Regionlabelling : Semi-automated Labeling System for Air Quality Data

Siddhi Belgamwar, Sonam, Aditya Kumar Patrayadi

September 11, 2024

Abstract

This project focuses on developing a semi-automated labelling toolbox for air quality data collected from geographically distributed sensors. The dataset records concentration levels for pollutants PM2.5, NO2, and PM10 across different regions. The goal is to assign labels in the form of AQI levels to locations or regions based on the Air Quality Index (AQI). It is divided into six levels from 1 to 6. The application first informs user about AQI levels and colour coded categories (Very Good, Good, Moderate, Poor, Very Poor, and Extremely Poor corresponding to each level) along with their description. The system allows for both manual and automated label assignment, as well as the ability to visualize data trends. It supports functionalities such as marking regions, removing and suggesting labels, and relabelling AQI levels. Furthermore, missing data imputation is handled using temporal and spatial methods. Spatial methods use nearest neighbours to compute mean pollutant value. They are also used while suggesting a label to the user. The developed system is designed with a user-friendly UI, enabling smooth interaction and decision-making based on air quality trends.

Keywords: air quality, region labelling, spatiotemporal data, AQI, label automation.

1 Introduction

Air quality is one of the most critical environmental indicators, as it directly affects human health and the environment. With increasing pollution levels, monitoring air quality has become essential, especially in urban areas. Data collected from sensors across different regions provide information on pollutant levels like PM2.5, PM10, and NO2, all of which contribute to assessing the overall air quality in an area.

This project aims to develop a semi-automated labeling toolbox that assigns air quality labels to these recordings based on the Air Quality Index (AQI). The AQI is a widely used standard that categorizes air quality into six levels, providing an easy-to-understand interpretation of how polluted the air is in a region. This

labeling system will allow stakeholders, such as city planners, environmental agencies, and the general public, to make informed decisions on managing air pollution and protecting public health.

2 Data

The dataset used in this project is provided by WHO. It contains 40,388 rows and 20 columns, including detailed pollutant concentration data from various locations over time. The main pollutants considered for the AQI calculation are:

- **PM2.5**: Fine particulate matter smaller than 2.5 micrometers, which can penetrate deep into the lungs and pose severe health risks.
- **PM10**: Coarse particulate matter smaller than 10 micrometers, which can cause respiratory issues and reduce air quality.
- **NO2**: Nitrogen dioxide, a harmful gas that is commonly produced by vehicles and industrial processes, contributing to smog and acid rain.

It contains columns 'who_region', 'iso3', 'country_name', 'city', 'year', 'version', 'pm10_concentration', 'pm25_concentration', 'no2_concentration', 'pm10_tempcov', 'pm25_tempcov', 'no2_tempcov', 'type_of_stations', 'reference','who_ms','latitude', 'longitude','population','population_source','weblink'

3 Pre-processing

Data preprocessing is a crucial step in ensuring that the dataset is clean, well-structured, and ready for analysis. The dataset used for this project contained a number of columns that were deemed irrelevant and missing values, which required cleaning and transformation before the labeling process could be applied.

3.1 Libraries Used

We employed a combination of Python libraries for data manipulation, imputation, and visualization:

- **pandas** and **numpy** : For data manipulation and handling.
- **KNNImputer (from sklearn.impute)**: For imputing missing values using K-Nearest Neighbors.
- **StandardScaler (from sklearn.preprocessing)**: For standardizing features.
- **cdist (from scipy.spatial.distance)**: For computing distances between data points.

- **NearestNeighbors (from sklearn.neighbors)**: For finding nearest neighbors in spatial imputation.
- **matplotlib:**: For visualizing data distributions (e.g., histograms),

3.2 Dataset Size and Column Selection

The dataset initially contained 40,388 rows and 20 columns. However, not all columns were found relevant to the objectives of this project, therefore following columns were removed.

- iso3, version, pm10 tempcov, pm25 tempcov, no2 tempcov, type of stations, reference, web link, population, population source, who ms.
- 3 rows are dropped because they have missing year. Cannot be used in the application as we cannot map them to any year between 2010 to 2022.

3.3 Data Cleaning and Reformatting

Several steps were taken to clean and reformat the dataset:

- **Reformatting Column Values**: The who region column values were reformatted for better readability, mapping numeric codes like 4 Euro to Eur and 6 Wpr to Wpr.
- **City Name Conversion**: Some city names were in non-English languages, such as Korean and Chinese, and these were converted into English equivalents for consistency (e.g. . was converted to “Incheon”).
- **Handling Missing Years**: Three rows were dropped from the dataset as they contained missing year data, making it impossible to map them to the time period between 2010 and 2022.
- **Correcting Country Names**: For example, the country name “Turkey” was corrected to the appropriate English spelling.
- **New columns based on pollutant value added**: .New columns such as ”pm25_imputed”, ”pm10_imputed”, ”no2_imputed” were added with a value of ”yes” if the pollutant value is missing, and ”no” if the value is present to the dataset. This was done to inform user if the pollutant concentration value used is authentic or artificially generated.
- **Previous approach**: Initial approach was to impute 30000 new rows in the dataset to ensure that every city will have data from years 2010 to 2022. This was discarded because almost half the data would be artificial. It also caused performance issues (like rendering speed of the data for heatmap). So the original data size was retained as it is and missing values were handled for the same.

- **Outliers removal :**Outliers were not eliminated because we need to have data such that cities with AQI levels from 1 to 6 would be available. Handling of outliers (removal or squashing in a defined range) resulted in only 4 levels which was incorrect.
- **Handling missing values:** Missing data is common in large datasets, especially when dealing with time-series data from multiple sensors. We applied two main techniques to handle missing pollutant values: .
  - **Mean calculation and Temporal Imputation**Calculate mean for each city with partial missing values.For cities with some of the missing values, we use mean value of each city to handle them. Empty cells are filled with mean pollutant value of that city
  - **Mean calculation and Spatial Imputation**: For cities with missing pollutant data for entire columns (e.g., missing PM2.5 data for a city for all years), we used K-Nearest Neighbors (KNN) to impute missing values. The KNN algorithm identifies neighboring cities with similar geographic features (using latitude and longitude) and fills in the missing data based on their values. The neighbourhood is determined based on optimum distance and number of neighbours computed.
  - Distances are calculated using Euclidean distance.
  - **Important Considerations:**
    - Accuracy : Euclidean distance is not accurate for long distances on Earth's surface because it does not account for the Earth's curvature.
    - For larger distances, the Haversine formula or other geodesic distances are considered. But for our ease of calculations, we are using Euclidean distance.
    - Coordinate System: Ensure that the latitude and longitude are in the same coordinate system and units.
- **Using Approximate Nearest Neighbors:**
  - To avoid calculating the full distance matrix, we have used approximate nearest neighbours method NearestNeighbors from scikit-learn with a smaller n_neighbors parameter.
  - Using NearestNeighbors for finding approximate distances is more memory-efficient for large datasets.
  - Approximate Nearest Neighbors (ANN) methods are designed to find nearest neighbors more efficiently than exact algorithms, especially in high-dimensional or large datasets. They trade off some accuracy for significantly reduced computational time and memory usage, which is particularly useful when dealing with very large datasets or when real-time performance is required.

3.4 Interpreting the Histogram to Find Optimum Threshold Distance

- We look for regions where the histogram shows significant peaks or troughs. These indicate natural groupings of distances.
- **Threshold Selection**:
  - **Significant Drop:** A significant drop in frequency after a certain distance can be a good point to choose as the threshold.
  - **Percentiles:** Percentile lines help to choose thresholds that include a certain percentage of nearest neighbours. For example, the 75th percentile includes the closest 75% of neighbours.

3.5 Steps to Find Optimum Number of Neighbours

- **Define the Threshold Distance:** Decide on a reasonable threshold distance to use for finding neighbours. This distance can be based on domain knowledge, geographic distribution, or exploratory analysis.
- **Calculate Neighbours and Distances:** Use the `NearestNeighbors` class from `scikit-learn` to find neighbours within the threshold distance.
- **Analyse the Distribution of Number of Neighbours:** Plot the distribution of the number of neighbours for each data point to understand how many neighbours are typically found.
- **Determine the Optimum Number of Neighbours:** Look for a number of neighbours that captures enough data points without including too many distant or less relevant neighbours. The median value is chosen here.

3.6 Handling Cities with Fewer Neighbours

- Consider all the available neighbours.

3.7 Handling Cities with More Neighbours than the Optimum Number

- Trim to the optimum number to include only the closest ones. This ensures that only the most relevant and closest neighbours are used for imputation.

3.8 Mean Pollutant Value Computation

- Iterate through each year, and for every city with missing values, find the neighbours and use their respective pollutant concentration for that year to compute the mean pollutant value to handle the missing data.

3.9 Addition of Label Columns

- Once the dataset is complete, calculate AQI Levels and Categories, and add those columns to the dataset. Categories and levels are determined using the standard concentration values of pollutants. Our dataset includes pollutants PM10, PM2.5, and NO2, and the AQI levels are derived from the European Air Quality Index calculation website.

POLLUTANT	INDEX LEVEL *(based on polluant concentrations in µg/m3)*					
	1 **Very good**	2 **Good**	3 **Medium**	4 **Poor**	5 **Very Poor**	6 **Extremely Poor**
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Sulphur dioxide (So_2)	0-100	100-200	200-350	350-500	500-750	750-1250
Particules less than 10 µm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Particules less than 2.5 µm ($PM_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800

Note: PM10 and PM2.5 values are based on 24-hour running means

Figure 1: AQI Level and Category Determination

4 Health Impacts of AQI Levels

The six AQI levels correspond to varying degrees of air quality and associated health risks:

- **Level 1 (Very Good)**: Air quality is excellent, and there are no known health risks for the general population.
- **Level 2 (Good)**: Air quality is generally acceptable, though sensitive individuals may experience minor respiratory symptoms.
- **Level 3 (Moderate)**: Sensitive groups, such as children and the elderly, may experience health effects, but the general population is unlikely to be affected.
- **Level 4 (Poor)**: Health effects may be felt by sensitive groups, and some individuals in the general population may also experience discomfort.
- **Level 5 (Very Poor)**: Health warnings are issued for the entire population. Prolonged exposure may cause more severe health effects.

- **Level 6 (Extremely Poor)**: The air quality is hazardous, and everyone may experience serious health effects. Long-term exposure at this level can lead to chronic health conditions.

This classification system helps public health officials issue advisories and informs the public about the risks associated with different levels of air pollution.

5 UI Design

5.1 Technologies and Libraries Used

The user interface was designed with modern web development tools and libraries to ensure a smooth and interactive user experience:

- **Angular**: A popular framework for developing dynamic single-page applications. It was used to build the front-end components of the system.
- **CSS and HTML**: Styling and layout of the application were done using standard web development technologies.
- **geojson and Folium**: These libraries were used to handle geographic data and visualize AQI levels across different regions in an interactive map.

5.2 Features of the UI

The UI was designed to allow users to upload datasets, select specific cities or regions, and view AQI data in real-time. Key features include:

- **Map Visualization**: Users can select geographic regions on an interactive map to view AQI levels or adjust labels for specific cities.
- **Dropdown Menus**: Year selection and other options allow users to filter data based on specific time periods.
- **Real-time Feedback**: When a city is selected, its AQI data is displayed dynamically, and any changes made to labels are reflected immediately on the map.

6 Frontend Development

The following tasks were undertaken for the development of the frontend interface:

6.1 app.jsx

- **Purpose**: This file manages and displays either a cover screen or a world map depending on the application's state. It serves as the central hub for switching between the introductory and map views.

- **Libraries Used**:
  - React: For managing state and rendering components.
  - GlobalStyles: For applying global CSS styles.
  - Ant Design (commented): For UI components like loading spinners.
  - Plotly (commented): For creating interactive charts and maps.

6.2 Frontend includes the following components

- Ant Design : provides ready-made UI components for React, like buttons, forms, tables, and modals.UI charts are used for Data visualization.
- React
- charting
- chat interface,
- API calls via Axios. API calls using Axios for server communication and fetching data from API.

6.3 For different map-related data operations, we have used axioms for HTTP requests to a server.

- **getMapData:** Sends a POST request to fetch map data for a selected year.
- **getHighlightedMapData**:Sends a POST request to fetch highlighted map data based on the selected year, data for dataframes (dataForDf), and whether to show an overlay.
- **getHighlightedMapDataWithUpdatedMarker:** Sends a POST request to update highlighted map data with a selected city and the updated data.
- **getSuggestLabelAQI:**Sends a POST request to suggest labels for AQI based on the selected year and city.
- Button labeled "Start" initiates interaction with the main application.
- **uploadFile:** Uploads a file using FormData to the server.
- **downloadData:** Sends a POST request with JSON data to download a file.
- **getYears:** Sends a GET request to retrieve the list of available years.

6.4 Functionalities:

- **Cover component:** creates a full-screen splash screen which contains information about Air Quality Index (AQI) categories and a "Start" button using Ant Design's Button component.
- **HighlightMap component:** uses Ant Design's UI elements to interact with a map and file management system, providing functionality for uploading, downloading, and viewing a map configuration for a selected year.
- **Instructions component:** provides guidance to the user about how to use your AQI labeling map application, using Ant Design's Modal and FloatButton components.
- **LineGraphs component:** handles rendering a bar chart using MUI's BarChart component, which visualizes pollutant (NO2, PM10, PM2.5) concentration data for different cities which are flitered out after selecting the region by the user.
- **RelabelAQI component:** user is allowed to upload, download, and relabel Air Quality Index (AQI) data for selected cities in a region. The component interacts with the server to fetch suggested AQI values, update AQI values for individual cities or entire regions, and display a map with highlighted data.
- **worldMap component:** utilizes a multi-step process to select a year, highlight a region on the map, and relabel AQI (Air Quality Index) values for the selected region(s).

6.5 Important Components:

- **currentStep**: Tracks the current step in the multi-step process using the navigation bar.
- **Navigation Bar:** Steps component is used to render a step navigation bar, which includes three steps.
- **Select Year:** The user selects a year.
- **Highlight:** The user highlights a region on the map.
- **Relabel:** The user relabels AQI values for the selected region(s).
- **showInstructions:** Controls the visibility of a modal with instructions.
- **mapConfiguration:** Stores the selected year, city, map highlighting status, master data (AQI data for all cities/years), and filtered data (AQI data for selected cities).

- **Instructions Modal:** allows the user to view guidance on how to use the tool throughout the application.
- **Yearselection:** This component allows users to select a year from a dropdown menu. Once a year is selected, the heat map will be updated as per the data.

7 Backend Development

The web app is built using Flask. Backend contains two main files: app.py and util.py. The application handles data related to air quality (AQI) and provides various routes for retrieving and visualizing data, including maps generated with Folium. It enables user engagement with the application.

7.1 app.py - Main Application Script

This file serves as the main script for the Flask web application. It imports the necessary libraries, manages backend services such as data retrieval, file operations, and map visualization, and defines several routes to handle requests and responses related to AQI data.

7.1.1 Libraries Used:

- **Flask, rendertemplate, request, jsonify (from Flask):** Core components for creating routes and handling requests and responses in the web application.
- **Markup (from markupsafe):** Used to render HTML in the responses.
- **pandas (as pd):** Used for data manipulation and handling CSV files.
- **CORS:** Enables cross-origin resource sharing, allowing the API to be accessed from different domains.
- **Polygon (from shapely.geometry):** Handles geographic shapes for map visualization.
- **util:** Contains custom utility functions for map generation and data processing.
- **os, json, warnings:** General-purpose Python libraries used for various utility functions.

7.1.2 Global Variables:

- **app:** The Flask application object.
- **data:** AQI data loaded from a CSV file (‘aqidata.csv‘).

7.1.3 Route Descriptions:

- **/years (GET):** Retrieves the unique years available in the dataset and returns them in JSON format.
  - **Function:** ‘getYears()‘
  - **Returns:** JSON object containing a list of unique years.
- **/masterData (POST):** Accepts a selected year as input, filters the dataset for that year, and returns AQI data along with an HTML map generated using Folium.
  - **Function:** ‘home()‘
  - **Returns:** JSON containing AQI data and the generated HTML for the map.
- **/highlighted (GET, POST):** Handles requests to highlight a marked region on the map. Processes the user input and updates a GeoJSON file to reflect highlighted areas.
  - **Function:** ‘highlighted()‘
  - **Returns:** JSON containing updated AQI data and the generated map HTML.
- **/uploadFile (POST):** Uploads a file to the server and saves it as ‘data.csv‘.
  - **Function:** ‘uploadFile()‘
  - **Description:** Accepts a file from the user, saves it as ‘data.csv‘ in the server directory, and returns a status code.
- **/downloadFile (POST):** Saves the provided JSON data as a CSV file.
  - **Function:** ‘downloadFile()‘
  - **Description:** Converts JSON data received from the client into a DataFrame, prints the first 10 rows, and saves it as ‘data.csv‘.
- **/suggestLabel (POST):** Suggests an AQI value based on a given city and year by using nearby cities’ pollutant levels.
  - **Function:** ‘suggestLabel()‘
  - **Description:** Takes a city and a year, filters AQI data for the selected year, and computes a new AQI level using the nearest neighbors. The list of nearest neighbors is also returned.
  - **Returns:** Suggested AQI value and a list of neighboring cities.
- **/updateHighlightedMap (GET/POST):** Updates a map by highlighting regions or cities based on the provided data, year, and city.

- **Function:** ‘updateHighlightedMap()‘
- **Description:** Accepts a year, city, and data, then updates the map to highlight regions or cities within a given polygon using GeoJSON data.
- **Returns:** A JSON object containing the HTML representation of the updated Folium map.

7.1.4 Error Handling:

Each route includes exception handling to return a 500 response code if something goes wrong.

7.1.5 Map Generation:

- The map updates are handled using Folium, and geographic calculations are performed using `Polygon` from the Shapely library.
- Highlighted areas are determined by checking whether the data points fall within a polygon.

7.1.6 Data Storage:

- Files are stored as `data.csv` and `data.geojson` on the server.
- These functionalities allow users to upload/download files, request AQI suggestions, and update maps dynamically.
- Folium is used for map rendering, and GeoJSON is used for geographic data management.

7.2 util.py Utility Functions

This file contains helper functions that are used within the main application to process data and generate visualizations. These functions are called within the routes defined in `app.py` and include operations for file reading, data imputation, and label suggestion.

- **Key Functions**: Contains utility functions for data processing and handling. This includes operations for file reading, data imputation, and label suggestion.
  - **create_folium_map(location, data)** Generates a Folium map centered at a given location and overlays markers based on AQI data for each city.
    - **Parameters:**
      - **location:** Coordinates to center the map.
      - **data:** AQI data for various cities.
    - **Returns:** HTML representation of the generated Folium map.

- **calc_aqi(pm10, pm25, no2)** Calculates the AQI level based on the concentrations of pollutants PM10, PM25, and NO2.
  - **Parameters:**
    - **pm10:** PM10 concentration.
    - **pm25:** PM25 concentration.
    - **no2:** NO2 concentration.
  - **Returns:** AQI level (integer) based on the highest risk pollutant.
- **compute_new_aqi(city_name, df, k=3)** Computes a new AQI level for a city by averaging the values of the nearest neighboring cities if the data for the city is missing.
  - **Parameters:**
    - **city_name:** Name of the city for which AQI is being recalculated.
    - **df:** DataFrame containing AQI data for all cities.
    - **k:** Number of nearest neighbors to consider (default is 3).
  - **Returns:** Recalculated AQI value and the list of neighboring cities used.
- **getColorByAQI(aqi_value)** Returns the color code based on the AQI level, used for map visualization.
  - **Parameters:**
    - **aqi_value:** AQI level (integer).
  - **Returns:** Color corresponding to the AQI value.

7.3 General Workflow:

- Users send requests to the Flask API, such as retrieving AQI data for a specific year or updating highlighted regions on the map.
- The app processes the data using Pandas and utility functions from util.py.
- The results, including AQI data and map visualizations, are returned to the user as JSON responses or rendered HTML.
- This structure allows for an interactive application where users can visualize air quality data dynamically.

8 Future Work and Scope

8.1 Future Work

In this section, we outline the potential avenues for future work that can build upon the current project. Some of the key areas for future development include:

- **Enhanced Data Collection:** Expanding the dataset to include additional regions and pollutants to improve the robustness and accuracy of the AQI predictions.
- **Real-Time Data Integration:** Incorporating real-time data feeds to provide up-to-date air quality information and dynamic map updates.
- **User Interface Enhancements:** Improving the user interface to include more interactive features, such as customizable map layers and advanced filtering options.
- **Advanced Analytics:** Implementing more sophisticated analytical models and machine learning algorithms to enhance the prediction capabilities and provide deeper insights.
- **Scalability:** Developing solutions to scale the system for larger datasets and higher user loads, potentially by leveraging cloud services.
- **Cross-Platform Support:** Expanding the project to support multiple platforms and devices, including mobile and desktop applications.

8.2 Scope of the Project

The current project provides a solid foundation for monitoring and analyzing air quality data. However, the scope can be extended in several ways:

- **Geographical Expansion:** Extending the geographical coverage of the project to include more diverse locations and international cities.
- **Integration with External APIs:** Integrating with external data sources and APIs to enrich the dataset and provide additional context for the air quality information.
- **Public Engagement:** Developing features that facilitate public engagement, such as educational resources on air quality and interactive dashboards for community use.
- **Collaborative Research:** Partnering with academic and research institutions to conduct studies on the impact of air quality on public health and the environment.
- **Regulatory Compliance:** Ensuring that the system adheres to relevant regulations and standards for data privacy and environmental monitoring.

9 Conclusion

This project provides a robust framework for visualizing air quality data, assigning labels based on AQI, and allowing manual and automated intervention. The system can handle data input, process it effectively, and display actionable insights for decision-makers.

References

[1] React - A JavaScript library for building user interfaces.
https://reactjs.org/

[2] Styled-components - Visual primitives for component-based styling.
https://styled-components.com/docs/api#createglobalstyle

[3] Ant Design - The world's second most popular React UI framework.
https://ant.design/

[4] Plotly - Graphing libraries for making interactive, publication-quality graphs online.
https://plotly.com/

[5] Axios - Promise-based HTTP Client for the browser and Node.js.
https://axios-http.com/

[6] Flask - A lightweight WSGI web application framework in Python.
https://flask.palletsprojects.com/

[7] Pandas - Python Data Analysis Library.
https://pandas.pydata.org/

[8] Flask-CORS - A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.
https://flask-cors.readthedocs.io/en/latest/

[9] Shapely - A Python package for manipulation and analysis of planar geometric objects.
https://shapely.readthedocs.io/en/stable/

[10] Folium - Python Data, Leaflet.js Maps.
https://python-visualization.github.io/folium/

[11] GeoJSON - Format for encoding a variety of geographic data structures.
https://geojson.org/

[12] Python - Official site of the Python programming language.
https://www.python.org/

[13] NumPy - The fundamental package for scientific computing with Python.
https://numpy.org/

[14] Scikit-learn - A machine learning library for Python.
https://scikit-learn.org/

[15] Bootstrap - Open-source toolkit for developing responsive, mobile-first websites.
https://getbootstrap.com/

[16] MarkupSafe - A library for safe HTML and XML handling.
`https://palletsprojects.com/p/markupsafe/`