Predicting Pollution with Machine Learning

Matthew Jones

New Hall School

Table of Contents

[Abstract 4](#_Toc65163057)

[**Predicting Pollution with Machine Learning** 5](#_Toc65163058)

[Met Office Datapoint & London Air Quality. 5](#_Toc65163059)

[Django & Database Storage 5](#_Toc65163060)

[Locating Sources and Storing the Data 6](#_Toc65163061)

[Amazon Web Services 7](#_Toc65163062)

[Matching Pollution and Weather Locations 8](#_Toc65163063)

[Data Frames 9](#_Toc65163064)

[Training Algorithm 9](#_Toc65163065)

[Evaluation of model generation 11](#_Toc65163066)

[Website Creation 12](#_Toc65163067)

[Conclusion 14](#_Toc65163068)

[References 15](#_Toc65163069)

[Source Evaluation 18](#_Toc65163070)

[Ansible Documentation 19](#_Toc65163071)

[Bootstrap Documentation 19](#_Toc65163072)

[DataTables Documentation 19](#_Toc65163073)

[“Implementation of Haversine formula for school location tracking.” 20](#_Toc65163074)

[Department for Environment Food & Rural Affairs (DEFRA) 20](#_Toc65163075)

[Django Software Foundation 20](#_Toc65163076)

[CronHowTo 21](#_Toc65163077)

[jQuery 21](#_Toc65163078)

[Keras 21](#_Toc65163079)

[Appendices 25](#_Toc65163080)

[NEAT 26](#_Toc65163081)

[Testing NEAT 27](#_Toc65163082)

[Evaluation of NEAT. 28](#_Toc65163083)

[Original Model 28](#_Toc65163084)

[Overall Data Analysis 28](#_Toc65163085)

[NO2 Model 29](#_Toc65163086)

[O3 Model 31](#_Toc65163087)

[PM10 Model 33](#_Toc65163088)

[PM2.5 Model 35](#_Toc65163089)

[SO2 Model 37](#_Toc65163090)

Abstract

In this project, I have investigated the use of AI to predict pollution, based upon the weather forecast. This was based upon the hypothesis that the weather has a direct impact on the levels of pollution.

I built a machine learning application using Google’s TensorFlow and I trained it on a combination of Met Office and London Air data. I then evaluated the model using test data being a mixture of Met Office weather data and observed London Air data to find out how good the model was at predicting different kinds of pollution. The models turned out to be effective at predicting O3, PM10 and PM2.5 and poor at predicting NO2 and SO2. Finally, I built a web application with a public URL (pollutionepq.co.uk) to make predicted pollution data available for different locations in the South East of England.

Glossary:

AWS – Amazon Web Services

YAML – YAML Ain’t Markup Language

HTML – HyperText Markup Language

CSS – Cascading Style Sheet

NEAT – Neural Evolution of Augmented Topologies

NO2 – Nitrogen Dioxide

O3 – Ozone

PM10 – Particulate Matter 10 Micrometres

PM2.5 – Particulate Matter 2.5 Micrometres

SO2 – Sulphur Dioxide

Predicting Pollution with Machine Learning

The quantity of pollutants that are emitted by vehicles, aircraft and industry are clearly the most important predictor of pollution levels. It seems likely however, that the current weather conditions also have a strong effect. My hypothesis for this project is that the weather can be used to predict pollution levels, given that the amount of pollution emitted by vehicles and industry is likely to be relatively constant. In this project I will examine the relationship between weather and pollution data for several locations, to explore if there is a correlation. I will use machine learning to predict pollution levels based upon the weather forecast. Finally, I will set up a website to make this prediction available to users on the internet.

### Met Office Datapoint & London Air Quality.

Machine Learning requires real, representative data for both training and evaluation of the model. My first step was to set up a system to collect this data from respected sources.

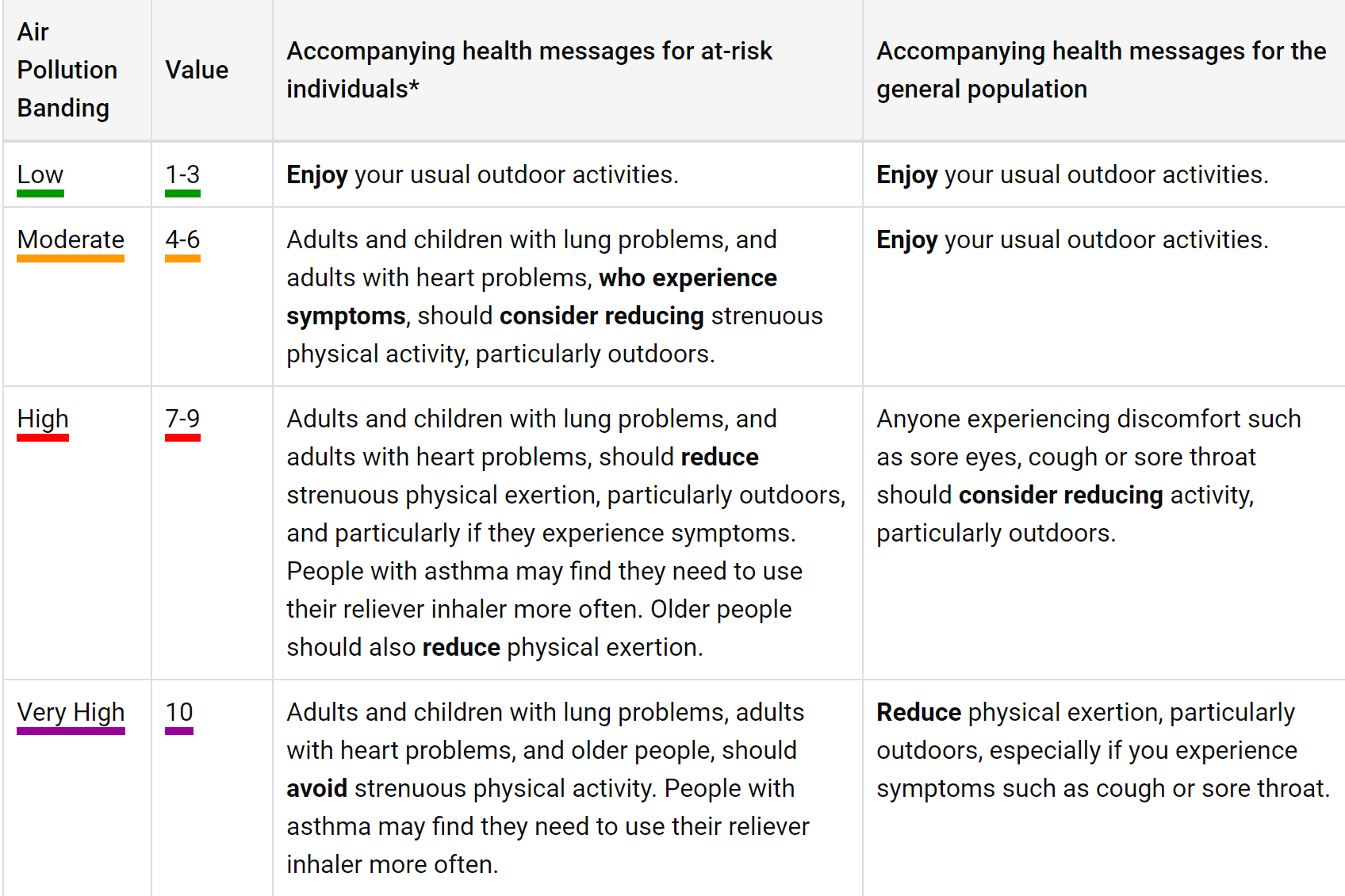
### Django & Database Storage

For this project I am using a Python web framework called Django[[1]](#footnote-2). Django makes it very easy to design the database and build a web application. To store the data, I have used a relational database server called MySQL. I have hosted the MySQL server on an AWS Compute Server so that it can run 24/7. Django was then configured to connect with the remote database[[2]](#footnote-3). Django models were used to define the structure of the database tables, as well as allowing the Python code to interact with the database. For this project I made 4 tables:

* ***WeatherLocation*** which stores the name, region, county, latitude, longitude, and elevation of each weather station.
* ***PollutionLocation*** which stores the assigned site code, site name, latitude, and longitude of each pollution logging station.
* Each ***WeatherLocation*** has several ***WeatherObservation*** records that store readings consisting of a timestamp, wind gust speed, temperature, wind speed, pressure, etc.
* Each ***PollutionLocation*** has several ***PollutionObservation*** that store readings consisting of a timestamp, species code, species description, the air quality index as well as the band that the index falls into.

### Locating Sources and Storing the Data

For weather data in the UK, I am using the Met Office DataPoint API[[3]](#footnote-4). I build a Django “command”[[4]](#footnote-5) to query the DataPoint API at regular intervals. This command sends two HTTP Requests[[5]](#footnote-6) using the python requests library[[6]](#footnote-7) asking for the weather data for the last 24 hours from all the weather stations within a boundary that I set around London. These requests return two JSON data structures, one containing the data about the weather stations and the other containing the weather readings for each weather station.

For pollution data I used a resource provided by Imperial College London[[7]](#footnote-8) that provides daily readings of pollution levels at different locations around London and the South East. It provides the data using the government standard index scales for pollution levels.[[8]](#footnote-9) I set up HTTP Requests to retrieve both the pollution readings and locations and store them in the relevant database tables.

### Amazon Web Services

By using an AWS EC2 Linux server, I was able to collect data 24 hours a day over an extended period.

To configure the server, I used an Ansible[[9]](#footnote-10) build configuration. Ansible is an automation engine used to configure a server. I built an Ansible YAML file that defines everything about the server and sets it up automatically. For the server, my configuration did the following:

* Update the compute unit.
* Install the database software and set up the database.
* Upload my project repository and install the commands to update the database.
* Configure a crontab to run the commands automatically.

A crontab[[10]](#footnote-11) is a feature of Linux that allows for the easy scheduling of command execution. My crontab runs the weather update script every 2 hours and the pollution update script every day.

### Matching Pollution and Weather Locations

For the machine learning to work, pollution and weather stations need to be paired based on proximity to each other to ensure that readings will be as closely matched as possible. I implemented an algorithm called the Haversine formula[[11]](#footnote-12) that compares two sets of co-ordinates to find the straight-line distance between them. This formula uses spherical trigonometry to find the shortest distance between two points around the surface of a sphere.

### Data Frames

After pairing pollution and weather stations, I loaded training data into the program. This used Data Frames, which are a powerful tool for analysing tabular data. I used the Python pandas[[12]](#footnote-13) library to load all my pollution and weather readings from the database into a single in-memory table. I used linear interpolation to fill gaps in the data, caused by weather station failures. The pollution readings are to be used for output data and the weather readings for input data. 80% of the data was used for training and 20% for testing.

### Training Algorithm

I evaluated two different machine learning algorithms; NEAT (See Appendices) and TensorFlow.

Machine Learning is a branch of AI that allows for the development of applications that learn from data and improve in accuracy over time.[[13]](#footnote-14)

TensorFlow is a machine learning platform by Google. It allows for the development and training of machine learning models. During this project I learnt about TensorFlow from community forums and the online resources and tutorials that Google provide.

TensorFlow required me to normalise[[14]](#footnote-15) the data. This changes all numeric values to the same scale without affecting variances in the range. I normalised the data separately for my first attempt. This resulted in problems with deploying the model successfully as it was not possible to normalise the data being used for the predictions with the same parameters. For my second attempt I used TensorFlow’s built-in normalisation pre-processor[[15]](#footnote-16) that can be added as a layer of the training model specifically for maintaining the same normalisation parameters in the deployed model.

I then developed a TensorFlow training script that set the model layers and Hyperparameters. Hyperparameters are the variables controlling the training process of a Machine Learning model. To set these decide on a sensible number of layers. Many problems can be solved with one single Dense layer[[16]](#footnote-17), which is a generic Neural Network layer made up of many nodes. Hyperparameters can then be selected by hyperparameter tuning. This is the process of running multiple potential Hyperparameters through the machine learning model and using the best ones. TensorFlow has a library called Keras Tuner built specifically for this purpose.[[17]](#footnote-18)

Keras includes a feature called callbacks.[[18]](#footnote-19) Callbacks can be configured to perform different functions throughout the training process. For my training algorithm I made use of the “ModelCheckpoint” callback.[[19]](#footnote-20) This monitors a pre-specified statistic of the model training and whenever it achieves a better value in the validation data than the previous best, it saves that as the chosen model for production use. I have the ModelCheckpoint function monitoring the absolute mean error value of the model’s output. This means that whenever the differences between the predictions and the real values are the lowest it saves the model.

After configuring the model and the callbacks, the model needs to be compiled. This is the last step before model training. An optimizer is defined to improve the speed and performance when training a model. The most relevant ones to this project are “RMSProp”, “SGD” and “Adam”. While setting up the compiler I defined the loss function which is used to determine the difference between the training data and the predicted values.

My final attempt at creating a TensorFlow model was a success. Using the built-in normalisation pre-processor, I was able to configure it as a permanent part of the model. This was an issue with my first model (See Appendices). For layers I used one Dense layer of 64 units and an activation function of ‘relu’ along with a second Dense layer of 1 unit to give the desired output density of 1 value. When compiling the model, I used an optimizer of ‘Adam’ and a loss function of mean absolute error to try and bring that value as low as possible.

### Evaluation of model generation

From my analysis (See appendices) I believe that the best model generated was the Ozone model. This was the most accurate over a large range. Next were the two Particulate Matter models. The NO2 and SO2 models were poor, with low accuracy and predictive power.

### Website Creation

#### HTML & CSS

I created a web application to display my Artifact. To do this I learned how to write HTML and CSS using documentation provided by W3Schools. I learned how to make drop down selection boxes[[20]](#footnote-21) and checkboxes[[21]](#footnote-22) to allow the user to select the data that they required.

The CSS I used to design the look of the website was based upon a framework called Bootstrap.[[22]](#footnote-23) This is a responsive page framework useful for building websites that work on both mobile devices and computers. I made use of bootstrap to split aspects of my page up in such a way that it would be mobile friendly.

#### JavaScript

To make the dropdown boxes and checkboxes functional I wrote JavaScript using a plugin called jQuery.[[23]](#footnote-24) My jQuery code was used to retrieve the current values from the dropdown boxes and checkboxes. The function then sends the data via an HTTP Request in JSON format to the Django application that returns the predicted values from the model for the selected location.

To display the data returned from the web application I wrote another function using a JavaScript plugin called DataTables[[24]](#footnote-25). DataTables allow the data to be presented in an attractive, sortable, and even searchable table.

#### Django Views

A view[[25]](#footnote-26) is a component of Django that can responds to HTTP requests. I built a view to return the pollution readings for a selected location. The view makes an HTTP Request to the Met Office Datapoint service. This requests the forecast for the next 5 days. The weather forecast is then run through the relevant AI models to generate pollution predictions for each requested pollution type. This data is then returned to the web page that requested it in JSON format for display.

#### Apache 2

I ran my Django application using the commonly used Apache[[26]](#footnote-27) web server and the mod\_wsgi Apache plugin. To configure this, I first had to load my website code onto a server I have connected to the Internet. I set up static libraries for the non-dynamic parts of the website such as image files. The Apache configuration file is very intuitive to understand after reading the documentation. To set up a website a virtual host must be configured, and the URL and location of the website passed in. I obtained my domain name (pollutionepq.co.uk) from Google Domains.

# Conclusion

This project has taught me a lot about programming. I have learnt how to make responsive web applications using Django, HTML and JavaScript. I have also learnt about machine learning and artificial intelligence; this has potentially been my favourite part of this project. My delve into the world of artificial intelligence will more than likely lead to me attempting several more projects of this nature with different data and data types. Web development and Machine Learning are both massive parts of the evolving Internet and so are very important skills to have in the modern age. In my opinion this project went very well given how I started, with no knowledge of machine learning and my only web development experience being website builders online. I have achieved my goals for this project.

References

Ansible. n.d. *Ansible Documentation.* Accessed February 2020. https://docs.ansible.com/ansible/latest/user\_guide/intro\_getting\_started.html.

Bootstrap. n.d. *Bootstrap Docs.* Accessed December 2020. https://getbootstrap.com/docs/5.0/getting-started/introduction/.

CodeReclaimers. 2019. *NEAT-Python.* 23 November. https://neat-python.readthedocs.io/en/latest/neat\_overview.html.

DataTables. n.d. *DataTables Manual.* Accessed December 2020. https://datatables.net/manual/.

Dauni, P. 2019. “Implementation of Haversine formula for school location tracking.” *Journal of Physics: Conference Series* 3. https://iopscience.iop.org/article/10.1088/1742-6596/1402/7/077028/pdf.

Department for Environment Food & Rural Affairs. n.d. *Daily Air Quality Index.* Accessed December 2020. https://uk-air.defra.gov.uk/air-pollution/daqi.

Django Software Foundation. n.d. *Databases.* https://docs.djangoproject.com/en/3.1/ref/databases/.

—. n.d. *Django.* https://www.djangoproject.com/.

—. n.d. *How to use Django with Apache and mod\_wsgi.* Accessed December 2020. https://docs.djangoproject.com/en/3.1/howto/deployment/wsgi/modwsgi/.

—. n.d. *Models.* https://docs.djangoproject.com/en/3.1/topics/db/models/.

—. n.d. *Writing custom django-admin commands.* https://docs.djangoproject.com/en/3.1/howto/custom-management-commands/.

—. n.d. *Writing Views (Django).* Accessed December 2020. https://docs.djangoproject.com/en/3.1/topics/http/views/.

gweatherby. 2016. *CronHowto.* 20 November. Accessed February 2020. https://help.ubuntu.com/community/CronHowto.

IBM Cloud Education. 2020. *Machine Learning.* 15 July. Accessed 02 25, 2021. https://www.ibm.com/cloud/learn/machine-learning.

jQuery. n.d. *Ajax (jQuery).* Accessed December 2020. https://api.jquery.com/category/ajax/.

Keras. n.d. *Callbacks API.* Accessed August 2020. https://keras.io/api/callbacks/.

n.d. *London Air.* Accessed February 2020. http://www.londonair.org.uk/london/asp/datadownload.asp.

n.d. *Met Office DataPoint.* Accessed February 2020. https://www.metoffice.gov.uk/services/data/datapoint.

Microsoft. 2019. *Microsoft Docs (Normalize Data).* 5 June. Accessed August 2020. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data.

pandas development team. n.d. *pandas.* Accessed August 2020. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html.

Reitz, Kenneth. 2020. *requests 2.25.1.* 16 December. Accessed January 25, 2021. https://pypi.org/project/requests/.

TensorFlow. n.d. *Introduction to the Keras Tuner.* Accessed December 17, 2020. https://www.tensorflow.org/tutorials/keras/keras\_tuner.

—. n.d. *tf.keras.callbacks.ModelCheckpoint.* Accessed September 2020. https://www.tensorflow.org/api\_docs/python/tf/keras/callbacks/ModelCheckpoint.

—. n.d. *tf.keras.layers.Dense.* Accessed August 2020. https://www.tensorflow.org/api\_docs/python/tf/keras/layers/Dense.

—. n.d. *tf.keras.layers.experimental.preprocessing.Normalization.* Accessed September 2020. https://www.tensorflow.org/api\_docs/python/tf/keras/layers/experimental/preprocessing/Normalization.

w3schools. n.d. *HTML <input type = "checkbox">.* Accessed December 2020. https://www.w3schools.com/tags/att\_input\_type\_checkbox.asp.

—. n.d. *HTML <select> Tag.* Accessed December 2020. https://www.w3schools.com/tags/tag\_select.asp.

—. n.d. *What is HTTP?* https://www.w3schools.com/whatis/whatis\_http.asp.

# Source Evaluation

|  |  |
| --- | --- |
| Ansible Documentation | From Ansible I followed documentation on how to set up a build configuration file, this was used for configuring the server that performs data collection.  This documentation is made by the company that developed the software. Therefore, they are the authority figure on this topic.  This documentation is there to teach others how to educate software developers and system administrators on how to use Ansible. |
| Bootstrap Documentation | From Bootstrap I followed documentation on how to make a website that is responsive and therefore both computer and mobile friendly.  This documentation is made by the company that developed Bootstrap. Therefore, they are the authority figure on this topic.  This documentation was created to educate web developers on how to use Bootstrap |
| DataTables Documentation | This documentation was used to create the table to display the model output on the website.  This documentation is made by the company that developed DataTables. Therefore, they are the authority figure on this topic.  This documentation was created to educate developers how to use DataTables |
| “Implementation of Haversine formula for school location tracking.” | This paper was used to find out how to calculate the distance between two sets of GPS coordinates.  The paper was published by the IOP as an example usage for this formula. They are not the original discoverers of the formula.  This paper was written to show how location tracking can be used to find the closest school to a student. It gives detailed analysis of how this works and so is a good source of information on how to use the Haversine formula |
| Department for Environment Food & Rural Affairs (DEFRA) | From DEFRA I found information on the government standard pollution reading scale. This allowed me to understand the index scale of the London Air data and use that data further on my website to display more detail for my predictions.  This documentation is made by the government who also decided on this scale. Therefore, they are the authority figure on this topic.  This documentation is there as government information on the dangers of pollution levels and how to respond to different pollution index levels. |
| Django Software Foundation | From the Django Software Foundation, I learnt how to setup a Django application, develop database models, write Django commands such as my commands for collecting weather data and training the AI and write views to both display my web page and return the data from the generated models.  This documentation is made by the Django Software Foundation who are the developers who made the Django Framework. Therefore, they are the authority figure on this topic.  This documentation exists to educate developers on how to use the Django Framework. |
| CronHowTo | This documentation was used to help me set up scheduled commands on the data collection server.  This documentation was written by the Ubuntu community, and so although in the end it was useful. It is not necessarily the best source for this information.  The purpose of this documentation is to educate on how to use the Crontab functionality in Ubuntu/Linux |
| jQuery | The jQuery documentation was used to set up the ajax HTTP requests from the website. This was how the website communicated with the other Django views to get the output from the AI model.  This documentation was written by jQuery who are the developers who wrote jQuery. Therefore, they are the authority figure on the topic.  The purpose of this documentation is to educate JavaScript developers on how to write jQuery code. |
| Keras | Keras is a part of TensorFlow. This documentation was used to learn how to use the callback feature of Keras. This allowed for the easy saving and manipulation of the model throughout training.  This documentation was written by Keras who the developers are who wrote this module. This makes them the authority figure on this topic.  The purpose of this documentation is to educate developers on how to use Keras. |
| London Air | This website was used to retrieve the pollution values used for training.  This website is owned and operated by Imperial College London. It makes use of the Government pollution index. It takes scientific measurements of pollution levels and therefore can be deemed accurate and a good source of information.  The purpose of this website is to report pollution levels across the London area. |
| Met Office Datapoint | This website was used to retrieve both the historical weather values for training as well as the forecast values for the deployed model.  This website is owned by the Meteorological Office which is a government owned weather recording and forecasting service. They are therefore trustable as a source of accurate weather information. |
| Microsoft Docs | From this documentation I learnt about the normalization of data in programming and data science.  This website is owned by Microsoft and is used for their visual studio documentation. This therefore increases how trustable this source is. |
| Pandas Development Team | This website taught me how to create in-memory tables in python using the pandas API.  The Pandas Development Team are the creators of the pandas API. Therefore, they are the authority on this topic.  This website is to educate developers on how to use the pandas API. |
| Requests | This documentation taught me how to use the requests library. This is used through the project to send HTTP requests between views and retrieve data in the downloads from the Met Office and London Air.  This documentation has been made by some of the developers of the Requests library and therefore they are the authority on how it works making this a very trustworthy source. |
| TensorFlow | TensorFlow was the main part of this project and their resources where invaluable when creating my artifact. I used this documentation to learn how to create models, configure model layers, normalize data as part of a model, tune and learn about hyperparameters and set up checkpoints in conjunction with Keras callbacks.  This documentation is written by Google who are the owners and developers of TensorFlow. Therefore, they are the authority figure on this topic. |
| w3schools | W3schools is a set of documentation for a variety of programming aspects. It teaches web development online for free. I used it to learn about drop down boxes and checkboxes in HTML as well as the differences between HTTP and HTTPS  W3schools are dedicated to teaching web development online. They are very highly rated by the online web development community and this makes them a trustworthy source of information. |
| IBM Cloud Education | This page was used to help me define what machine learning actually is.  IBM are one of the largest technology companies in the world, they have been a leader in the development of software development and computers as a whole. Their input on this is very trustable. |
| CodeReclaimers | Code Reclaimers wrote the Python implementation of the NEAT algorithm as well as the documentation that I used to develop my NEAT game.  As Code Reclaimers wrote the Python API for NEAT, he is therefore the authority on the topic of using that API. This also makes him a trustable source on the topic of NEAT as a whole even if he is not the original creator |

# Appendices

|  |
| --- |
| NEAT NEAT[[27]](#footnote-28) is an evolutionary algorithm that uses machine learning to create artificial neural networks. It works by defining a fitness value for a generation that is affected by user defined parameters, by doing this the effectiveness of the algorithm will try and better itself towards the events that gain it an addition to the fitness function. As it trains it progresses through a user-set number of generations which evolve through reproduction or mutation. The simulation terminates when the set number of generations is reached, or an individual reaches the maximum fitness threshold. |
| Testing NEAT To test this algorithm, I implemented it into a game I had previously made, this game was based on trying to land a spaceship on a planet at a safe speed with a set amount of fuel. To do this I had to remove the user controls and replace them with the output from the algorithm. For my configuration I used the activation function Cube, this returns a value between -2 and 2. The configuration was also set to return 2 outputs, these were used respectively to control the state of the thrust and the directional controls with thrust activation being any value above 0.5 and directional controls being either side of 0.5 up to 1 and down to 0.  By running this program, I was surprised to find that the number of landings per generation increased almost exponentially as shown by this graph:  The UI for the program was the game along with different stats for the current simulation as shown below: |
| Evaluation of NEAT. Although NEAT is relatively easy to write code for, it has the problem of needing to define what is good, it also lacks the ability to save a model once it has been generated, this makes me unable to deploy a saved model to production, rendering my product impossible to make by these means and therefore I was forced to select a different AI solution. |
| Original Model Initially I attempted to predict specific pollution levels for a location based on the pollution and weather values of all the other locations. Although this made successful predictions, it was not useful for the eventual goal of this project which was to create a web resource that could be used to view the predictions of the model. This is because the model required the weather forecast for all possible locations in the database as well as the pollution forecast for all the pollution locations being used in the original model. |
| Overall Data Analysis From this graph we can see the relationships between different aspects of the weather data that is being used as the input to the model. This graph shows that there is a negative correlation between temperature and screen relative humidity however, that is the only correlation that is clearly visible.  This graph also shows the distributions of the data values. We can see that the values for wind gust, temperature, wind speed, screen relative humidity and visibility are all approximately normal distributions. The reason that weather type does not appear to be normally distributed is because instead of being a recorded value it is continuous data. |
| NO2 Model This graph shows the training loss value and the validation loss value across the entire training process for the NO2 model. We can see that it improves the most in the first 20-25 epochs and then the rest of the training is used to iron out any remaining error. An epoch is the time for one pass of the data training set.  This graph shows the real and predicted values for this trained model after having reloaded the best version of the model. Here we can see that it works well for the cases where the real value would be 1. However, it predicts incorrectly for anything above this. While attempting to train this specific model I have not been able to eliminate this problem and so I believe the model has either not found any correlation between the weather data and NO2 values or there was not enough variance in the training data to make an effective model. The closer values are to the y=x line the more accurate the models are.  The next graph shows the error of all predictions, this shows the model to be better than it is, as in reality most of the values were an index of 1. |
| O3 Model This graph shows the loss values of the O3 model over the training session, as with NO2 it improves the most at the very beginning and then uses the rest to try and iron out any issues. However, this model is starting to show signs of overfitting towards the end where the validation loss starts drifting above the training loss. This is of course counteracted by the checkpoint callback that saves the best run as the actual model.  This graph shows that the model is working to a reasonable degree, quite a lot of which will be remedied when rounding the output of the model to get an integer index value. The gradient of this model is not quite on the y=x line but is significantly closer than the NO2 model.  This graph shows the error for the predictions. Since the largest proportion of the values fall within a -0.5 – 0.5 error range we can be confident that when the values are rounded the model will be more accurate than what has been shown in this testing, even if it has already proved itself as a good model. |
| PM10 Model This graph is very similar to the loss graph for the O3 model, however less overfitting is evident in this one. This shows that the model required slightly longer for the best possible training than the O3 model.  This graph shows the model’s predictions against the true values for the PM10 model, it shows that for indexes 1 and 2 it is making reasonable predictions. At index 3 it starts to drift below the y=x line and then at index 4 the model begins to fail. This could be caused by a lack of data above indexes 1 and 2 which would fit with my observations of the database.  Here we can see that the error for this model is very good, however as we saw with the previous graph that does not hold up for the higher index values. |
| PM2.5 Model This graph shows almost no overfitting apart from when looking very closely. This is a sign that the model is accurate. It also required a lot of this time to become this accurate.  This graph shows that the PM2.5 model is very similar to the PM10 model in terms of the ability to predict the higher indexes.  This graph shows the similarities again with the PM10 model. When at lower indexes it is very accurate, however at higher indexes the accuracy diminishes. This is probably for the same reasons, being that higher indexes occur rarely in my collected data. |
| SO2 Model I did not have particularly high hopes for this model, this is because when analysing my data, I noticed that all but 7 of the recordings across the last 11 months where an index of 1. This led to the following model:  As shown by these three graphs I can say with certainty that this model was not able to find any correlation between the weather data and the pollution values at all, this is shown by the consistency of predicting an index of 0 which is not a value present on the scale. |

1. (Django Software Foundation n.d.) [↑](#footnote-ref-2)
2. (Django Software Foundation n.d.) [↑](#footnote-ref-3)
3. (Met Office DataPoint n.d.) [↑](#footnote-ref-4)
4. (Django Software Foundation n.d.) [↑](#footnote-ref-5)
5. (w3schools n.d.) [↑](#footnote-ref-6)
6. (Reitz 2020) [↑](#footnote-ref-7)
7. (London Air n.d.) [↑](#footnote-ref-8)
8. (Department for Environment Food & Rural Affairs n.d.) [↑](#footnote-ref-9)
9. (Ansible n.d.) [↑](#footnote-ref-10)
10. (gweatherby 2016) [↑](#footnote-ref-11)
11. (Dauni 2019) [↑](#footnote-ref-12)
12. (pandas development team n.d.) [↑](#footnote-ref-13)
13. (IBM Cloud Education 2020) [↑](#footnote-ref-14)
14. (Microsoft 2019) [↑](#footnote-ref-15)
15. (TensorFlow n.d.) [↑](#footnote-ref-16)
16. (TensorFlow n.d.) [↑](#footnote-ref-17)
17. (TensorFlow n.d.) [↑](#footnote-ref-18)
18. (Keras n.d.) [↑](#footnote-ref-19)
19. (TensorFlow n.d.) [↑](#footnote-ref-20)
20. (w3schools n.d.) [↑](#footnote-ref-21)
21. (w3schools n.d.) [↑](#footnote-ref-22)
22. (Bootstrap n.d.) [↑](#footnote-ref-23)
23. (jQuery n.d.) [↑](#footnote-ref-24)
24. (DataTables n.d.) [↑](#footnote-ref-25)
25. (Django Software Foundation n.d.) [↑](#footnote-ref-26)
26. (Django Software Foundation n.d.) [↑](#footnote-ref-27)
27. (CodeReclaimers 2019) [↑](#footnote-ref-28)