Predicting Aviation Weather Conditions for Flight Planning

By Nate Simon and James Wang

CS230, Spring 2020

Introduction

We are applying deep learning to historic weather data at airports in the US to make informed short-term predictions (forecasts) of weather conditions. We are using weather data in the form of a METAR (Meteorological Aerodrome Report). These reports are published by over 900 ASOS (automated surface observing system) sites in the US and historical reports are available.

Task

Our task is to train a model on past data for short-term weather prediction. Specifically, if the weather is likely to transition from ‘VFR’ (visual flight rules - i.e., good) to ‘IFR’ (instrument flight rules - i.e., bad) within a specified prediction window.

Our labeling scheme is the following:
X -> Numericalized Metar Data for T consecutive historical reports.
Y -> (1) When the weather state transitions from VFR to IFR at any point in the prediction window. (0) Otherwise.

The goal is to input a novel example X and receive an informed prediction of the VFR->IFR transition (worsening weather) based on previous weather patterns. This could be used in real-time at small airports without weather prediction equipment for flight planning.

Related Work

Deep learning has been applied to weather forecasting, and METARs have been used as a data vehicle for such efforts. The Argonne National Lab[1] showed the promise of deep learning in speeding up large-scale weather forecasting, as an “accurate alternative to physics-based parameterizations.” A team at Tel Aviv University applied dynamic convolutional layers to radar images for short-range weather prediction,[2] and another team at UT Arlington used METAR data to construct an RNN for wind speed forecasting.[3] These efforts used

[1]Wang et al. "Fast domain-aware neural network emulation of a planetary boundary layer parameterization in a numerical weather forecast model."
[2]Klein et al. “A Dynamic Convolutional Layer for Short Range Weather Prediction.”
[3]Ghaderi, Amir, Borhan M. Sanandaji, and Faezeh Ghaderi. "Deep forecast: deep learning-based spatio-temporal forecasting." arXiv preprint arXiv:1707.08110 (2017).

METARs to collect data on a single measurement (such as wind speed), rather than using sequences of *entire* METARs as the data input to predict general weather conditions. To the authors’ knowledge, this specific effort has never been implemented (it is novel).

Dataset and Features

We access this data through the Iowa State University Iowa Environmental Mesonet[4]. Available METAR records go back to 1928.
A raw METAR is in the following form:
KPAO 252247Z 13007KT 10SM FEW030 OVC120 11/06 A2974
This contains information on the airport location, time and date, wind direction and speed, visibility (distance), cloud layers coverage and altitude, and other variables. For this data to be interpretable to a neural network, we numericalize it, label it, and then reshape it to the input form X. To numericalize our data, we map continuous measurements between 0 and 1, and classifying features, such as cloud coverage and weather codes, to a one-hot encoding. (See Appendix-1 for more details).

Although this varied somewhat between iterations of our network design, the overall structure of our example matrix was at least 5,000 examples, where each example was composed of approximately a day of data (data is reported every ~10 mins at weather stations) and was used to make a prediction four hours into the future. The amount of data used to train the model can essentially be arbitrarily large, since each station has data logs extending for several decades.

Methods and Results

In the last quarter, we have prototyped several neural network architectures and hyperparameter configurations. Initially, we started with a single logistic regression neuron that was performing binary classification on examples labelled 1 if weather was IFR (bad) and 0 if weather was VFR (good). After training this model on a year of data, we were surprised to see that our model was performing very well - with a training accuracy, test accuracy and F1 score of 96.7%, 93.2% and 0.83, respectively. However, we realized that these unexpectedly high scores were because our initial labelling scheme was making the weather prediction task too easy and that our model was simply looking at existing weather conditions and guessing that these conditions would persist into the future.

[4]“Iowa State University Iowa Environmental Mesonet,” n.d.

In the next iteration of our network design, we devised a new labelling scheme that would predict *changes* in weather conditions: (1) if conditions began as VFR and *changed to IFR* within the prediction window (0) otherwise (no change, or IFR-> VFR). As expected, the performance of the single neuron network decreased drastically with this more challenging prediction task, producing a training accuracy of 90%, a test accuracy of 79% and an F1 score of 24%. The discrepancy between accuracy and performance is explained by data imbalance. Since VFR weather is typically much more common than IFR weather, only a small fraction of examples are positive (1) labels in a typical data set. As a result, our simple model learned to label nearly all predictions as (0)s, which resulted in high training/test accuracy but low precision and recall.

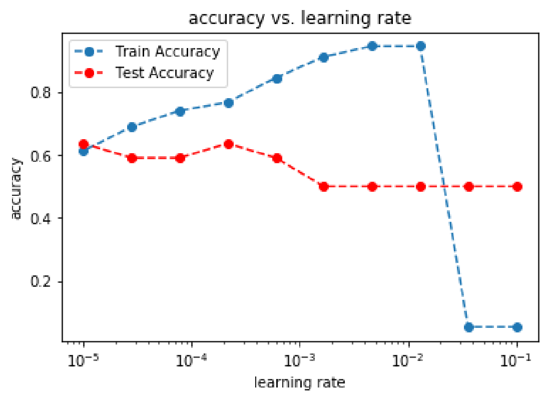
Based on the recommendations of our Project TA Shubhang, we addressed the data imbalance problem by downsampling. Before shuffling our data into training and test sets, we randomly sample negative (0) examples to match the number of positive (1) examples. Using the same logistic regression model as before, we were able to produce a training and test set accuracy of 93% and 75% (respectively) and an F1 score of 72%.

Encouraged by these results, we moved forward to implementing a deeper neural network model in TensorFlow. Our early experiments used a three layer network with 6, 3, and 2 neurons in the first, second and third layers. These layers were performing LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SOFTMAX calculations to predict weather outcomes[5]. With various combinations of different learning rates, minibatch sizes, and dataset sizes we were unable to match the relatively high performance of our logistic regression model and could only produce a maximum test accuracy of ~60%.

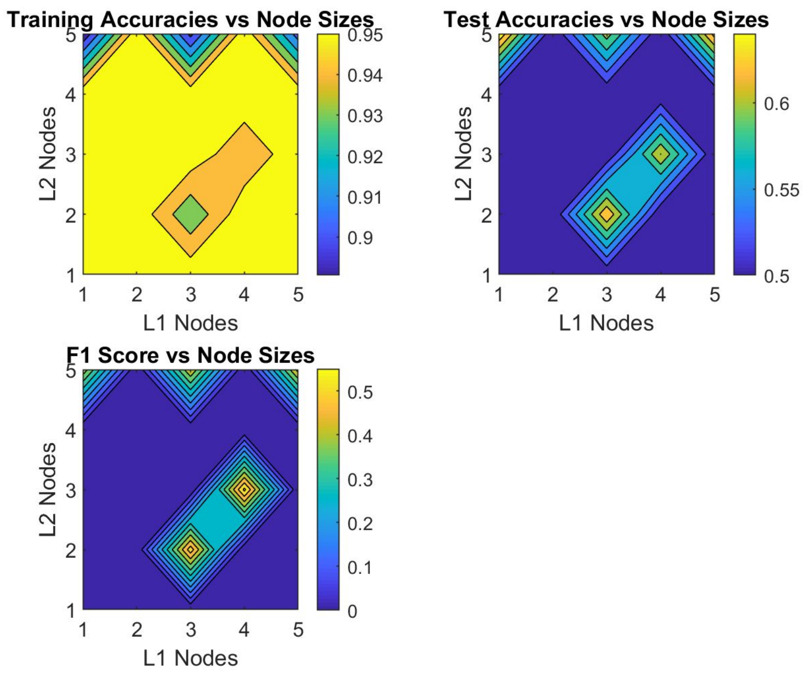
To improve the performance of our TensorFlow model, we implemented L2 regularization and downsampling in every epoch instead of just performing downsampling on the whole dataset before it was fed into the network (which resulted in ignoring most of the examples labelled 0). We hoped that L2 regularization would reduce the overfitting evident in our early experiments and that epoch-wise downsampling would expose the network to all the examples labeled 0 in the dataset when iterating. These improvements improved our network’s performance only marginally: [train: 0.86, test: 0.7]. (See Appendix-2 for a diagram of our iterative approach, cost-plot comparisons in Appendix-3.)

[5] Heavily based on the TensorFlow Tutorial programming assignment from Improving Deep Neural Networks_Hyperparameter tuning, Regularization and Optimization.

Next, we began the arduous tasks of tuning hyperparameters. The hyperparameters that we focused on were learning rate and neuron amount in each layer. We first estimated the ideal learning rate for our existing three layer model (with 6, 3, 2 neurons) by iterating through different possible learning rates in a for loop, incrementing the learning rate on the log scale every iteration.



From this plot, we chose a learning rate of 5*10^-4 and proceeded to iterate on different neuron sizes in the first and second layers of the network. For both of these layers, we iterated from 1-5 neurons, resulting in the following performance visualizations:



This suggests that a model with 3:2:2 nodes - producing a train accuracy of 94%, a test accuracy of 62% and and F1 score of 0.56 - would perform the best. With this network size, we iterated again through different learning rates.

Although different learning rates could produce significantly higher test accuracies (~80%) than that of the model described above, other learning rates did not produce an F1 score higher than 0.56. As such, the hyperparameters of our final TensorFlow model are as follows:

Learn rate: 1.0E-5	Epochs: 50	Batch size:64	Lambda: 1.0e-5	Dataset: 2.9 yr, 11,346 examples	Examples: 48 hrs long, 4 hour prediction window

The hyperparameters of our logistic regression model with downsampling and L2 regularization were the following:

Learning rate: 0.005	Epochs: 1000	Dataset: 1 year, 5273 examples	Examples: 48 hrs long, 4 hour prediction window

Conclusion

At the conclusion of our work, we found that our simple logistic regression model significantly outperformed our layered network model, even after iterating through many different network configurations. We currently do not have enough information to definitively say why the single neuron is outperforming a larger network, but it could be because METAR weather data is simple in nature and tends to be overfit by larger and more complex networks.

More troubleshooting is certainly necessary. Weather is a complex and sequential distribution, which we'd imagine would best lend itself to be modeled by a more complex neural network rather than a single logistic regression neuron. We believe that working further on the TensorFlow network would be worthwhile. For instance, the next step for troubleshooting the network could be to make sure that the network is actually learning weather features in the way that is typical for neural networks - that is to mean that different neurons are more or less activated by specific kinds of weather data. To test this, we could group weather examples into categories that represent different weather phenomena (very cloudy, clear, rainy, snowy etc) and feed each weather group into the network to see if the activations of different neurons prefer specific weather features.

Since weather is sequential, an RNN might also be the best suited architecture to make weather predictions and could be investigated further.

All of our code for this project can be found here:
https://github.com/JamesW97/CS230-Project

Acknowledgements and Contributions

Firstly, we’d like to thank our Project TA, Shubhang Desai, whose ML wisdom and good humor was invaluable to our learning (and sanity). We’re also grateful to Professor Ng and the staff of CS 230, who have crafted an engaging and accessible learning experience for us all, even given the more than troubling circumstances of the past quarter.

In terms of contributions of team members, James Wang contributed most of the team’s good looks and charm. He also wrote the data collection/formatting code as well as data shuffling/down sampling helper methods, initial TensorFlow model, and network size tuning methods.

Nate Simon wrote the initial logistic regression and downsampling implementations as well as the learning rate tuning method and data visualization methods. He also made the final video and wrote the majority of the milestone reports, which have been conveniently chopped up and reformatted into this final report.

References

Ghaderi, Amir, Sanandaji Borhan, and Faezeh Ghaderi. “Deep Forecast: Deep Learning-Based Spatio-Temporal Forecasting.” *Arxiv.Org*, n.d. https://arxiv.org/pdf/1707.08110.pdf.

“Iowa State University Iowa Environmental Mesonet,” n.d. https://mesonet.agron.iastate.edu/request/download.phtml.

Klein, Benjamin, Lior Wolf, and Yehuda Afek. “A Dynamic Convolutional Layer for Short Range Weather Prediction.” *IEEE Xplore*, n.d. http://openaccess.thecvf.com/content_cvpr_2015/papers/Klein_A_Dynamic_Convolutional_2015_CVPR_paper.pdf.

Wang, Jiali, Prasanna Balaprakash, and Rao Kotamarthi. "Fast domain-aware neural network emulation of a planetary boundary layer parameterization in a numerical weather forecast model." Geoscientific Model Development (Online) 12.10 (2019).

Appendix

1) Numericalizing our input X

i) How we numericalize continuous data.

Measure	**Raw**	**Numericalizing Method**	**Result**
Month/Day	Dec 1st	Fraction of year	0.907
Time	2:47 AM	Fraction of day	0.116
Temperature	10 C	Convert to K, div by hottest recorded temp	0.858
Visibility	10 sm	Divide by 10 sm	1.000

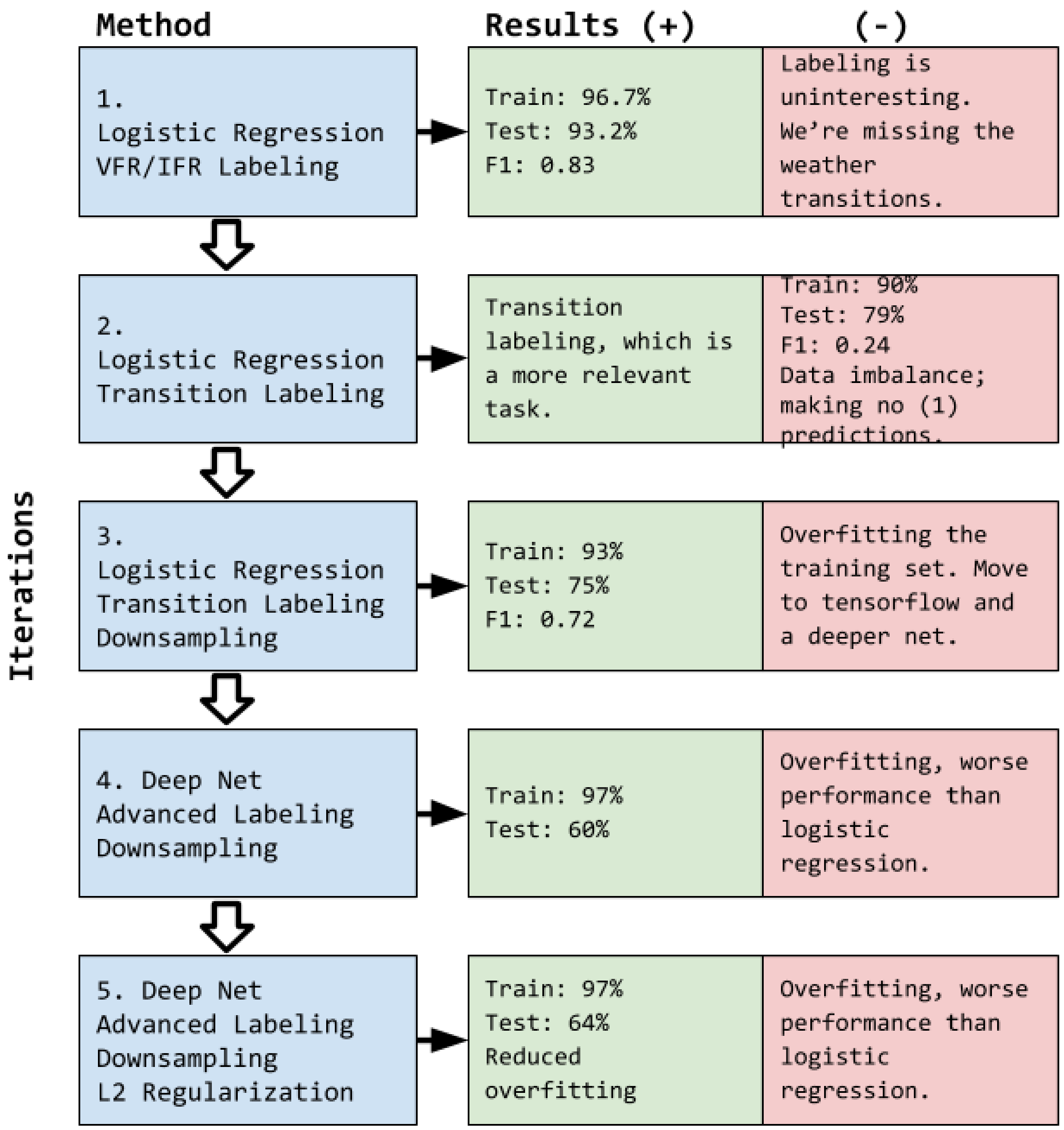
ii) For classifying features, such as cloud coverage and weather codes, we used one-hot encoding. (See below for example, cloud coverage is ‘broken.’)

Cloud	CLR	SKC	FEW	SCT	BKN	OVC	OVX
Encode	0	0	0	0	1	0	0

Given the number of classifying features in a metar (including rain, thunderstorms, etc) we currently have 45 one-hot encodings, and will need to expand this to include countless permutations, such as frozen rain, blowing snow, etc.

2) Diagram of our iterative process

The following chart is a high-level look at the iteration process from Methods 1 to 4, based on the positive (+) and negative (-) qualities of each method.

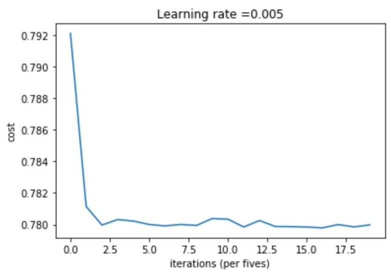

Method
Results (+)
(-)
Iterations
1. Logistic Regression VFR/IFR Labeling
Train: 96.7% Test: 93.2% F1: 0.83
Labeling is uninteresting. We're missing the weather transitions.
2. Logistic Regression Transition Labeling
Transition labeling, which is a more relevant task.
Train: 90% Test: 79% F1: 0.24 Data imbalance; making no (1) predictions.
3. Logistic Regression Transition Labeling Downsampling
Train: 93% Test: 75% F1: 0.72
Overfitting the training set. Move to tensorflow and a deeper net.
4. Deep Net Advanced Labeling Downsampling
Train: 97% Test: 60%
Overfitting, worse performance than logistic regression.
5. Deep Net Advanced Labeling Downsampling L2 Regularization
Train: 97% Test: 64% Reduced overfitting
Overfitting, worse performance than logistic regression.

3) Additional Model Performance Values

For Node Sizes: 6,3,2

288 LPE, 24 IBE (Lines Per Example, Increments between examples)

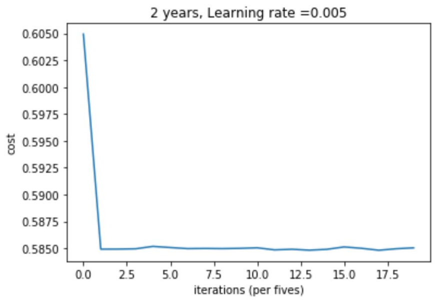
3 years: (1/1/2016 - 1/1/2019)



0.5 test

0.4 train

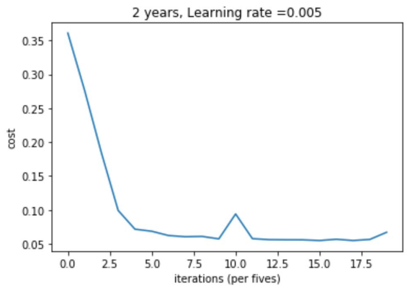
2 years: (1/1/2017 - 1/1/2019)



Train Accuracy: 0.50

Test Accuracy: 0.47

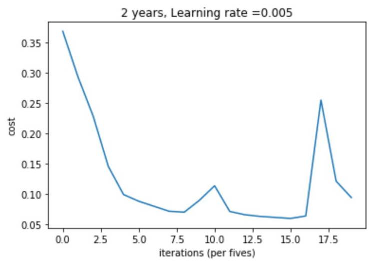
1 year: (1/1/2018 - 1/1/2019) - 100 epochs



Train Accuracy: 0.97

Test Accuracy: 0.6

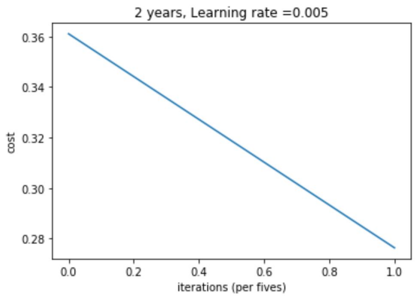
With lambd = 0.7 regularization:



Train Accuracy: 0.97

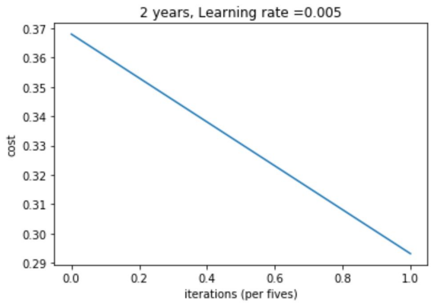
Test Accuracy: 0.64

1 year: (1/1/2018 - 1/1/2019) - 10 epochs



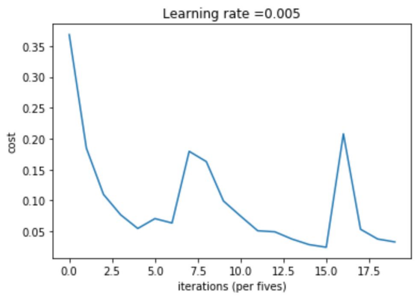
Train Accuracy: 0.86
Test Accuracy: 0.7

With lambd=0.7 regularization



Train Accuracy: 0.82
Test Accuracy: 0.7

For Node Sizes: 25,12,2
288 LPE, 24 IBE (Lines Per Example, Increments between examples)
With lambd=0.7 regularization



Train Accuracy: 1.0
Test Accuracy: 0.5