Final Report

Project title:

Weather wizard

Names of group members:

Wenzheng Zhao
Mohammed Almousa
Manpreet Singh
Yuqi Nai

Dataset(s) we used:

In the process of designing and developing our weather prediction model, we aimed to provide practical and significant insights by predicting three primary weather conditions: cloudy, sunny, and rain. These conditions were selected based on extensive surveys and interviews, and their tangible impact on day-to-day lives. Furthermore, these are the weather conditions of utmost concern to the majority of people.

During our initial search for suitable datasets, we considered various weather datasets available online. Each of these potential datasets incorporated different parameters and labels. Our primary criteria were that the dataset should have measures of temperature, pressure, and humidity, which aligns with the sensors embedded in our nano kit. In our search for datasets, we came across few and here we list them to see which one or ones we could use. The table below show each dataset and it shows if the 3 parameters are available:

Name/parameters	Temperature	Pressure	Humidity
NOAA GHCN	✓	✓	✗
NCEI CDO	✓	✓	✓
ECMWF	✓	✓	✓
Weather Istanbul Data	✓	✓	✓
London Weather Data	✓	✓	✗

Historical Hourly Weather Data 2012-2017	✓	✓	✓

We shortlisted several datasets, including those from Istanbul, London, and the 'Historical Hourly Weather Data 2012-2017'. These datasets were given particular attention due to their similarity to the climate and rain frequencies in Seattle, the geographical location of our interest. However, one caveat with the London dataset was the absence of humidity measures.

Our first choice was the Istanbul dataset, as it satisfactorily met our requirements. Our initial experiments with this dataset yielded promising results, achieving over 75% accuracy using only the three aforementioned sensors. However, during the further refinement and optimization phase of our project, we noticed that the Istanbul dataset, despite its initial performance, did not translate well when applied to Seattle's weather conditions. The crux of the issue was the significant imbalance in the dataset regarding the weather conditions, particularly for cloudy and rainy conditions. These conditions only accounted for 20% and 3% of the Istanbul dataset respectively, which is not representative of Seattle's weather. We also observed a significantly higher number of readings measured for clear weather (over 80% of the data) which explains why the test accuracy was high on only the 3 readings resulting in a misleading imbalanced model.

Recognizing the limitations of the Istanbul and London datasets, we reconsidered the 'Historical Hourly Weather Data 2012-2017' dataset, which we initially shortlisted. This comprehensive dataset records the hourly weather conditions of major cities in the US and Canada. Importantly, it includes the desired measures of humidity, temperature, and atmospheric pressure, and offers a sufficient distribution of the three weather types for model training. Using this dataset, we combined the Seattle data from two other cities in the dataset, San Diego and Portland. Adding San Diego resulted in having an imbalanced weather, so we approach it by undersampling the majority class to get a balanced model.

Approach:

To create the neural network the dataset we selected will be trained on, we started off with an existing larger model that had been trained on weather data in a different country. The original neural network had a large number of layers (greater than 10) with 30 neurons each that were fully connected between each of the layers. We first tried modifying the larger model and changing the number of its inputs to what we were able to measure with the Arduino board. This approach resulted in a very large model with an unreasonable accuracy. During this process we also recognized that our dataset was unbalanced for the amount of data per label. We had twice the data for sunny/clear conditions as we did for cloudy and raining conditions. In order to combat this, this took just the data for the sunny/clear conditions, and removed samples at random from the sunny/clear dataset. We removed just enough of the samples to roughly balance the number of data samples per label.

Balancing the training and validation data did gain us an accuracy improvement but the model was still too large to fit onto the Arduino. Next, we started removing the dense, fully connected, layers from the model to see what kind of impact it would have on the accuracy of the model. Surprisingly, we were able to get down only three layers in our model while seeing a minimal in accuracy. Each of the three layers we ended up using had multiple of three as its number of fully connected neurons. First layer, after the input layer, has 9 neurons, and the second layer has 27 neurons, and finally the last layer, which is also the output layer, has a size of 3 neurons. The 3 neurons on the output layer represent the number of classes we wanted to classify the input to.

Lastly, we wanted to deploy our trained model to the Arduino, which we were able to do with the EloquentTinyML library. We decided to use this library for model deployment instead of TFlite library because the complexity of the code to run inference on the model was much lower, we could have concise code that was easy to read and understand. One place in the Arduino deployment where we had to come up with an algorithm was on the output of the ML model. We had to come up with an approach to interpret the output of the model into something human readable, we wanted something that was fast and didn't take too long. To summarize, we accomplished the integration of deep models in Arduino by utilizing the EloquentTinyML library. Through the utilization of the generated model.h TinyML model, we were able to perform inference on the Nano 33. Additionally, we successfully implemented real-time communication via serial port, enabling on-site visualization and experimental verification on the upstream device

Problems we worked on:

During the process of implementing our ideas and deploying the project, we encountered several issues. Most of the problems were resolved through collaboration, but we spent a significant amount of time on the following issues, mainly because they were either difficult to reproduce consistently or were non-code-related problems that couldn't be solved programmatically.

1) Finding the good and balanced databases with the 3 parameters we have sensors for
    a) Our goal is to predict the weather in Seattle, and for that purpose, we need a suitable dataset. We extensively searched for potential datasets, but most of them either lacked the necessary sensor data or didn't meet our desired weather distribution conditions. Some datasets that met our requirements had extremely unbalanced data, which would significantly impact the performance of our ML model deployment. After conducting continuous searches, we finally made the decision to utilize the "Historical Hourly Weather Data 2012-2017" dataset. This comprehensive dataset provides records of hourly weather and conditions for major cities in the US and Canada. Importantly, it includes the essential measurements of humidity, temperature, and atmospheric pressure, while also

offering a huge amount of data points (over 3M) for (sunny, rainy, cloudy) for training our model. Based on these factors, we finally selected this dataset for our project. We also undersample the major class which had about 30% more data to have comparable data points for the 3 classes.

2) Organizing the dataset
    a) After finding the dataset we believed would work well with our application, it's now our focus how to put this data together. The data is scattered around 4 excel sheets. So we decided to have them all in a dataframe and check the inputs to make sure they do not have null data (which it did) otherwise that would break the training. We also made sure that the units used are matching the unit we measure for, thus, we had to convert the temperature to degree celsius. After that we shifted to the output and decided which among the 50+ labels they had are suitable for our 3 class prediction and finally see the data points we have per class to insure classification balance.

3) Writing the Arduino code and figuring out our sensor missing the humidity
    a) After confirming that our board had the required sensors, we began writing the hardware program for Arduino. However, we encountered errors related to the model library and spent a considerable amount of time debugging the code between Arduino and Python for the model integration. We resorted to function by function debugging to identify any issues with the code as whole. We were under the impression that all Arduino BLE 33 Nano has a humidity sensor but after continuously failing to sense that, we found out that although the board is supposed to have one, Arduino decided to not include one for the tiny ML board line. After getting boards with humidity sensors, our code was continuously breaking and disconnecting from the board. We believed the issue was by calling a few external functions that might have pushed the computing or the memory limit for the board, thus, we decided to write the function that translates the prediction into words as If statements to have less computing which worked.

4) Coming up with good neural network structure
    a) The number of dense layers and the number of neurons have been definitely a challenge to come with the minimum number of parameters but yet, keeping the accuracy the highest. We started by having more than 10 dense layers with as high as 128 neurons then we made our way down layer at a time or reducing the number of neurons. Every time we do a change, we test to see the effectiveness of the action as we test it against the accuracy of the model. Until we get to the lowest number of neurons and layers that keep our accuracy to about 55%. Trial and error was the approach not by us only, but by many people in the ML community as the puzzle to answer the question of the perfect design still unknown but only based on the application and the number of parameters we are feeding the network.

5) Using libraries we were not familiar with
    a) When deploying the model to the Arduino, we used a different ML library than the one we had been using in the labs called EloquentTinyML. While EloquentTinyML was easier to use in terms of the amount of overhead code required to instantiate the model when compared to the official tensorflow lite libraries. We found that the latest version of the EloquentTinyML library had API changes that made the code we had not work as expected. We kept chasing various errors as we progressed through the implementation phase. After further research we had to downgrade the version of this library in order to get it to run inference on the TFlite model.

Parts of Project Implemented:

- Project Proposal: We developed a comprehensive project proposal that outlined the objectives, research questions, and methodologies for our project.

- Project Pipeline Design and Management: We designed an effective project pipeline that guided the execution of our project and ensured smooth collaboration among team members by assigning different tasks to various team members according to the project process.

- Dataset Selection and Evaluation: We conducted thorough research to identify relevant datasets for our project. We evaluated these datasets based on criteria such as data quality, size, and suitability for model training purposes.

- Dataset Cleaning: We performed data cleaning to ensure the integrity and accuracy of the collected datasets. This involved techniques such as removing outliers, handling missing values, and normalizing features.

- Model Pipeline Architecture Design and Evaluation: We designed a comprehensive model pipeline architecture to address the problem. Our pipeline included several stages, each playing a different role in the overall process. These stages comprised data import, cleaning, machine learning model design, evaluation, quantization, and conversion of the model into a header file format suitable for deployment on tinyML devices.

- Model Quantization: We deployed model quantization techniques to reduce the model's size and improve its inference speed without sacrificing performance.

- Model Deployment: We took responsibility for deploying our trained models in real-world scenarios. We developed a seamless deployment pipeline and ensured compatibility with different deployment platforms.

- Real-Time Demonstration and Verification of tinyML Terminal Equipment: We conducted real-time demonstrations and verification of tinyML terminal equipment. This allowed us to assess the performance and suitability of our models for resource-constrained devices.

Experimental results and analysis:

The model has the following characteristics:

- The first Dense layer takes inputs of shape NB_features which is equal to 3, and has 9 neurons with ReLU activation function.
- The second Dense layer has 27 neurons, also with ReLU activation function.
- The final layer is a Dense layer with NB_classes which is equal to 3 neurons, and it uses softmax activation, which makes it suitable for multi-class classification.

The model is compiled with the Adam optimizer, categorical cross entropy as the loss function, and accuracy as the metric for model performance. It has 390 parameters in total and all of them are trainable. There aren't any non-trainable parameters after compiling the model. After splitting the dataset into training set 85% and test set 15%, running 200 epochs we got the final accuracy is about 55%.

We also use dynamic range quantization to compress the model for a smaller size. The size of the model before quantizing is 36288 bytes and is 3.6KB after quantization. Since the model itself doesn't have many parameters, we decided not to add a pruning step and this resulted in a very small reduction in accuracy of the model despite being quantized.

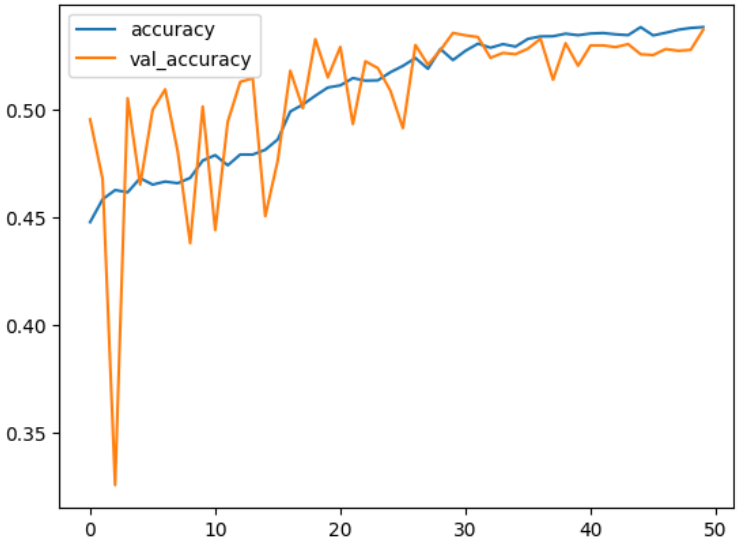

Figure 1. Accuracy plot

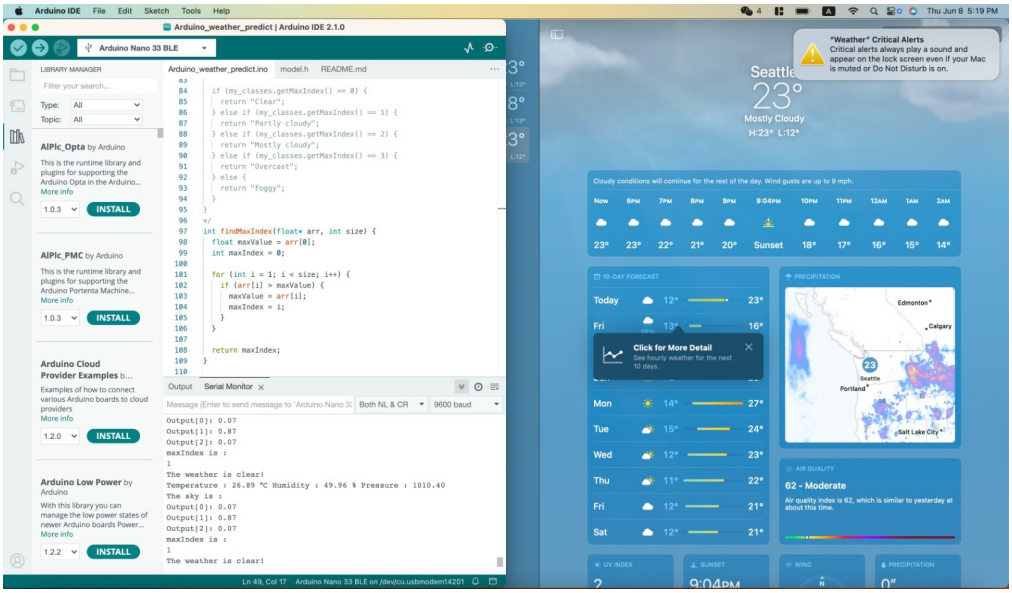

Figure 2. Field test results

Our model as we stated in our presentation is collecting live feed of 3 measurements and based on those measurements classify if the weather is more likely to be clear sky, cloudy, or rainy. The goal of massive deployment of such a device in smart cities would result in a more real time accurate reading. As our figure 2 suggests, the satellite data was predicting mostly cloudy weather where our model classified it as clear (sunny) weather. The real condition was more as what our model predicted, the sun was shining and it provided a more accurate weather condition.

Future Improvements:

We noticed that despite having a large and a relatively diverse dataset to train with, we weren't able to achieve an accuracy higher than 55%. Our current hypothesis since we're only giving the model three inputs, these inputs must not be enough for the neural network to make an accurate prediction. This model also does not take the time of day into account. The model is trained in hourly reported data, meaning the model is predicting real time measurements of what's being fed by the sensors. Taking consolidation of the time we measure in could increase the accuracy of the model as for example measuring a temperature of 25 degree celsius at night would be interpreted differently than it would during the afternoon. Because it could be more likely to have a clear sky at 25 degree celsius at night and it would be less likely to have sunny days with that temperature during the afternoon. We think that having additional inputs to the model might have a chance of improving the accuracy of the model as it would be better at making accurate predictions.

Additionally, we would have wanted to incorporate some kind of network based model, that would take input data from multiple sensors spread out to cover a larger area to better

determine a good estimate of weather conditions. This would've required some kind of wide area network in which edge IoT devices could communicate their sensor readings. Some type of central machine learning model running on something like a Raspberry Pi could be used as a way to get a more representative prediction of local weather.

Group Roles:

- Wenzheng Zhao
    - Proposal
    - Pipeline Code Implementation
    - Model Deployment
    - Arduino code debugging
    - Debug
    - Readme
    - Restructuring report and information

- Manpreet Singh
    - Pipeline Code implementation
    - Arduino Hardware Testing, Tested each individual sensor on its own to ensure proper operation.
    - Arduino code development, Developed the Arduino code to interface with the sensors and run inference on the ML model.

- Mohammed Almousa
    - Proposal
    - Pipeline code implementation
    - Model structure and tests
    - Data preprocessing
    - Arduino code debugging and model deployment
    - Slides preparation
    - Report review

- Yuqi Nai
    - Data analysis
    - Arduino code debugging and model deployment
    - Test sensors on new boards
    - Field accuracy test

Code of the project implementation

Here is the link of our project repo:
https://github.com/Bobbed1999/EEP595-Weather-Wizard/tree/main

References

1. https://www.tensorflow.org/
2. https://scikit-learn.org/
3. https://www.tensorflow.org/lite
4. https://www.tinyml.org
5. https://keras.io/
6. https://github.com/BaptisteZloch/MakeIT-How-to-forecast-weather-with-Arduino-Nano-33
7. https://www.kaggle.com/datasets/vonline9/weather-istanbul-data-20092019
8. https://www.kaggle.com/datasets/emmanuelfwerr/london-weather-data
9. https://cds.climate.copernicus.eu/cdsapp#!/dataset/reanalysis-era5-pressure-levels?tab=overview
10. https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data