafari

Sequence Prediction with Recurrent Neural Networks: Predicting the Next Airport a Plane Will Visit

Group 1
Gayathri Chandrasekaran,
Jonathan Giguere,
Patrick Maus

Table of Contents

Introduction

Our group was interested in exploring Recurrent Neural Networks (RNNs), so we chose a project focused on predicting the next value in a given sequence. Specifically, we analyzed 3.6 million US airline flights over a period of 6 months. Each record in our dataset includes the airline name, a unique identifier for each aircraft, the origin and destination airports, and the time of the flight. We defined a flight as one unique aircraft flying from one of about 350 airports to another. Our research question is "Given a sequence of N airports visited by a unique plane, can we predict the next airport (N+1)?"

We believe this approach is applicable to numerous different problems beyond predicting the next airport. Our world is awash in devices that record their time and location, leading to an explosion of geospatial data that can inform everything from advertising to pandemic responses. Often, the data is too dense for traditional geospatial analysis methods and a common approach is to represent a dataset as a network or sequence of known locations visited.[1] Applying a similar methodology as the one proposed will likely lead to additional insights in multiple fields and business cases.

For our project, we first conducted extensive exploratory data analysis to better understand the distribution of our data across airlines, airports, and airplanes. We ultimately decided to segment our data to subsets grouped by airlines as it appeared aircraft within different airlines followed their own patterns. Next, we examined the data for stationarity and autocorrelation to ensure there was a signal embedded within the sequences that our models could identify and then learn for prediction. We found strong evidence that our data was stationary and had "colored" signals rather than white noise.

Next, we conducted preprocessing on the data to create equal-length tokenized sequences which could be used as inputs for an RNN. Since we were unsure of the ideal sequence length for this problem, we needed to test multiple different airline subsets with multiple different sequence lengths as a data input for each of our models. To facilitate efficient model development, training, and evaluation in an environment with multiple different input datasets, we developed a data ingest and preprocessing pipeline using a series of Python scripts which employed different "model_name" and "run_name" variables to track activity across different RNN models (LSTM, GRU, seq2seq, etc) and airline/sequence length "runs".

Our team developed, trained, and evaluated models including LSTM, GRU, 1dCONV, bi-directional LSTM, sequence2sequence, etc. I personally was in charge of implementing the sequence2sequence model.

To evaluate our models, we built another script to plot the training history and visualize the precision, recall and F1 measure of our models' predictions for the target (N+1) airport.

[1] Wang, Senzhang, J. Cao and Philip S. Yu. "Deep Learning for Spatio-Temporal Data Mining: A Survey." *ArXiv* abs/1906.04928 (2019): n. pag.

Description of the Data Set

Summary

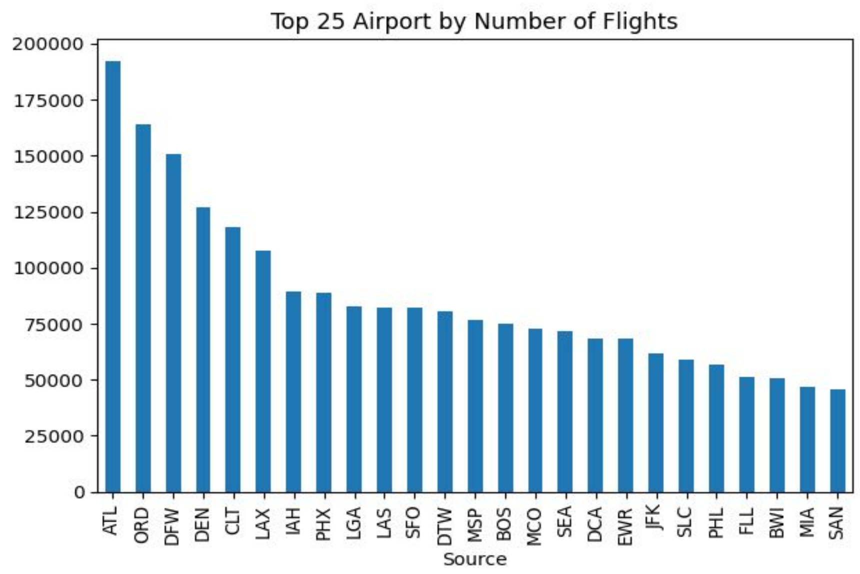
The data set was downloaded from the Bureau of Transportation Statistics, a US Department of Transportation agency charged to collect, analyze, and store transportation statistics.[2] Using six months of data from September 2019 to February 2020, we reviewed about 3.7 million flights from 5,716 unique aircraft. We choose to only use data up to February 2020 to avoid impacts from the Coronavirus pandemic to affect the sequence patterns. Within our data, there are 19 different airlines, but the top five represents about 2/3 of the entire data. Several sample rows are included below.

Table 1. Data sample from February 1, 2020.

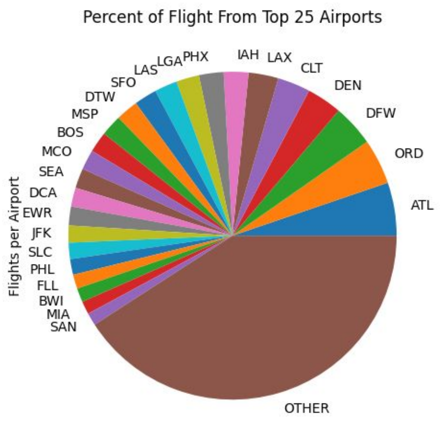
FL_DATE	OP_UNIQUE_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN	DEST	DEP_TIME	ARR_TIME
2020-02-01	MQ	N269NN	3825	ORD	TUL	1646	1820
2020-02-01	MQ	N908AE	3829	JFK	BNA	1336	1458
2020-02-01	MQ	N663AR	3831	GNV	MIA	844	1020
2020-02-01	MQ	N618AE	3833	DFW	SJT	852	955
2020-02-01	MQ	N618AE	3833	SJT	DFW	1024	1132

Exploratory Data Analysis

Next we reviewed the distribution of flight activity across the airports in the data. Our analysis found that the data distribution of number of flights per airport was left-tailed, with a few airports much busier than others.



[2] https://www.bts.gov



The centralization was greater in some airports than others. For example, Delta Airlines has a major hub in Atlanta, which accounts for about 25% of all flight activity for Delta flights. As we built, trained, and evaluated a model that predicts the next sequence, we had to be aware of this centralization because our models could 1) generate artificially higher accuracy scores by more frequently predicting the busiest airport or 2) learn to predict flight activity out of the largest airports but poorly generalize to smaller airports.

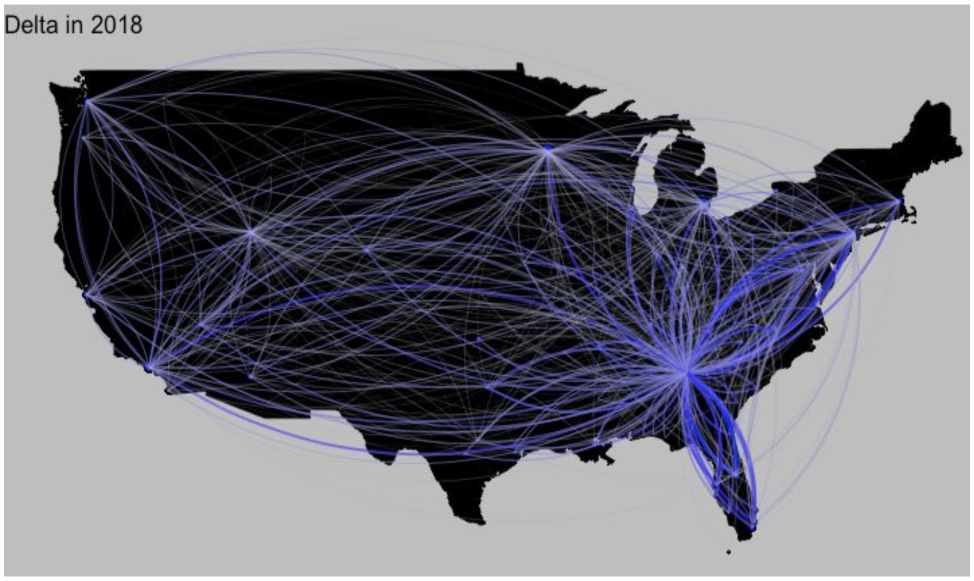
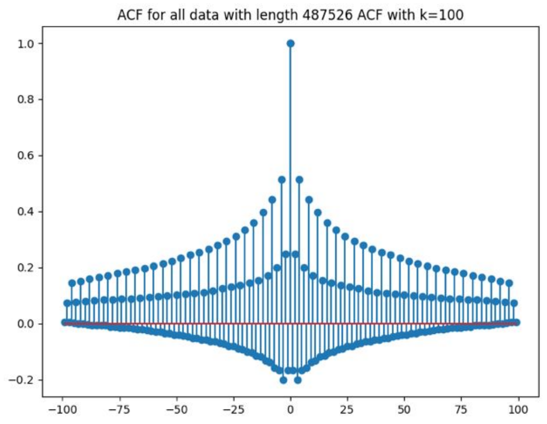


Figure 1. A weighted network plot of Delta's flight activity in 2018 snapped to a map of the continental US. Darker, thicker lines represent greater numbers of flights and clearly show Delta's employment of a "hub and spoke" model.

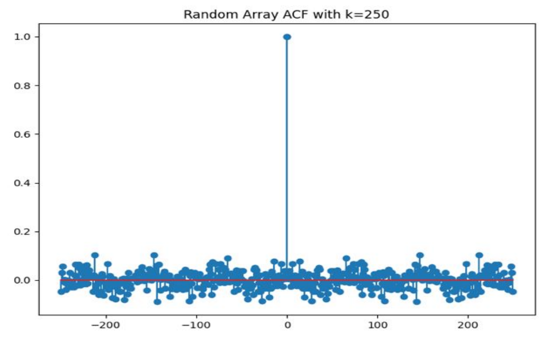
Analysis of Autocorrelation and Stationarity

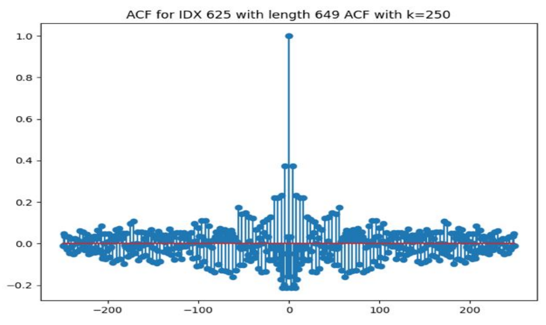
Once we completed our exploratory data analysis, we next focused on ensuring our data was fit for use in sequence prediction. Any machine learning model depends on an actual “signal” to be present in the data, which by definition has a pattern that can be mapped from the domain (or training data) to the range (or test data). In our case, if aircraft do not follow an observable pattern when flying from airport to airport, we would be trying to predict randomness, or “white noise.”



Our first check was to plot the autocorrelation of the entire dataset. An autocorrelation plot shows how well a signal correlates with itself at equal time lags. If the plot shows an impulse of 1 at time lag 0 but then very little or known correlation at subsequent time lags, the data is likely to be random white noise. On the other hand, an autocorrelation plot with strong correlation for multiple lags indicates that previous events influence the current event and suggest an underlying pattern.

Our first ACF plot shows all data for Delta airlines across our time range. We see that there is a definite correlation with later lag times indicating an underlying pattern whose strength fades as the time lags increase. For reference, the next ACF plot shows a random dataset, where we would not expect to have correlations between later time lags.





We also surveyed a selection of random aircraft to see if individual aircraft ACF plots showed good autocorrelation. We found multiple examples similar to the above figure for IDX 625, which corresponds to aircraft tail number 'N849DN'. This shows strong evidence of autocorrelation with several time lags that can decrease in intensity as the time lags increase.

Next, we examined stationarity. A stationary time series is a dataset that lacks a trend or seasonality. Most models, including our RNNs, assume a dataset to be stationary so we should ensure our data is most likely stationary. Since our sequences are drawn from a dataset that likely has both trend and seasonality *in the volume of activity*, we need to be particularly cautious. Fortunately, our selection of sequence order somewhat insulates our data from underlying trend or seasonality. For example, Aircraft A could make a flight between three airports every week, Aircraft B could make multiple flights to multiple airports in the first month before going into maintenance, and Aircraft C could be dormant for the first 5 months and only active the last month of our data. Because we only care about the ordered aspects of airports visited for each flight, our data is somewhat protected from trend and seasonality effects on the volume of activity.

To test for stationarity, we used a similar methodology to the ACF plots. We first used an Augmented Dickey-Fuller (ADF) test against the whole dataset, subsets for different airlines, and finally on individual aircraft. Our results were also similar to the ACF. We found that across the entire data and at the airline levels, we obtained very low test statistics and small p-values which suggest the data is stationary. For example in the Delta airlines data, we obtained a test statistic of -30.293, far below our cutoff of -2.862 for a .05 p-value. This suggests that we can reject the null hypothesis the data is not stationary and treat it as stationary.

When we examined individual aircraft, we found similar results to the ACF plots. Most aircraft yielded low ADF test statistics, suggesting we can treat the data as stationary. There were several ADF tests which yielded high test statistics and p-values, suggesting some aircraft trips were non-stationary. However, this was a very small number of aircraft. In the Delta Airlines data, 94% of all aircraft yielded a p-value less than 0.05 during the ADF Test. Since this was a small number of aircraft, we decided that we could treat the data overall as stationary.

My Contribution to the Project

Summary

My contribution to the project was to implement a sequence2sequence model. This deep learning method is unique in that it uses two separate RNN models. One is called an encoder which learns states based on an input sequence. The other is called a decoder and uses the states from the encoder to predict an output sequence one item at a time. I have never worked with sequence2sequence before and have limited experience with time-series/sequential machine learning problems so implementing this took a lot of research. I read a lot of literature on using this model architecture for NLP problems like natural language generation and machine translation. Researchers using this architecture for a sequential problem exactly like ours was impossible for me to find.

Percentage of Code Used from Internet

I followed this example very closely: link. It has around 100 lines of code including spaces and comments. My final script modling_seq2seq.py is 309 lines of code with comments and spaces. Most of the added 200 or so lines were to have more robust evaluation, load and save data and models, and graph performance. I probably modified 30-40 lines from the linked GitHub repository.

Lines from Internet	Added Lines	Modified Lines	Percentage
100	209	35	21%

How Sequence2Sequence works

There are two different "modes" for sequence2sequence models; training and inference. Using French to English machine translation as an example, during training, an English phrase is given to the encoder character by character, its states are then passed to the decoder with a special character that indicates the beginning of a French phrase. From here, the decoder makes a prediction and error is calculated by comparing the predicted character against the ground truth next character. Normally, the decoder uses its predictions recursively to generate character after character.

To prevent the decoder from making erroneous predictions in the sequence that would get worse as the French sentence goes on, we use something called teacher forcing during training. In this translation example, teacher forcing would be implemented by feeding the decoder the ground truth next French character no matter what the decoder predicts. This allows the decoder to properly learn the French sequences of characters. To exemplify this, we will translate the English word *go* to its French translation *va*. *Go* is fed to the encoder character by character. The encoder states and a special character ('\t' for example) is passed to the decoder to make a prediction. If the decoder predicts the letter 'r' when the ground truth character is 'v', on the next iteration the decoder is fed 'v' instead of the 'r'.

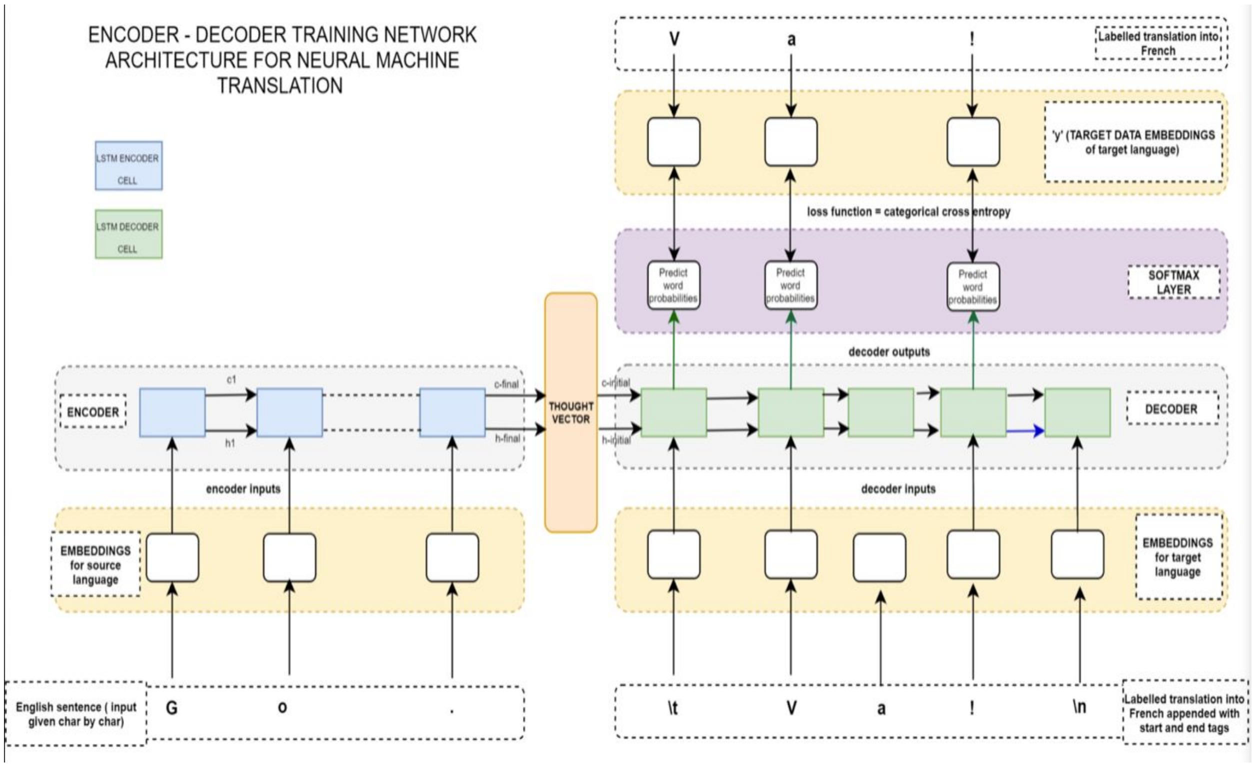

ENCODER - DECODER TRAINING NETWORK ARCHITECTURE FOR NEURAL MACHINE TRANSLATION
LSTM ENCODER CELL
LSTM DECODER CELL
V
a
!
Labelled translation into French
'y' (TARGET DATA EMBEDDINGS of target language)
loss function = categorical cross entropy
Predict word probabilities
SOFTMAX LAYER
decoder outputs
ENCODER
c1
h1
c-final
h-final
THOUGHT VECTOR
c-initial
h-initial
DECODER
encoder inputs
decoder inputs
EMBEDDINGS for source language
EMBEDDINGS for target language
English sentence (input given char by char)
G
o
.
\t
V
a
!
\n
Labelled translation into French appended with start and end tags

Figure 5. Sequence2sequence model in training mode using teacher forcing.

At inference time, this teacher forcing is not used and the model is free to recursively predict the next element of a sequence one at a time using the previously predicted character and the updated states from the decoder. In the next section we will show how sequence2sequence was applied to our problem domain.

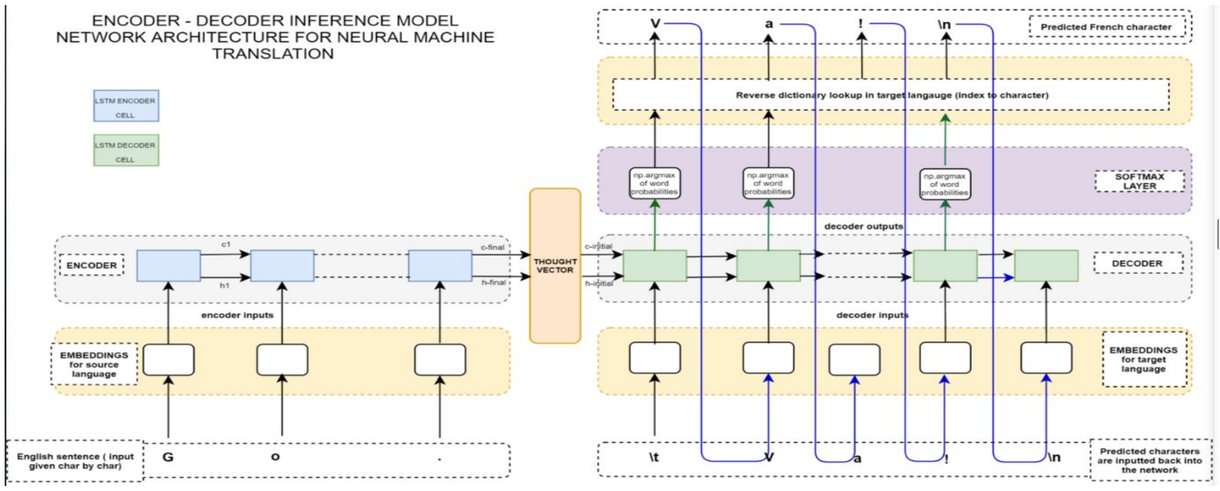

ENCODER - DECODER INFERENCE MODEL NETWORK ARCHITECTURE FOR NEURAL MACHINE TRANSLATION
LSTM ENCODER CELL
LSTM DECODER CELL
ENCODER
encoder inputs
EMBEDDINGS for source language
English sentence (input given char by char)
G
o
.
THOUGHT VECTOR
c-final
h-final
c-initial
h-initial
DECODER
decoder outputs
decoder inputs
EMBEDDINGS for target language
SOFTMAX LAYER
np.argmax of word probabilities
Reverse dictionary lookup in target langauge (index to character)
Predicted French character
Predicted characters are inputted back into the network
\t
V
a
!
\n

Figure 6. Sequence2sequence model in inference mode not using teacher forcing.

Applying Sequence2sequence to Our Problem

To apply the model to our problem, I experimented with various beginning and ending sequence lengths. For example, the Delta Airlines data with sequence length 25 could be divided into a beginning sequence of 5 and an ending sequence of 20 or a beginning sequence of 12 and ending sequence of 13. Ultimately I found that the model performed best with beginning sequence length consisting of all flights except for the last. The ending sequence in this case would just be the last flight in the sequence.

With defined beginning and ending sequences, I now had data analogous to the English and French sentences in the above example. My input to the encoder became all but the last airports in the sequence. My input to the decoder became the states from the encoder and a special starting character. This required padding the ending sequences with beginning character ‘1111’ and ending character ‘9999’. When the decoder receives ‘1111’ and the encoder states as its input, it predicts the next airport in the sequence.

Experimental Setup

We built, trained, and evaluated a wide-range of models including LSTM with one or two layers, GRU-based models, bi-directional LSTMs, 1D-convolutional models, and Sequence-to-Sequence models. Out of which LSTM & sequence2sequence produces best results for our dataset. Let's discuss in brief about the experimental setup of these models.

Data Preprocessing

Our RNN models work best with sequences of equal length, so a key part of our data preprocessing was splitting the airports visited into equal-length arrays. To accomplish this, we read in the DataFrame from the EDA_and_cleaning.py script, where each row includes the string of all airports visited by each aircraft. This str is then split into a Numpy array, and then broadcast through a function that takes the Numpy array, a window length, and a stride to generate an array of arrays each the length of the window. For example, if there were originally 7 elements [0,1,2,3,4,5,6] and the window is 3 and stride is 1, we would get

[[0,1,2],
[1,2,3],
[2,3,4],
[3,4,5],
[4,5,6]]

as the resulting array. We performed this processing for every unique aircraft and concatenated all the resulting sequences together. This results in an array with a shape of n (number of series) by the window length.

Next, we used the Keras tokenizer function to map the unique string elements representing airports into integers. After tokenization, we generated y (the target variable) by slicing off the last column of the resulting series of sequences. The target variable is also one-hot encoded to a binary representation. This data array will be our target for one-step ahead prediction. The remaining data is now in the shape of n (number of series) by the window length - 1 and will be our feature set. Finally, we split the data into a train and test set and save the resulting arrays for use in modeling.

Framework

Keras was used to implement all our RNN models. Keras is a user-friendly and simple to understand deep learning framework that the members of our team have experience using. Additionally, it allows for easy implementation of layers with GRU and LSTM units.

Model Implementation

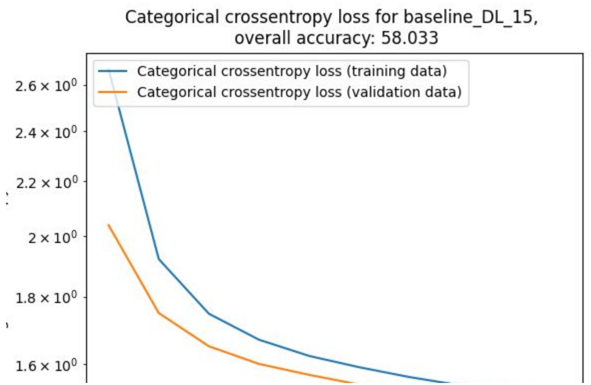
All models were built using Keras. We compiled our models with categorical cross entropy as the loss function and accuracy as the metric. We then fit the model, determined the accuracy, and then saved the model for further evaluation in the model_evaluate.py script.

Sequence2sequence

For our sequence2sequence implementation, we used categorical cross entropy as the loss function and accuracy as the metric. The encoder and decoder both use a single LSTM layer with 256 neurons. The model used sequences of flights in lengths 5, 10, 15, 25, and 50. All sequence2sequence models were trained for 10 epochs with a batch size of 64. The Adam optimizer was used with a 0.001 learning rate. The sequence2sequence implementation can be found in modeling_seq2seq.py and it includes training, testing, and evaluation all in one script.

Model Evaluation Metrics

To evaluate the performance of our model, we employed several different metrics. All models were saved as hdf5 files with filenames corresponding to the model name and run type



(airline and fixed sequence length, eg. “DL_15” uses data only from Delta airlines with a sequence length of 15). Additionally, the training history of each model was saved as a Pandas DataFrame for future retrieval and visualization.

All model evaluations occur in the model_evaluation.py script (except for sequence2sequence). In this script, required x and y data, models, corresponding history files and tokenziers are loaded from disk. Each model’s training history is then plotted to compare the training and validation loss to ascertain if the model overfit or underfit. The example on the right shows the training history for our baseline model on the Delta data with a sequence length of 15 (“baseline_DL_15”). In this plot, we can see that both losses drop and converge near the tenth epoch. Additional training may improve performance on the training data but would likely overfit the data and not improve the validation data. This supports our use of 10 epochs in training the baseline model.

Next, we determined the overall accuracy of each model. Simply put, this metric determines for each predicted next airport, how many times did the model get the correct answer divided by the total number of predictions. We also calculated the Cohen’s Kappa score, which is scaled from -1 (indicating no agreement) to 1 (complete agreement). Cohen’s Kappa is usually used to compare the results from two different manual annotations but can also be used to evaluate models in cases where we are concerned about the likelihood of randomly guessing affecting our accuracy. Because of its additional inclusion of uncertainty, Cohen’s Kappa score is a complimentary metric to overall accuracy.

In addition to these metrics, we generated a confusion matrix and a classification report to evaluate where our models did well and where they did poorly. As mentioned earlier in our EDA section, there is a risk that our models may only learn patterns associated with larger, busier airports. Since the airports have more flight activity, correctly predicting activity at these airports would inflate the accuracy of the model across all the data. To analyze if this was occurring, our team developed a scatter plot to show the precision and recall of a model for each airport. By sizing the scatterplot mark for each airport to the number of flights at that airport, we can create a visualization to help us understand how well the model performed across airports of different sizes.

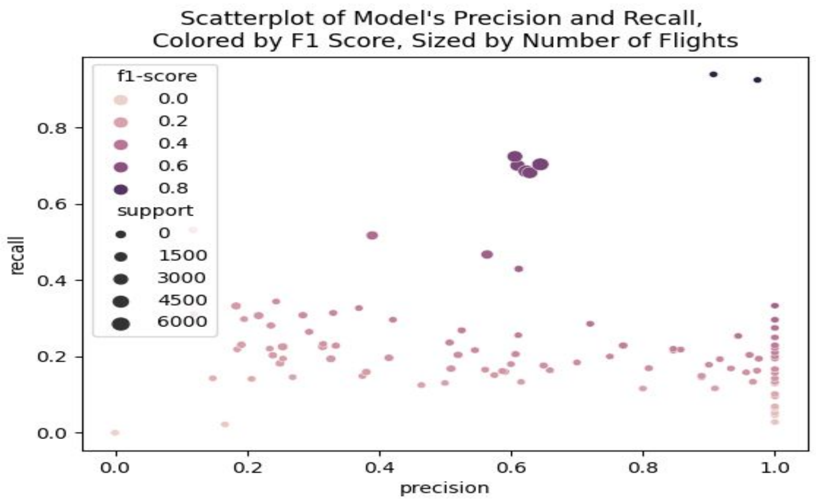


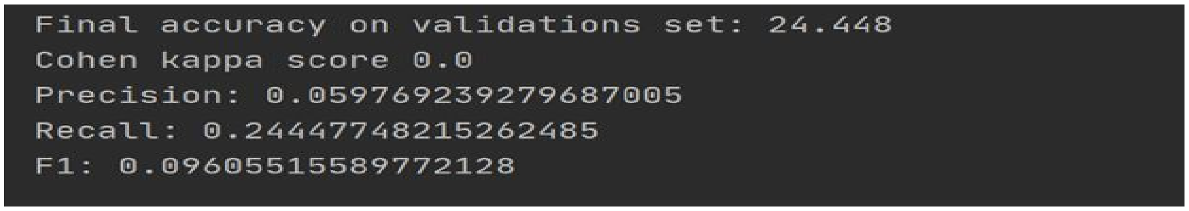
Figure 9. In this scatterplot from the baseline model using United Airlines data, we can see a tight grouping of the largest airports at around 0.6 precision and 0.7 recall. Smaller airports struggle to break 0.4 recall but many earn high precision scores.

Finally, our scripts recorded the overall accuracy and Cohen's Kappa score for each model against each subset of data in a log. This log allowed for our team to review how well a model performed and determine the best performing model.

Results

In order to evaluate the performance of the model, we generated dummy random sequences and fed this to our experimental set up and observed the results. This forms the base of the experiment to evaluate the other model performance.

Random Sequences Output:

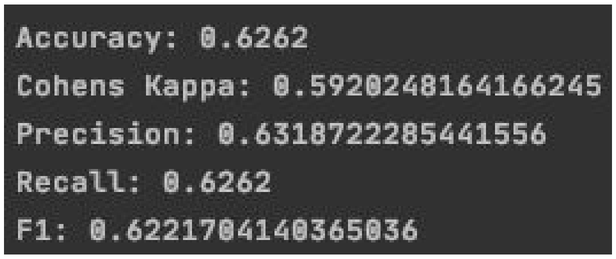


```
Final accuracy on validations set: 24.448
Cohen kappa score 0.0
Precision: 0.059769239279687005
Recall: 0.24447748215262485
F1: 0.09605515589772128
```

From the above result, we could see that the accuracy of the random sequences is close to 24%. The Cohen kappa score suggests that output is completely random. If any of our experimental networks produces accuracy less than this base accuracy, then the model is underfitting or performing bad.

Sequence2Sequence Method:

Airline: Delta
Sequence Length: 25



```
Accuracy: 0.6262
Cohens Kappa: 0.5920248164166245
Precision: 0.6318722285441556
Recall: 0.6262
F1: 0.6221704140365036
```

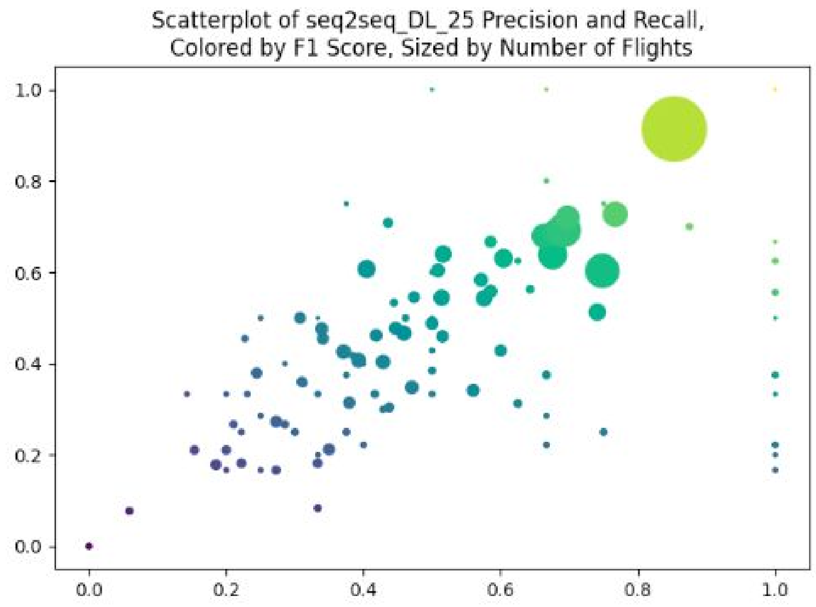
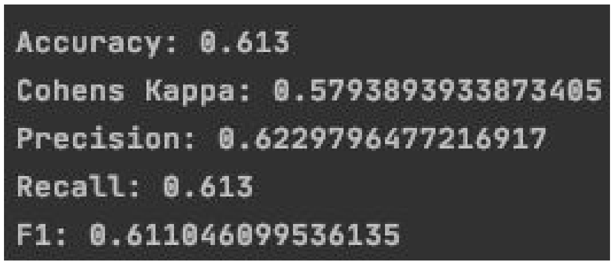


Figure 14. Scatterplot Precision vs Recall - Seq2seq network on Test set

Airline: Delta
Sequence Length: 50



```
Accuracy: 0.613
Cohens Kappa: 0.5793893933873405
Precision: 0.6229796477216917
Recall: 0.613
F1: 0.611046099536135
```

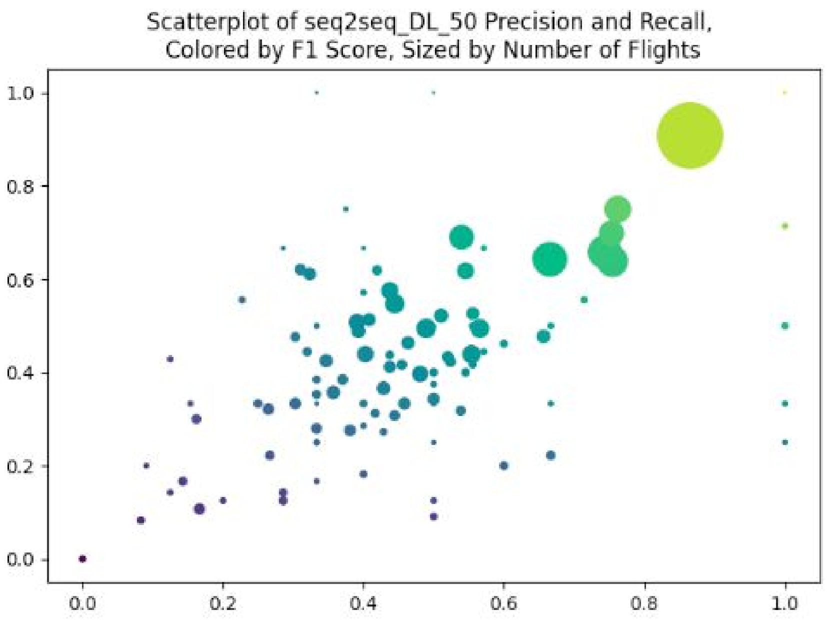
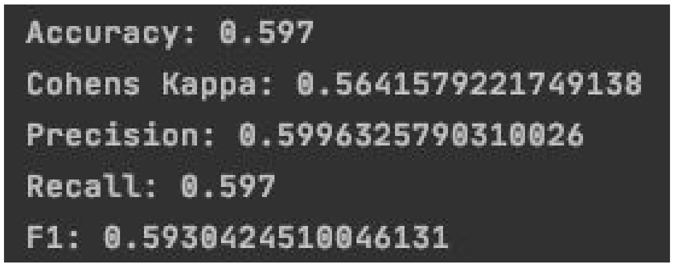


Figure 15. Scatterplot Precision vs Recall - Seq2seq network on Test set

Airline: Delta
Sequence Length: 15



```
Accuracy: 0.597
Cohens Kappa: 0.5641579221749138
Precision: 0.5996325790310026
Recall: 0.597
F1: 0.5930424510046131
```

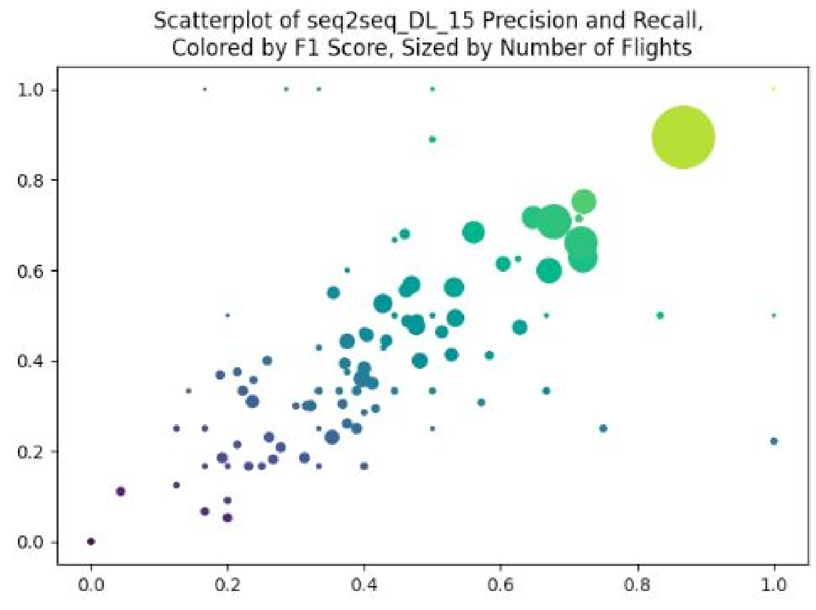
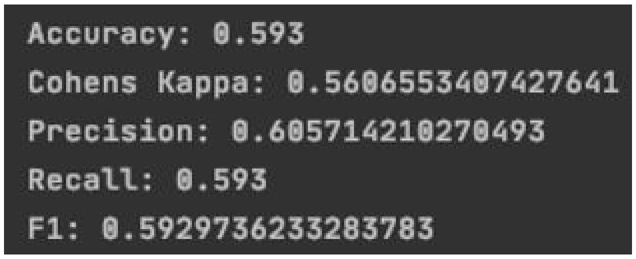
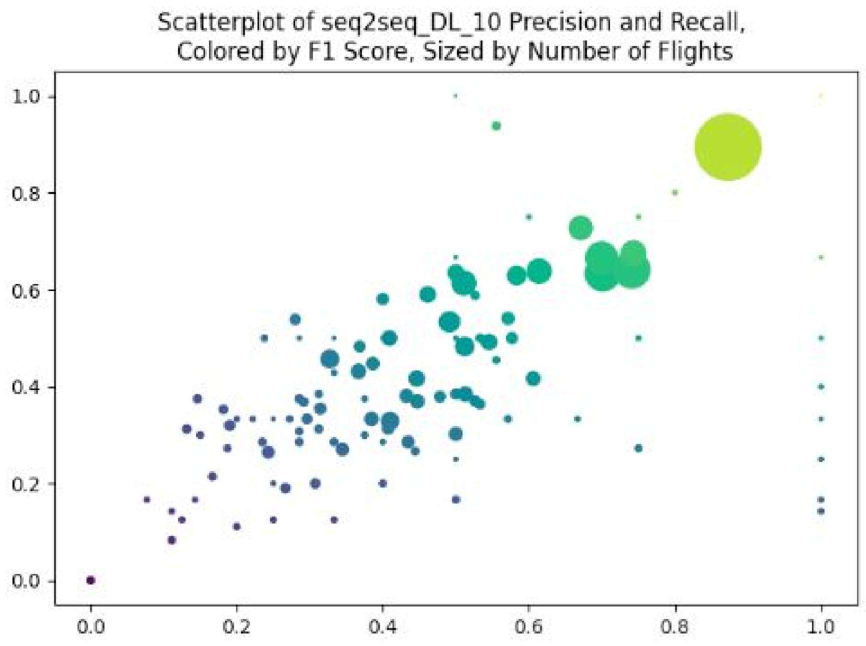


Figure 15. Scatterplot Precision vs Recall - Seq2seq network on Test set

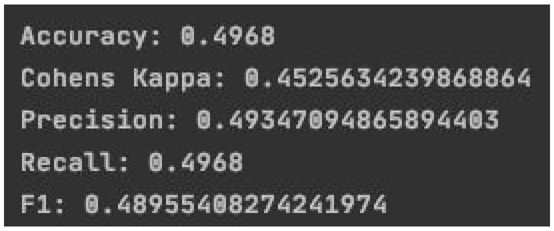
Airline: Delta
Sequence Length: 10



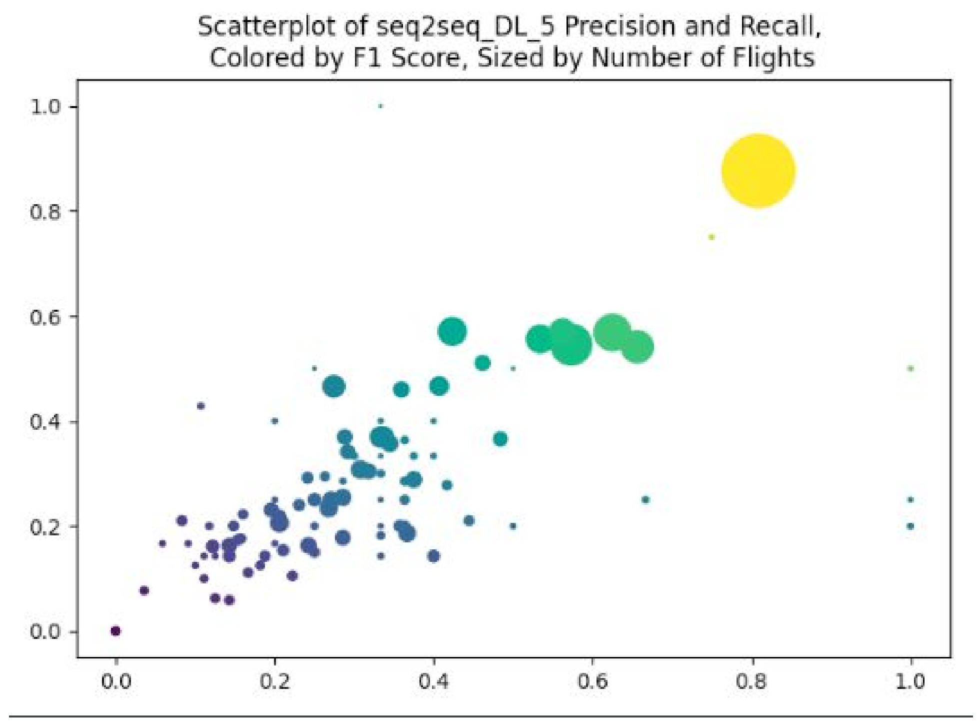
```
Accuracy: 0.593
Cohens Kappa: 0.5606553407427641
Precision: 0.605714210270493
Recall: 0.593
F1: 0.5929736233283783
```



Airline: Delta
Sequence Length: 5



```
Accuracy: 0.4968
Cohens Kappa: 0.4525634239868864
Precision: 0.49347094865894403
Recall: 0.4968
F1: 0.48955408274241974
```



Summary

The table below shows the results from using sequence2sequence on five different sequence lengths.

Table 2. Results from all seq2seq models.

Model	**Accuracy**	**Cohen's Kappa**	**F1**	**Sequence length**
seq2seq	0.6262	0.5920	0.6222	25
seq2seq	0.6130	0.5794	0.6110	50
seq2seq	0.5970	0.5642	0.5930	15
seq2seq	0.5930	0.5607	0.5930	10
seq2seq	0.4968	0.4526	0.4900	5

Summary and Conclusions

Evaluating Stationarity and Autocorrelation

Our group took multiple steps to verify that our data was a relatively stationary dataset with an embedded signal we could learn through our models. In addition to the described work with ACF plots and stationarity tests with the Augmented Dickey-Fuller test, we also applied a truly randomly generated dataset against a trained model and received far lower scores than on the actual data. This suggests there is an underlying pattern.

Repeated Sequences

During our analysis, our team noticed repeated elements in a sequence, eg an aircraft departing one airport to visit that same airport. In our research, this was usually accompanied by a time gap of several days. This activity could be related to flight maintenance, erroneous data, or non-commercial flights missing from our data. After discussion with our team, we decided it would be best to keep the data as is with these anomalies rather than correct them and introduce additional, unknown biases.

Sequence Length

In Table 2, we see that sequences of length 25 or 50 yielded the best results from our models. Our group theorized that sequences of more than 50 airports might perform better but would be challenging to use in a production setting. The longer the sequence, the less robust our model would be in picking up on drastic changes to the airport patterns.

Looking at Other Airlines

To limit the scope of the project, we only focused on Delta Airlines flights. As seen in Figure 1, Delta has a central hub in Atlanta which accounts for over 25% of all of its activity. This led to our models performing very well on predicting when a flight would go to Atlanta or other busy airports, but the models struggled to perform well on smaller airports. We tried to model United and American Airlines data and found that these less centralized datasets did worse than the Delta data with our models.

Conclusions Around Sequence2sequence

Our group expected sequence2sequence to perform the best on this task but it was outperformed by a simple single layer LSTM. As explained previously, different sequence lengths were tested with sequence2sequence but all hyperparameters stayed the same. Tuning the hyperparameters after finding the best sequence length would probably improve seq2seq's performance.

Personal Conclusions

This was my first time implementing sequence2sequence and I learned a lot in the process. My experience with machine learning on sequential data is limited so I feel like this project allowed me to learn a ton. Considering that I used the English to French sequence2sequence example, I borrowed a lot of code for the model.

References

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html?highlight=accuracy#sklearn.metrics.accuracy_score

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

https://towardsdatascience.com/neural-machine-translation-using-seq2seq-with-keras-c23540453c74

Understanding LSTM Networks -- colah's blog

Illustrated Guide to LSTM's and GRU's: A step by step explanation | by Michael Phi | Towards Data Science

Appendix A: Code for Project

Link to all code on GitHub: https://github.com/redhairedcelt/Final-Project-Group1