WEATHER PREDICTION REPORT

CI513

In this report we will look at using artificial intelligence to build a weather prediction system

16/08/2024

Contents

Introduction

Weather is commonly known for its sometimes-unpredictable behaviour. This is why weather prediction is such a fascinating subject, the weather has been studied for a long time, with weather reports and forecasts being easily accessible in the current day with a click of a button, however, even these reports can get the weather wrong. After all, it is just a prediction.

In this report we will be looking at the problem of weather prediction, using linear regression and machine learning we will aim to take a dataset and predict future weather from the data provided to us.

This report aims to implement an AI pipeline and develop a working weather prediction system. The pipeline will follow a process in which this document will also follow.

1. Research
    a. Data Selection
    b. Model Selection
2. Data Cleaning
3. Model Implementation
4. Model Testing
5. Future Predictions

The goal of this project is to be able to predict the weather up to 30 days in the future of the end of the dataset used.

Research

A successful weather prediction model depends heavily on the quality of the data and an appropriately chosen algorithm. In this section, the details of the selection process for the dataset and model will be outlined.

Data Selection

The dataset selected for this project is weather data from Kaggle called “historical hourly weather data” (BENIAGUEV, 2018), chosen for its wide range of meteorological variables, including temperature, humidity, pressure, wind direction, and wind speed. This dataset contains hourly recordings from October 2012 to November 2017, such an abundant dataset is imperative when developing an accurate weather prediction model.

Data Structure

Initially, the dataset was spread across several CSV files, each corresponding to the 36 cities, from which the data was gathered. Each file contains 45,254 hourly entries per city, which accounts for almost 10 million data points: 45,254 × 36 cities × 6 variables. The only exception is the *city_attributes.csv* file, which only holds the names of cities as well as their longitude and latitude coordinates, which are unnecessary for what we are trying to accomplish.

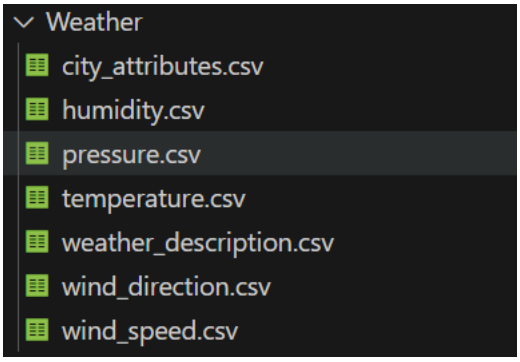

Figure 1 - .csv files of all data

Data Quality

Upon initial inspection of the datasets, it was noted that certain datasets had missing values, particularly in the initial entries. These gaps, if not dealt with, could potentially skew the models' predictions. As part of the data cleaning step, these missing values will be addressed using appropriate cleaning techniques.

Model Selection

Ridge Regression Model

“Ridge Regression is an adaptation of the popular and widely used linear regression algorithm. It enhances regular linear regression by slightly changing its cost function, which results in less overfit models.” (Giba, n.d.)

Ridge Regression is well-suited for situations where multicollinearity (a high correlation between independent variables) is common. This is often found within weather prediction where variables such as humidity, pressure, and temperature are interrelated.

Ridge Cross-Validation

To further improve the performance of the *Ridge Regression Model,* the use of *RidgeCV (Cross-Validation)* is implemented alongside *Repeated k-fold cross-validation*. This method involves repeating the cross-validation procedure multiple times and reporting the mean result across all folds and runs. As explained here:

"This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error." (Brownlee, 2020)

Algorithm Explanation

Ridge Regression

The *Linear Regression Model* is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to the observed data, for our purposes we will need to use the *Multiple Linear Regression* Equation:

$$y = \beta_0 + \beta_1 X1 + \beta_2 X2 + \cdots \beta_n Xn$$

Where, y is the dependent variable, X1/X2/Xn are the independent variables β_0 is the intercept, and $\beta_1/\beta_2/\beta_n$ are the slopes. The goal of the algorithm is to find the line of best fit that can predict values based on independent variables. (Linear Regression in Machine Learning, 2024)

However, *Ridge Regression* modifies the standard *Linear Regression Model* by incorporating a penalty into the loss function, defined by:

$$Cost\ Function = \sum (y_i - \hat{y}_i)^2 + \lambda \sum \beta_j^2$$

Where y_i represents the actual value, $\hat{y}_i$ is the predicted value, β_j is the coefficients of the model and λ is the parameter that controls the penalty’s strength. A higher λ value increases the penalty for large coefficients, shrinking them and reducing the complexity of the model. This

helps to maintain stability within the model and improve the accuracy of predictions. (mohi, 2020)

RidgeCV

RidgeCV is a method that combines *Ridge Regression* with *Cross-Validation* to automatically select the optimal regularization parameter λ. To find this value we follow some simple steps, by defining a range of values for λ.

Then split the dataset into small sections usually 5 or 10, these are called k folds, with the k representing the value. The *Ridge Regression Model* is then trained using k-1 folds of the data, validating the results on the remaining fold.

This is then repeated for each value of λ across all folds. Averaging out any variation due to data splits, offering a more robust estimate of model performance. Once complete, the average performance metric of each λ is calculated across all folds. The λ with the minimal validation error is chosen as the optimal value for the regularization parameter.

Finally, the *Ridge Regression* model is refit using the optimal λ value, the model can then be used to make predictions on new data. Evaluation of the model can then be done to confirm that the λ value chosen generalises well with the data.

Data Cleaning

Data cleaning is an important step in working with datasets, Data cleaning is used to remove null and not applicable values, as well as remove any data duplicates that could skew the data when working with it and give misleading conclusions.

Given the large volume of data in this project, the priority was to organize and clean the data to make it manageable and suitable for modelling. To achieve this, we utilized Python's *pandas* and *csv* libraries, imported as *pd* and *csv*, respectively.

```
import pandas as pd
import csv
✓ 1.3s

Humidb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\humidity.csv")
Presdb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\pressure.csv")
weathDescdb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\weather_description.csv")
wDiredb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\wind_direction.csv")
wSpeedb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\wind_speed.csv")
Tempdb = pd.read_csv("C:\Users\georg\Documents\GitHub\Y2S2AI\Datasets\Weather\temperature.csv")
```

Figure 2 - Getting all the .csv files into one place

Our dataset was originally dispersed across multiple CSV files, each containing data for 36 cities. To focus on a single location for our predictive model, we selected Portland due to its comprehensive data coverage. The relevant columns for Portland were extracted from each CSV file and merged into a new Data Frame, as shown in Figure 4. This consolidation allowed us to work with a unified dataset that included all necessary variables for the city of Portland.

Data Transformation

To make the dataset more suitable for our specific analysis, we performed several transformations. The original dataset recorded temperature in Kelvin, which was not ideal for our purposes. We converted the temperature from Kelvin to Celsius using the formula:

$$C° = K° - 273.15$$

This conversion was performed to ensure consistency with other datasets and ease of interpretation.

```
HumiP = Humidb["Portland"]
PressP = Presdb["Portland"]
weathP = weathDescdb["Portland"]
windDP = wDiredb["Portland"]
windSP = wSpeedb["Portland"]
TempDt = Tempdb["datetime"]
TempC = Tempdb["Portland"] - 273.15
header = ["DateTime", "Temperature", "Humidity", "Pressure", "WeatherDesc", "Wind Direction", "Wind Speed"]
data = pd.DataFrame(columns= header)
data["DateTime"] = TempDt
data["Temperature"] = TempC
data["Humidity"] = HumiP
data["Pressure)"] = PressP
data["WeatherDesc"] = weathP
data["Wind Direction"] = windDP
data["Wind Speed"] = windSP
data.head(4)
```

✓ 0.0s

	DateTime	Temperature	Humidity	Pressure	WeatherDesc	Wind Direction	Wind Speed	Pressure)
0	2012-10-01 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2012-10-01 13:00:00	8.930000	81.0	NaN	scattered clouds	0.0	0.0	1024.0
2	2012-10-01 14:00:00	8.933252	80.0	NaN	scattered clouds	4.0	0.0	1024.0
3	2012-10-01 15:00:00	8.941866	80.0	NaN	scattered clouds	18.0	0.0	1024.0

Figure 3 - merging all the .csv files into a single DataFrame for one city

We created new columns for *Year*, *Month*, *Day*, and *Hour* by transforming the *DateTime* field into a DateTime type. These new columns were essential for time-based analysis, such as aggregating hourly data into daily averages, which could be more meaningful for certain predictive models.

```
print(data.shape)
data['DateTime'] = pd.to_datetime(data['DateTime'])
data["Year"] = data['DateTime'].dt.year
data["Month"]= data['DateTime'].dt.month
data["Day"] = data['DateTime'].dt.day
data["Hour"]= data['DateTime'].dt.hour
data.head(4)
```

✓ 0.0s

```
(45253, 8)
```

	DateTime	Temperature	Humidity	Pressure	WeatherDesc	Wind Direction	Wind Speed	Pressure)	Year	Month	Day	Hour
0	2012-10-01 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2012	10	1	12
1	2012-10-01 13:00:00	8.930000	81.0	NaN	scattered clouds	0.0	0.0	1024.0	2012	10	1	13
2	2012-10-01 14:00:00	8.933252	80.0	NaN	scattered clouds	4.0	0.0	1024.0	2012	10	1	14
3	2012-10-01 15:00:00	8.941866	80.0	NaN	scattered clouds	18.0	0.0	1024.0	2012	10	1	15

Figure 4 - adding additional rows and cleaning the data

Data Cleaning Techniques

To ensure the integrity of our dataset, we employed the following cleaning techniques, handling missing data, removing duplicates, and retaining valid zeros.

From our investigation of the code, we knew that there were data entries without any data, rather than inserting values into these gaps, which could skew the data, we opted to remove these entries using the *dropna* function. To avoid affecting our model's performance by adding unverifiable data.

Although unlikely that the data had duplicate entries, as a precautionary measure, we performed a *drop_duplicates* function. This step was to ensure that no time stamps had multiple entries, which could have negatively impacted the model.

It was imperative to understand that in meteorological data variables such as windspeed or temperature could genuinely be zero. Removing these zeros could lead to the data being wrongly misinterpreted; for this reason, no zeros were removed from the data.

Saving the Data

After cleaning and transforming the data, the final step was to save the cleaned dataset to a new CSV file. This file was stored in a dedicated location to avoid any confusion or errors in future steps of the project, ensuring that the cleaned data could be easily accessed for modelling.

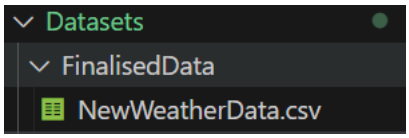

Figure 5 - the new .csv file saved for later use

Model Implementation

Data Preparation

Before implementation, the first step is to prepare the data. This involves loading the dataset and formatting it into a suitable state for model training and evaluation. The dataset is split into two parts, the training set which will contain the bulk of the data, while the test set is a smaller subset that represents new unseen data.

Feature Selection

Once the data is in a manageable state, the next step is to take the independent variables that will be used for the prediction, features such as humidity, pressure, windspeed, and wind direction are selected because they are most likely to have a significant on the temperature, the target.

Feature engineering also plays an important role here, features such as Hour, Day, Month, and Year were omitted from the data and DateTime was indexed to keep a temporal order to the data. While we had already converted the temperature from Kelvin to Celsius for ease of interpretation, this is something we could have done during this step as well.

Model Training

To ensure the model doesn't overfit the training data as well as generalise to the new data, we implement RidgeCV (sckit-learn, n.d.) and Repeated K-Fold Cross-Validation. To train the model on different subsets of the data multiple times. To ensure the best results, we ran the K-fold Cross-Validation in splits of 10, each of these repeating 5 times, and set the random state to 42 to ensure that we could repeat the outcome.

We then use Ridge Regression to train the model on our Test-Train Split fitting the data to the model and getting the results back including the Mean Absolute Error (MAE) and Mean Square Error (MSE) which we can use as metrics to see how accurate the model is.

```
X = weather[['humidity', 'pressure', 'wind direction', 'wind speed', 'weatherdesc']]
y = weather['temperature']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

cv = RepeatedKFold(n_splits=10, n_repeats=5, random_state=42)
model = RidgeCV(alphas=arange(0, 1, 0.001), cv=cv, scoring='neg_mean_absolute_error')

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('y prediction : ', y_pred)
print('mean_absolute_error : ', mean_absolute_error(y_test, y_pred))
print('MSE minimize factor : ', model.alpha_)
```

Figure 6 - training model on Test-Train Split

Model Evaluation

Receiving the MAE and MSE back from our model we can see that there is a bit of a difference between what it is predicting and what the data should be. With an average of 3.98, this means that on average the model is roughly 4 degrees Celsius off what it should be. Demonstrating a reasonable amount of accuracy but there is still room for improvement.

```
mean_absolute_error :  3.986729115670314
MSE minimize factor :  0.999
```

Figure 7 - Evaluation Metric results of the model

To help understand the model's evaluation better, a scatter plot comparing the predicted values and actual values, and residual plots have been produced.

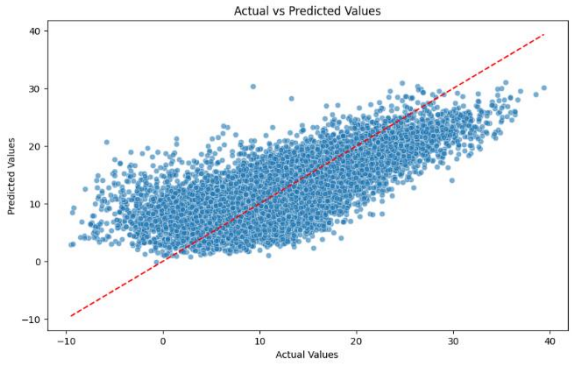

Figure 8 - Scatter plot

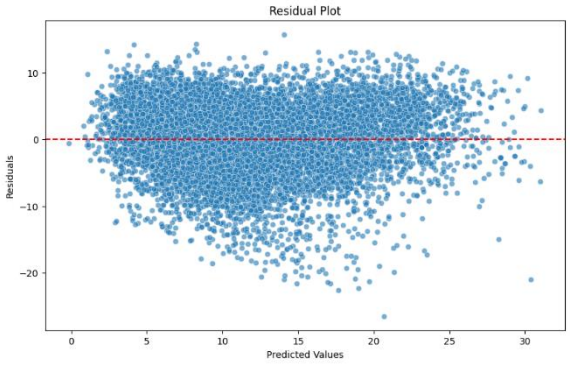

Figure 9 - Residual plot

What we can see from these graphs is that while there is a large mass in the correct areas, there is a wider range that has become outliers within the predictions, however, helps visualise how much 3.98 degrees is off as ideally all the dots should be perfectly centered on the red line.

Future Predictions

To make future predictions on the data we have, and to create some new data to work with, the first to do this we create new rows ranging over the next 30 days after the dataset ends, this led to the creation of an additional 744 rows.

To populate these new rows, we used the average of every day in the dataset that matches the new date (e.g. 12/12/2013, 12/12/2014,...,12,12/2016) to populate each of the features that are not the target variable. This gives the model something to work with.

We then put the information through our model, letting it run but this time we set a test_size to 0.016339797509498606 (= 774 / 45533) which is the last 744 items within the dataset or the next 30 days' worth of values.

The prediction was then plotted on a graph so that we can see the values and how they look compared to where the data left off.

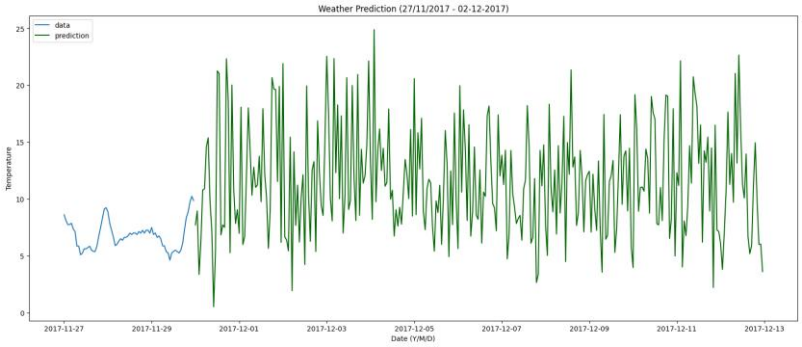

Figure 10 - predictions made by the model

Conclusion

To conclude, the Weather Prediction system was implemented through Data Cleaning and Preparation, Feature Engineering, and Training using Cross-Validation. And while the result was off by some degrees, there are still ways to improve this model. We could look at other alternatives to the Ridge Regression Model, RidgeCV, and Repeated K-Fold Cross-Validation, such as Lasso Regression, Elastic Net, Random Search Cross-Validation, and even investigate other model types such as Autoregressive integrated moving average (ARIMA) model. And other evaluation metrics like Root Mean Squared Error (RMSE) or Median Absolute Error.

Another Possibility to consider is the use of a larger dataset, while we had 45,254 spanning a few years, expanding this dataset with data from other sources could allow the model to get a better understanding of how the features correlate to one another.

Bibliography

BENIAGUEV, D. (2018). *Historical Hourly Weather Data 2012-2017*. Retrieved from Kaggle: https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data/data

Brownlee, J. (2020, August 26). *Repeated K-Fold Cross-Validation for Model Evaluation in Python*. Retrieved April 2024, from Machine Learning Mastery: https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/#:~:text=Repeated%20k-fold%20cross-validation%20provides%20a%20way%20to%20improve,mean%20result%20across%20all%20folds%20from%20all%20runs.

Giba, L. (n.d.). *Ridge Regression Explained*. Retrieved 04 2024, from machinelearningcompass.com: https://machinelearningcompass.com/machine_learning_models/ridge_regression/

Linear Regression in Machine Learning. (2024, August 7). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/ml-linear-regression/

mohi. (2020, September 18). *Implementation of Ridge Regression from Scratch using Python*. Retrieved from geeks for geeks: https://www.geeksforgeeks.org/implementation-of-ridge-regression-from-scratch-using-python/

sckit-learn. (n.d.). *RidgeCV*. Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeCV.html