Albus

Dominic McLoughlin

September 2021

1 Set up

I include a `requirements.txt` file in the package, which may be used to install the required dependencies. If using Anaconda, which I recommend, the following commands will set up a new environment:

```
conda create -n albus python=3.9
conda activate albus
pip install -r requirements.txt
python main.py
```

2 Data preprocessing

First, I copied the data columns to the top of the CSV. I then used `pandas.read_csv` to read in the data. I replaced all instances of `?` with `np.nan` and then converted the relevant columns to float data type. The machine learning package `scikit-learn` does not allow missing values, but this is not a problem – to avoid look-ahead bias in the time-series data, I will be creating new features that are based off lagged data anyway, so will treat missing values then.

3 Exploratory data visualisation

To check the data is behaving as expected, I plotted some of the columns against time. Figure 1 shows that the temperature fluctuates seasonally, which is good! I also show the first 5 rows and a summary of the columns. This becomes available by setting `constants.EXPLORE_MODE` to True.

4 Defining a benchmark

The benchmark for this data is the log loss of the underlying population probability for an adverse event. The constant probability estimate is 0.06314127861089187, simply the total number of adverse events divided by the number of days in the dataset. I calculate a benchmark log loss between this constant value and the actual events, giving a score of 0.2355247752891056. A successful model will

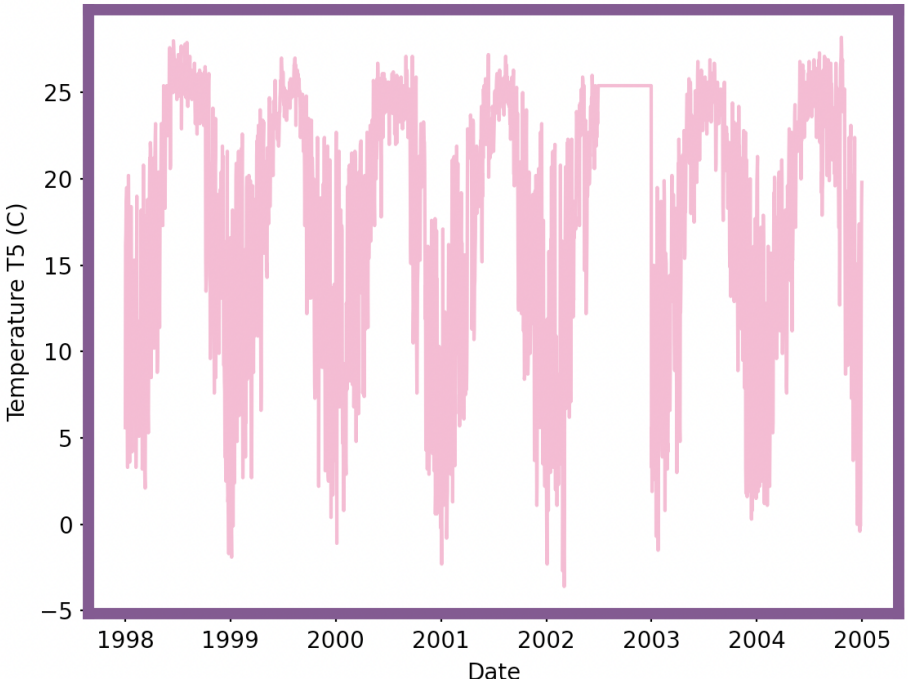

Figure 1: Temperature as a function of time.

have predicted probabilities which match the test data better, and will have a **lower** log loss than this benchmark.

5 Create new features

Here I create new features through three different methods.

Date features I make new columns for the month and day to account for periodic changes in the data

Lag features I add a column for each feature for each lag of increasing size from 1 to X, where X is the number of days as described in the problem definition. To change X, go to `constants.lag`. For example, T5 would have a column for lag-by-one, a column for lag-by-two ... a column for lag-by-X.

Window features I add another column for each feature, with a rolling mean of the last X days.

To avoid look-ahead bias (**crucial**!) I shift each of these columns by the appropriate amount. I then discard the first X rows, as these are now not fully defined (you can't have a lag-by-two column on the first day!).

6 Find best features

I perform a K-fold cross-validation on the data. However, because it is a time series, I use a time series split function to avoid disrupting the order, and to avoid look-ahead again. I choose a LogistionRegression model, with StandardScaler() scaling of inputs. Since this is a linear model, the coefficients represent the dependence of prediction on each feature, so I sort by these and take the top 15 features forward. I determined that 15 was the best number of features because this minimised the log loss.

7 Rebuild the model

I rebuild the model using only the best features (which are printed in the console). The best features are:

```
[T23_lag1 T_PK_lag1 WSR20_lag1 V85_lag1 WSR23_lag1
T19_lag1 WSR9_lag5 U70_lag2 T18_lag3 WSR22_lag1
 T19_lag3 T85_lag4 HT50_lag4 T70_lag3 WSR0_lag7 ]
```

Most of these are lagged by a small number of days, and only one of these has a lag as long as a week. This confirms that more recent data is in general a better predictor of ozone events.

8 Score the model performance

```
Benchmark log loss: 0.2355247752891056
   Final split predictive log loss: 0.12108586975976071
   Improvement over baseline log loss: 0.11443890552934488
```

This is good news! The model has beaten the benchmark by a considerable margin. Log loss is the preferred metric here for a simple reason. Consider using accuracy as a metric, what would happen? The mean probability of an event is 6%, meaning you could achieve an accuracy of 94% by simply guessing "No event" every day. Instead, log loss evaluates performance based on the calculated probability distribution. Having a lower loss means we are accurately predicting what will happen more accurately than just guessing no.

However, while the probability each day is useful, this has not actually predicted labels. To do this, you simply choose a threshold level and label a predicted event whenever the probability for that day goes above the threshold. This choice is up to the user of the model, so I don't make the choice here.

9 Plot the model predictions

Figure 2 shows the predicted probability as a function of time, with actual events flagged with vertical markers. Note that this algorithm has predicted high probability with no event around Christmas 2004 - perhaps this was a near miss! The figure only shows dates from the test split date range, because the model would predict anything before this with a high bias since that was the training data.

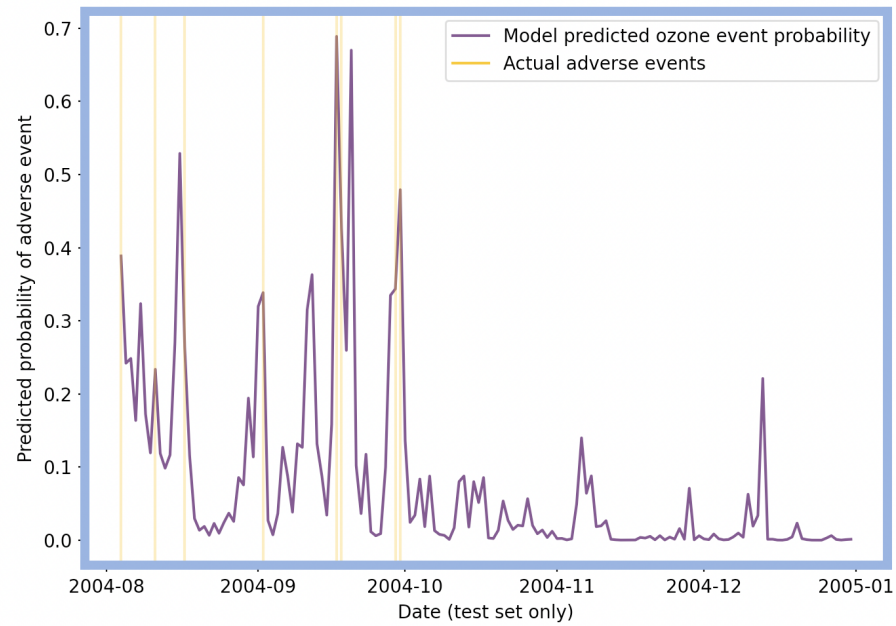

Figure 2: Predicted probability against time. Actual events are flagged with vertical markers.