Project 5 Pollution Vision

This manuscript (permalink) was automatically generated from Saran-Wang/dsproject@e7b1d92 on December 7, 2020.

Authors

- **Shiyuan Wang**
  · Saran-Wang
  Department of Civil and Environmental, University of Illinois

- **Weiqi Ni**
  · weiqini
  Department of Civil and Environmental, University of Illinois; Department of Environmental and Resources, Zhejiang University

- **Gemma Clark**
  · 441gclark
  Department of Civil and Environmental, University of Illinois

- **Xueao Li**
  · XueaoLi
  Department of Civil and Environmental, University of Illinois

Abstract

Introduction

Literature Review

There are many studies using digital camera and advanced algorithm to estimate the concentrations of Particulate Matters. Hong et al. [1] developed a novel method of predicting the concentrations and diameters of outdoor ultrafine particles using street-level images and audio data in Montreal, Canada. Convolutional neural networks, multivariable linear regression and genralized additive models were used to make the predictions. Wong et al. 2007 present an image processing method for estimating concentrations of coarse particles (PM10) in real time using pixels acquired by an internet video surveillance camera. In this paper, the authors present formulas for predicting particulate matter based on optical physics including light absorption, scattering, and reflection. They do not use machine learning tactics to estimate pollution concentrations, but their model results in root mean square error values of around 4 μg/m3. Liu, Tsow, Zou,& Tao (2016) [2] conducted the following steps to make use of images to predict the air pollution concentration: ROI (region of interest) selection, extraction for image features, support vector regression for model training and predicting.

Exploratory Data Anlysis

1. Variables Explanation

Table 1: Variables Explanation

Data Fields	Explanation
Temp(C)	ambient temperature
Pressure(kPa)	air pressure
Rel. Humidity	relative humidity
Errors	if the air measurement equipment has error during sampling (0=no)
Alarm Triggered	if any instrumental warning shows during sampling (0=no)
Dilution Factor	an instrumental parameter (should close to 1)
Dead Time	another instrumental parameter (ideally close to 0)
Median, Mean, Geo. Mean, Mode, and Geo. St. Dev.	parameters describe particle sizes, which can be ignored
Total Conc.	an output variable from the instrument that should not be used
image_file	the visual information of the traffic condition, corresponding to an image in the “frames” directory
Wind_Speed	the wind velocity during sampling
Distance_to_Road	the distance between camera and road
Camera_Angle	the angle of incidence between the camera and the road
Elevation	the elevation between the camera and the breathing zone
Total	the total measured particle number concentration (# / cm3) This is the dependent variable

2. Data Cleaning

Delete the useless columns in the dataset

- The first column titled unnamed is meaningless.
- The columns titled Median, Mean, Geo. Mean, Mode, and Geo. St. Dev. are parameters describing particle sizes, which can be ignored.
- The column titled “Total Conc.(#/cm³)” is an output variable and should not be used.

Delete the rows with equipment error during sampling

- train = train[train['Errors'] == 0].reset_index(drop=True), only keep the rows with no error (value = 0)

- train = train[train['Alarm Triggered'] == 0].reset_index(drop=True), only keep the rows with no warning (value = 0)

3. Visualization of the distributions of varibales

Figure 1 shows that "Wind_Speed", "Camera_Angle", "Distance_to_Road" and "Elevation" are all in discrete distributions, while "Temp(C)" are in continuous distribution. "Pressure(kPa)" has four clusters. It should also be noted that the "Dead Time" almost shares the same distribution as "Total".

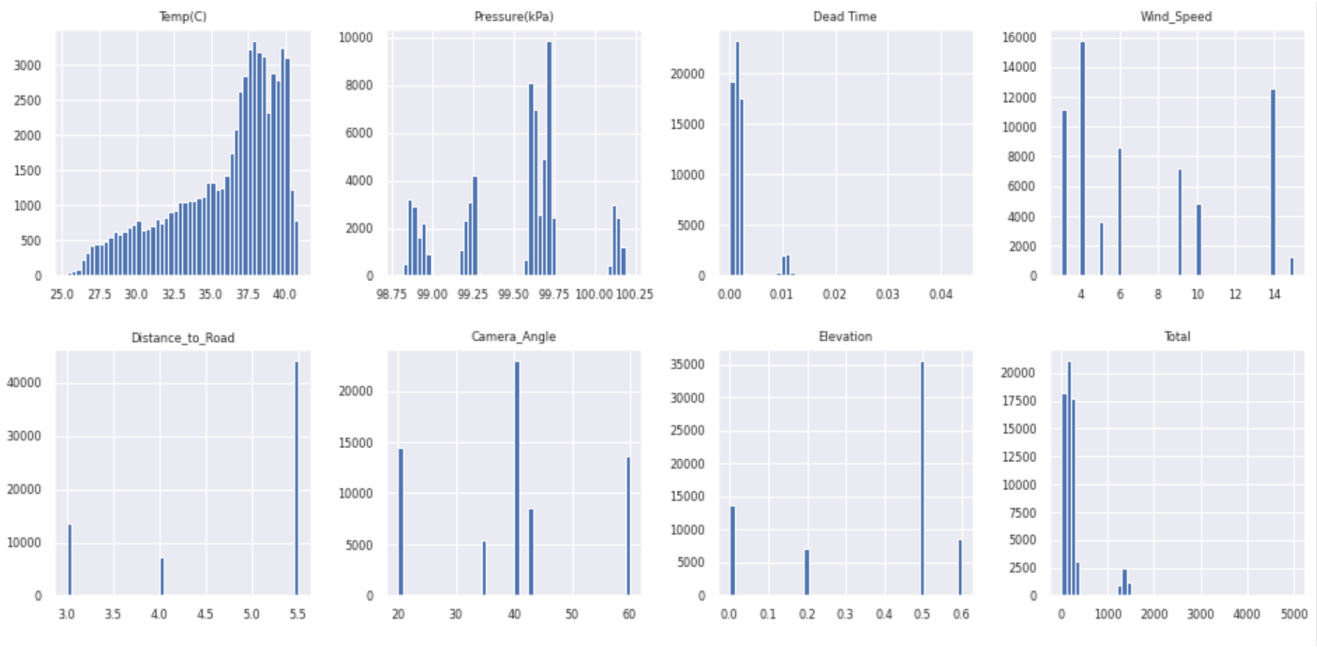


Figure 1: Variables Distribution

4. Correlations among variables

From the correlation map 2 we could see that "Dead Time" are extremely correlated with "Total", with a coefficient of 1, followed by "Camera_Angle", "Pressure(kPa)" and "Distance_to_Road", with coefficient of 0.52, 0.49, 0.44 respectively. Here you may be curious why "Dead Time" could be so closely related to "Total", and there is one possible explanation: Actually, "Dead Time" is an instrument parameter, and if there are more PM concentrations in the air, the instrument need more time to process, and vice versa.

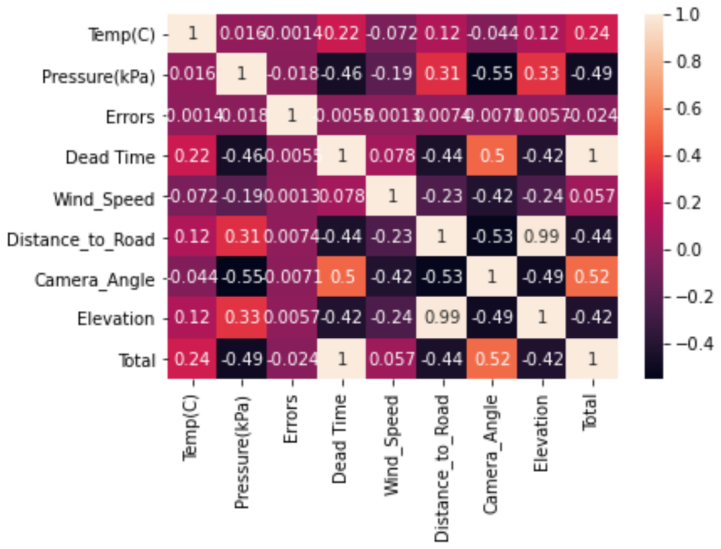


Figure 2: Variables Correlations

5. Visualize the total concentration based on different dates

Extract the total particle concentration data based on different dates and then visualize it. Basic steps are as below:

- Extract the date information from the column called “Image_file”: taking the image “video08052020_2771.jpg” as example, we will extract the date “0805”.
- Add one column named “Date”: 0805
- Group by “Date” and plot the date-based concentration diagram

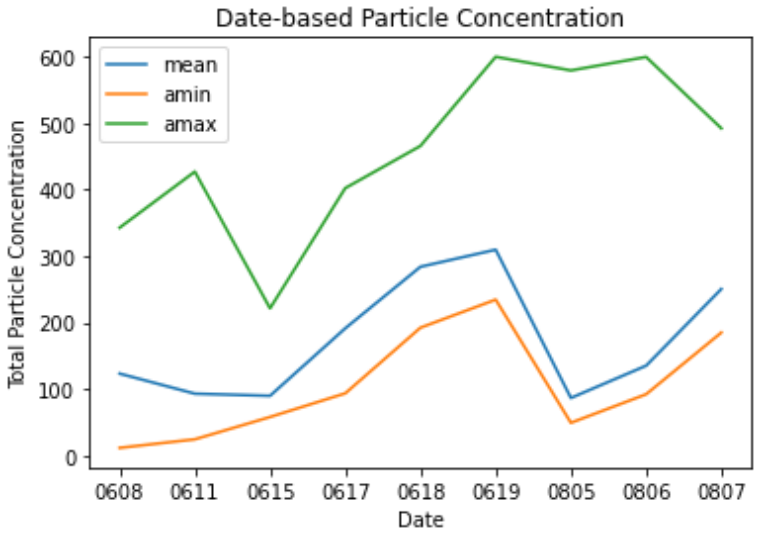


Figure 3: Date-based Particle Concentration

Model

Shiyuan's Model

My model setup splits into two part, the first is image data extraction, the second is the selection of appropriate model to fit this dataset.

Image Extraction

First I want to digitize images by extracting image features, there are mainly 6 features I want to extract: RGB, image luminance, image contrast, image entropy, transmission and amount of haze removed and number of cars on streets.

1. RGB

The RGB color model is one of the most straightforward parameters describing an image. Intuitively, in this case, we may expect more blueness and greenness if the PM concentrations are low since the color of tree and sky would be brighter when the air conditions are good. For each image, after deriving the RGB of each pixel, we take the average of them, and then divide each value by 255 to normalize it. The figure below 4 shows the distributions of RGB in this dataset. We can see that they are nearly normally distributed with mean 0.45, 0.55 and 0.35 respectively. For blueness, we could see a second peak at around 0.42.

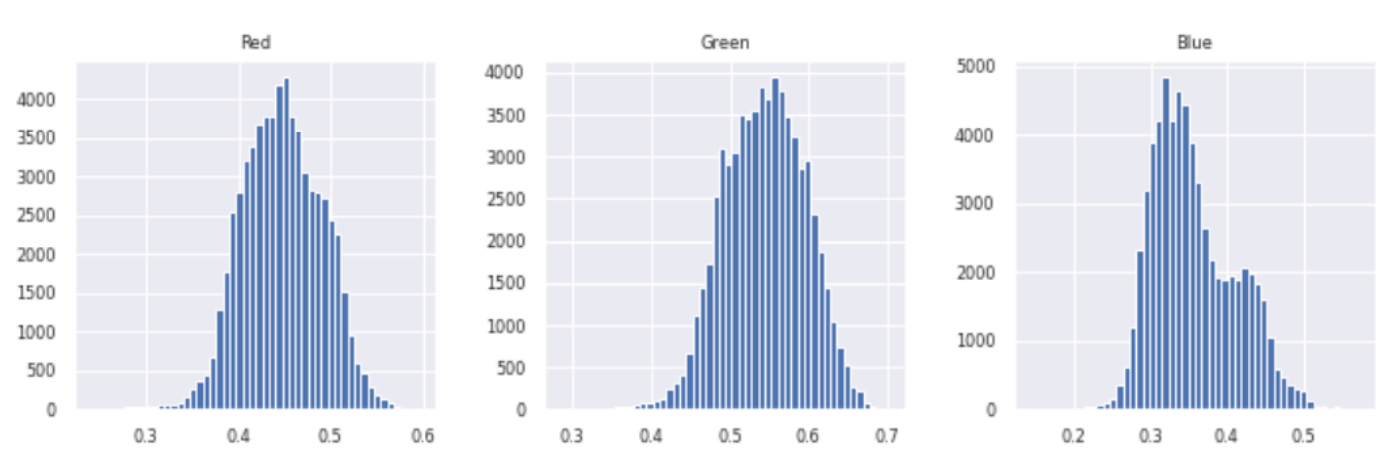


Figure 4: RGB Distribution

2. Luminance

Like RGB, luminance is also a very basic parameter describing an image, which could be an indicator of how bright the image will appear. The luminance of each image is calculated by taking the average of the luminance intensity of each pixel. From figure 5 we could also see that it's also normally distributed with a mean of around 130.

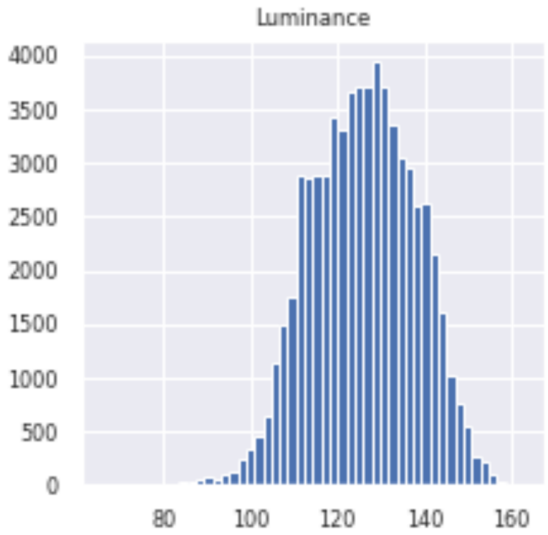


Figure 5: Luminance Distribution

3. Contrast

The image contrast is defined as the difference between the max and min luminance intensity of an image. Study [2] shows that the higher the PM concentrations, the lower contrast would be. It makes sense since the image would become vague and lighter when there are more particulate matters in the air. And often, one image would have pixels with the highest intensity of 255, as well as the lowest intensity of 0. Therefore, we can't see much difference if we want to derive the absolute contrast, since it would be 1 for most of those images. Therefore, we use root mean square of image intensity to describe image contrast.

$$Absolute_Contrast = \frac{I(i_{max}, j_{max}) - I(i_{min}, j_{min})}{I(i_{max}, j_{max}) + I(i_{min}, j_{min})} \quad (1)$$

$$RMS_Contrast = \sqrt{\frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(I(i,j) - avg(I))^2} \quad (2)$$

where I(i,j) is luminance intensity at (i,j) pixel.

From figure 6 we could see that the distribution is a little bit right-skewed with a small peak at around 35, and a larger one at around 50.

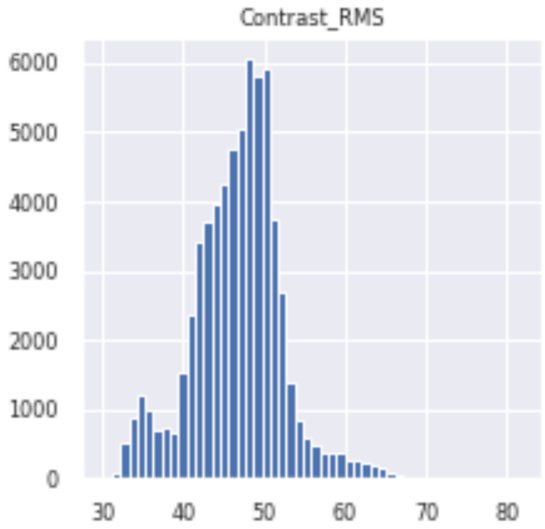


Figure 6: Contrast Distribution

4. Entropy

Image entropy is a statistical measure of randomness that quantifies information contained in an image. Usually, an image would lose its details with the increasing PM concentrations, and the image entropy will decrease as a result [2] . To do the calculation, I first converted the original RGB image to grayscale image and then used a module within python: *skimage* to calculate image entropy directly. The example code is shown as below:

```
colorIm =
        Image.open('../input/pollutionvision/frames/frames/video06082020_0.jpg

greyIm = colorIm.convert('L')
ImContrast = skimage.measure.shannon_entropy(greyIm)
```

Figure 7 is the original figure and figure 8 shows its entropy.

Figure 7: Original Image

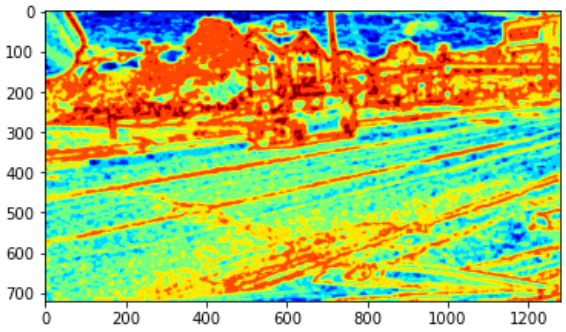


Figure 8: Entropy

5. Transmission and amount of haze removed

6. Number of cars on streets

Also, I take the number of cars on streets into account. Intuitively, the more cars on the street, the higher PM concentrations would be. There is a very efficient library in python called *cvlib*, within which a function called *object_detection* could detect the number of different objects appearing on an image. The example code is shown as below:

```
im =
        cv2.imread('../input/pollutionvision/frames/frames/video06082020_0.jpg
bbox, label, conf = cv.detect_common_objects(im)
output_image = draw_bbox(im, bbox, label, conf)
number_of_car = label.count('car')
```

As we can see in figure 9, the left image has 2 cars on street, and we can detect exactly two cars; while figure 10 has no car on street but 5 cars parking in the parking lot, and we could detect 5 cars.

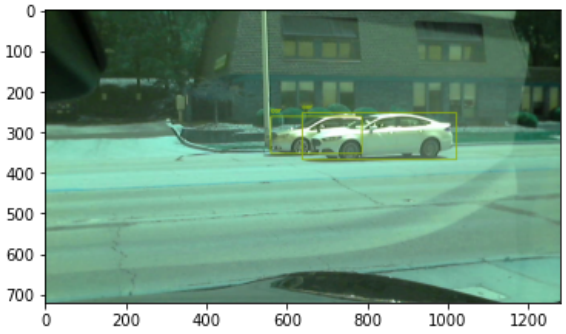


Figure 9: Two cars detected

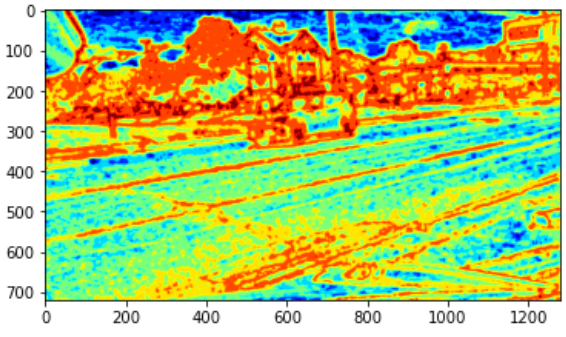


Figure 10: Five cars detected

Here comes the problem, this function could only detect the number of cars appearing on an image but can't identify which is in motion. But the moving cars are actually the ones which contribute to PM concentrations at the very moment. However, in this case, I just keep the original detection results, since if there are more cars in the parking lot, I just assume it's a traffic busy day, on which the PM concentrations would be higher than normal days.

7. Correlations among variables

Here we plot out the spearman correlations among those features in figure 11, the last column shows the spearman correlations between each feature and the Total PM concentrations. As we can see, the dead time, which is an instrument parameter, is closely correlated with PM concentrations, followed by Pressure, RGB, luminance and temperature. However, since the dataset is rather complicated, the correlations may mean nothing. Actually, the different combination of different features may have various impacts on the results of our model. And the correlations just provide us with a straightforward perception.

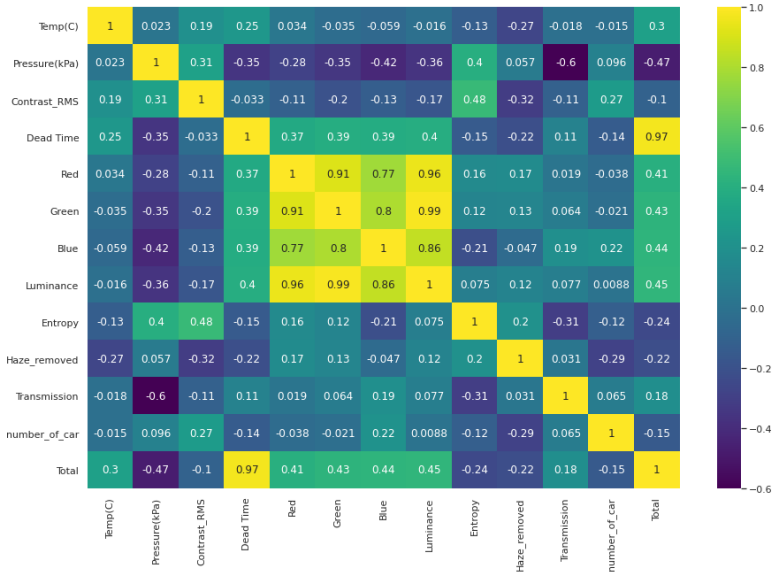


Figure 11: Variables Correlations (include digitized image data)

Model Selection

As mentioned before, we can't select the features barely based on their correlations with PM concentrations, since I have both numerical data and digitized image data, which could be very complicated. Therefore, I selected different combinations of features and run the model several times to select the one with best performance. At first, I tried the Neural network, but it doesn't do well in this dataset, with an MSE of around 800. Then I calculated the model accuracy using *cross_val_score*, and then I switch to random forest, which gives me an accuracy of 0.997. With the help of * GridSearchCV*, I could decide on the parameters for random forest: `RandomForestRegressor(max_depth=20, n_estimators=1000, random_state=3)`. With those parameters, I could get an RSME around 11.

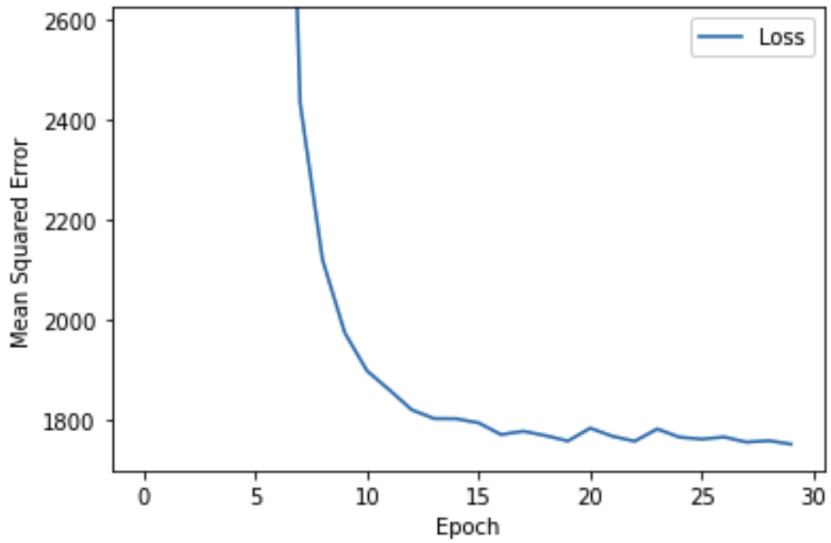
Gemma's Model

I used three different approaches to predict pollution concentrations: building a neural network using the numeric data, creating a neural network using the image data, and developing a random forest model using the numeric data.

For all three models, I performed the same first initial steps. I read in the "csv" files containing the training dataset and test dataset and removed the variables that should not be included in the model. These variables included parameters related to the particulate size and shape, superfluous instrument parameters, and variables with a standard deviation of 0. For each of the 64961 data points, there was one image and eight numeric variables: temperature, pressure, errors, dead time, wind speed, distance to road, camera angle, and elevation. There were no missing values in the dataset. I then randomly split the training dataset into a validation dataset (20% of the original training dataset) and a new training dataset (80% of the original training dataset).

Neural Network (Numeric Data)

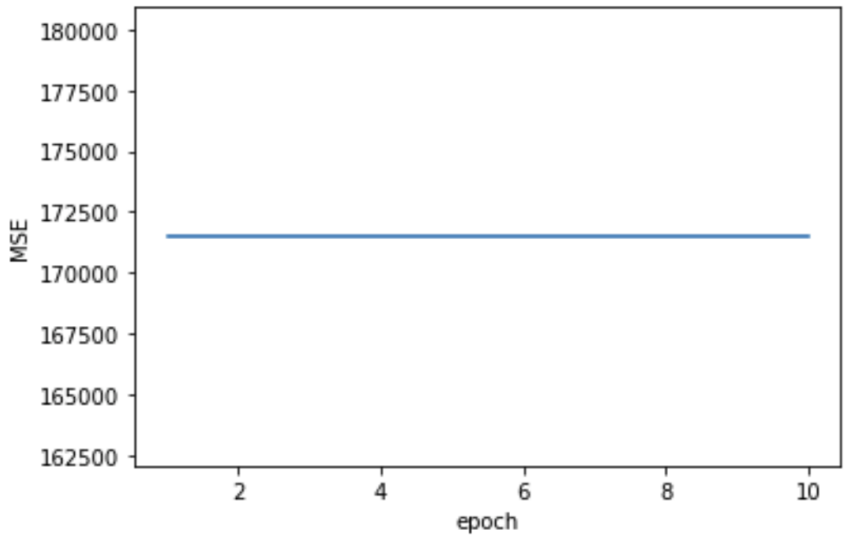
To prepare the numeric data for the neural network model, I set the eight numeric variables to be the independent "x" variables and the total pollution to be the dependent "y" variable in a tensorflow dataset. I set the batch size to be 50 and created a "Sequential" keras model. My model had four layers: two "relu" layers with 30 units each, a "sigmoid" layer with 30 units, and a linear layer with 15 units. Using a learning rate of 0.0005, loss of mean square error, and 30 epochs, I compiled the model and tested it on the validation dataset. The mean square error converged at around 1,500-1,800 depending on the random training and validation dataset generated in the initial setup.



Convolutional Neural Network (Image Data)

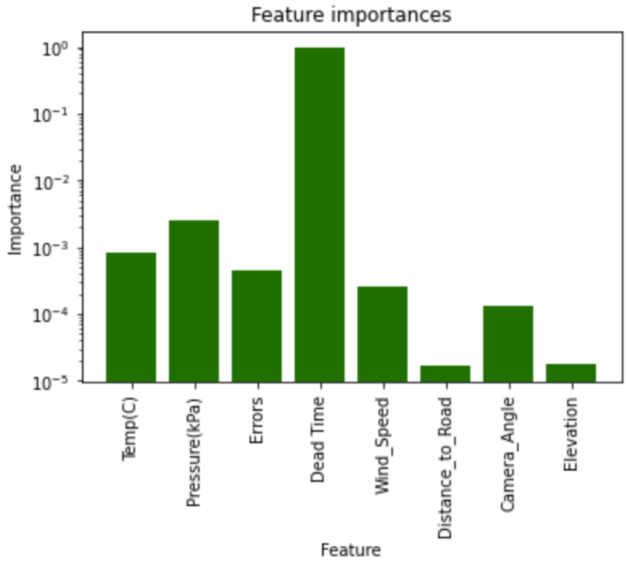
To prepare the image data for the neural network, I created a function that loaded the image from the image name, randomly flipped it vertically, randomly flipped it horizontally, and then randomly cropped the image to be 72 x 128 (from the original size of 720 x 1280). I initially tried using the entire image, but when compiling the model, my computer ran out of memory. Using a batch size of 25 and "imagenet" weights, I created a model using "applications.Xception" and added a normalization layer that normalized the image data from (0, 255) to (-1, +1). The weights of the normalization layer were the mean (0 + 255)/2 = 127.5 and variance (in this case set to be the square of the mean). My model used GlobalAveragePooling2D, had a Dropout at 0.5, and activation of "softmax." Using a learning rate of 0.00005, "optimizers.Adam," loss of mean squared error, and 10 epochs, I fit the model to my

validation dataset and regularly had mean square error values exceeding 170,000 for each of the different randomly selected validation and training datasets created in the initial setup.



Random Forest (Numeric Data)

For the random forest model, I set the eight numeric variables from the training dataset to be the independent "x" variables and the corresponding total pollution from the training dataset to be the dependent "y" variable. I also separated the validation dataset into the independent variables and dependent variable. My random forest model had 1000 "n_estimators" (trees), used mean square error as its criterion, had a maximum depth of 20, and had a minimum sample split of 2. I fit the random forest model using the training dataset and then used the model to make predictions for the validation dataset. The resulting mean square error was around 140, which was much lower than the mean square error produced by the neural networks for the numeric and image data. Looking at the features that had the most influence on the random forest model, the dead time had 100-1000-fold more impact on the predicted pollution than any of the other independent variables. This makes sense because I now know that the dead time is an instrument parameter stating how long the instrument has to "think" to measure the pollution, so longer dead times would be associated with higher pollution.



Since my random forest model performed best on my validation dataset, I used the entire original training dataset (not split into training and validation data) to create a random forest model with the same parameters. I then used this model to predict pollution values in the test dataset which resulted in a final mean square error of 124 or root mean square error of 11.2.

Weiqi’s Model

Xueao's Model

1. Preparation stage

In my literature review for this project, the article "Particle Pollution Estimation Based on Image Analysis" basically follows the following steps to make use of images to predict the air pollution concentration: ROI (region of interest) selection, feature extraction, regression model training and predicting.[2] The image feature extaction work is basically on the algorithm illustrated in this article.

The recommended features to be extracted from hazy image are transmission rate, RMS image contrast, image entropy, color, and smoothness of the sky. Since the images provided in this project don't include the sky, I cannot select the region of the sky or extract the feature of sky color and smoothness. Thus, I will only extract the transimission, RMS contrast and image entropy as the reference features.

- **Convert the images into gray scale or binary images**

The color images were converted into gray scale images, and then further into binary images with Otsu method. The Otsu method converts gray scale to binary images by selecting a threshold that minimizes the intra-class variance or maximizing the inter-class variance. The detailed coding process is shown in my Kaggle notebook. Part of the code is shown below.

```
import matplotlib.image as mpimg
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])
img = mpimg.imread(r'C:\Users\xueao\Desktop\CEE
        498DS\Project\video06182020_1271.png')
gray = rgb2gray(img)
plt.imshow(gray, cmap=plt.get_cmap('gray'), vmin=0, vmax=1)
plt.show()
```

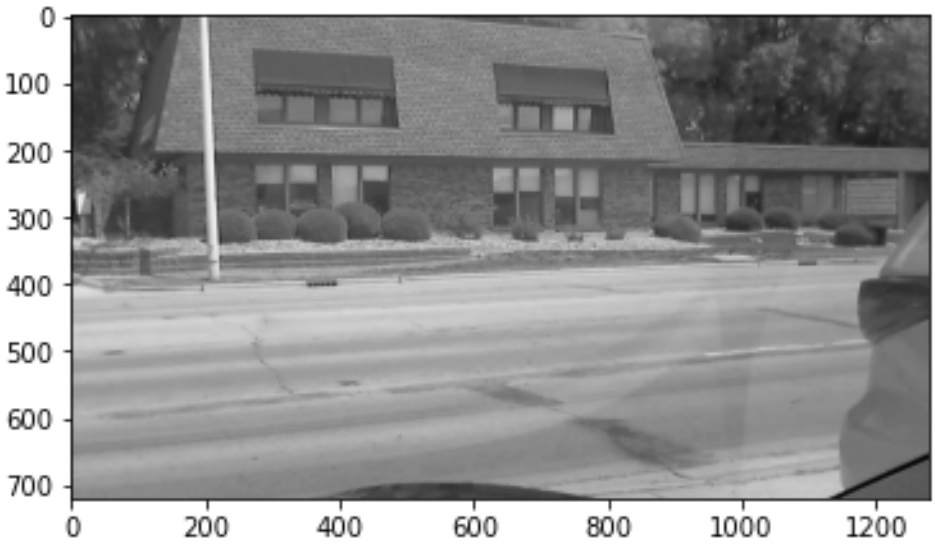


Figure 12: Ouput of Gray Scale Image

- **Feature extraction**

1. Transmission

Transmission is an important feature when we want to predict the PM concentration based on the images or videos. Liu, Tsow, Zou,& Tao (2016)"Transmission can be used to describe the attenuation of scene radiance. To solve for the transmission and thus the attenuation with a single hazy image, the concept of dark channel has been introduced, which assumes the existence of some pixels with zero or very low intensity at least for one color channel in all the outdoor images." Part of the code is shown below. For full version, please check with the Kaggle competition notebook.

```
def compute_transmission_rate(img,atmosphere_light_max,omega,dark_channel_img,

    h,w=img.shape[:2]
    img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    zero_mat=np.zeros((h,w))
    transmition_rate_est=cv2.max(zero_mat,np.ones_like(zero_mat)-omega*dark_ch

    #transmission_rate = guied_filter(img_gray,transmition_rate_est,
        guided_filter_radius, epsilon)
    transmission_rate=cv2.ximgproc.guidedFilter(img_gray.astype(np.float32),tr

    return transmission_rate
```

2. RMS contrast

Image contrast is another important feature when predicting the PM concentration. Further, according to Liu et al.(2016),"Human visual perception of air quality is related to image contrast, or visibility." The definition of RMS contrast is the standard deviation of the image pixel intensities. I will use the root mean square (RMS) of the image to represent the image contrast. Part of the code is shown below. For full version, please refer to Kaggle competition notebook.

```
pre_contrast = cv2.cvtColor(im2, cv2.COLOR_BGR2GRAY)
RMS_contrast = pre_contrast.std()
RMS_contrast
```

3. Image entropy

"Another image feature that can possibly provide PM information is image entropy, which quantifies information contained in an image, and is related to image texture."(Liu et al., 2016) To determine the image contrast and entropy, we first have to converted a color images into a gray scale image, which has been done in the step above.

4. Image and numerical data

Under certain conditions, the method of extracting features from images helps when predicting the PM concentration. But in this project, after trial, I found the correlation between image data and the air pollution concentration is not very high. The numerical data given in train and test dataset is more important for training and preditcion.

5. Training and validation data

Validation is essential when training and evaluating the models. Thus, after processing the train dataset with the similiar steps in EDA, I split the processed train data (X,y) into two parts: one part (80% of the original train dataset: X_train, y_train) for training, the other part (20% of the original train dataset: X_validation, y_validation) for validation. The detailed split can be realized by using train_test_split, and set the validation_size to be 0.2.

```
from sklearn.model_selection import train_test_split
validation_size = 0.2
seed = 3
X_train, X_validation, y_train, y_validation = train_test_split(X, y, \
        test_size=validation_size, random_state=seed)
```

In the following parts, I chose to use the numerical data to train my models: firstly evaluate all the 4 kinds of models using the scoring standard of r2 , secondly select and tune hyperparameters for the models with the best performance in the evaluation, thirdly train the selected models with the obtained best parameters, lastly follows the scoring standard of root mean squared error to select the final model.

2. Model introduction and evaluation

Here I will evaluate and compare four different models. They are Ridge regression, Lasso regression, RandomForestRegressor and GradientBoostingRegressor.

```
for estimator in estimators:
    scores = cross_val_score(estimator=estimator[1],
                             X=X_train,
                             y=y_train,
                             scoring='r2',
                             cv=3,
                             n_jobs=-1)
    #print('CV accuracy scores: %s' % scores)
    print(estimator[0], 'CV accuracy: %.3f +/- %.3f' % (np.mean(scores),
         np.std(scores)))
```

- **Ridge regression** [3]

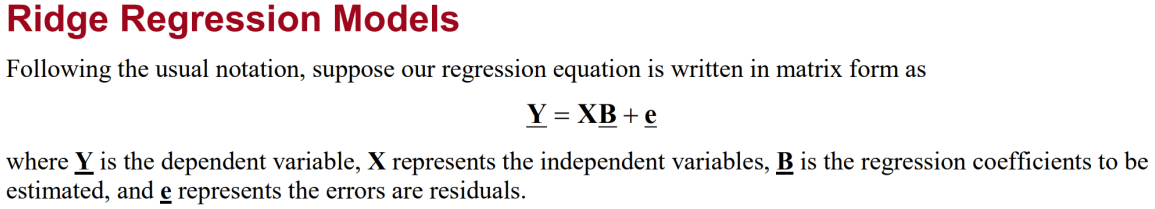


Figure 13: Ridge Regression Models

- **Lasso regression** [4]

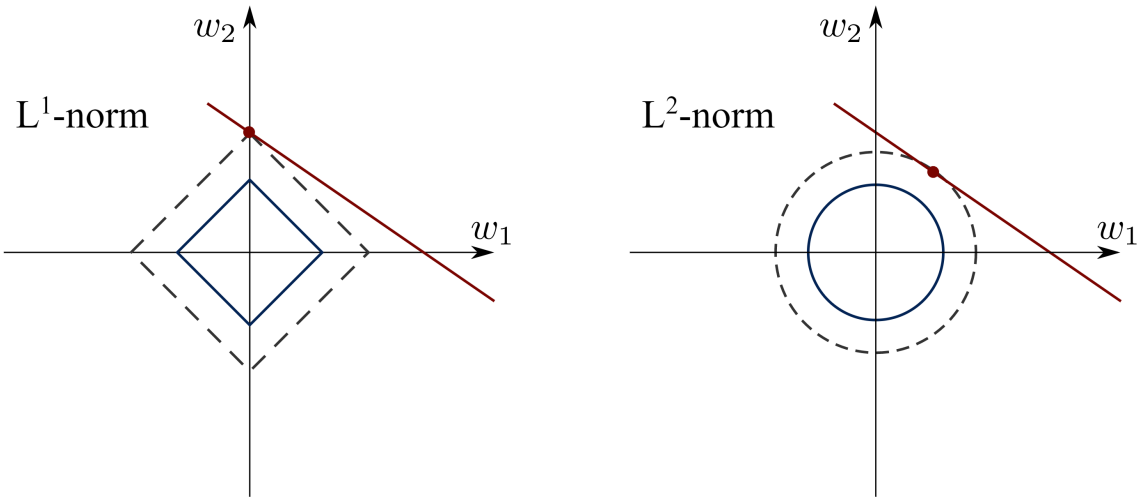


Figure 14: Forms of the Constraint Regions for Lasso and Ridge Regression

- **RandomForestRegressor** [5]

"A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. What is bagging you may ask? Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees."

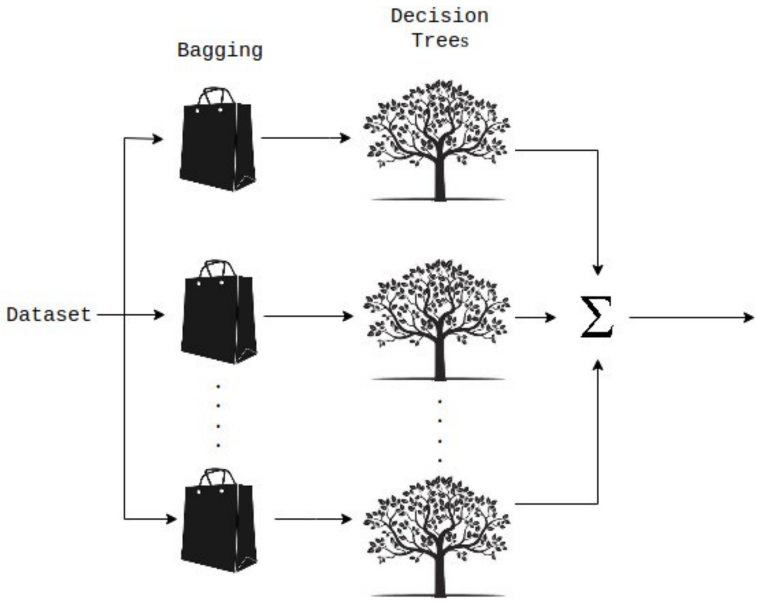


Figure 15: Random Forest Regression: Process

- **GradientBoostingRegressor** [6]

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

Figure 16: Gradient Boosting Algorithm

- **Evaluation method**: cross_val_score
- **Scoring standard**: r2

Model	CV accuracy
LassoCV	0.454 +/- 0.002
RidgeCV	0.888 +/- 0.000
RandomForest	0.997 +/- 0.001
GradientBoosting	0.997 +/- 0.001

Figure 17: Evaluation Results

3. Tune hyperparameters using GridSearchCV

The Evaluation results show that RandomForest and GradientBoosting have the best performance using the scoring standard of r2. Thus, I tune hyper parameters for the two models at the same time using GridSearchCV.

For RandomForest, I prepare 3 candidate values for the max_depth: [3, 10, 20], 7 candidate values for n_estimators: [10, 30, 50, 100, 300, 500, 1000]. Similarly, for GradientBoosting, I prepare 2 candidate values for the max_depth: [3, 10], 5 candidate values for n_estimators: [10, 50, 100, 250, 500].

Here is the coding of using GridSearchCV to tune hyperparameters for RandomForest. For more coding details, please refer to Kaggle competition notebook.

```
gs = GridSearchCV(
                estimator=RandomForestRegressor(random_state=seed),
                param_grid={'max_depth':[3, 10, 20],
                            'n_estimators':[10, 30, 50, 100, 300, 500,
        1000]},
                scoring='r2',
                cv=3,
                n_jobs=-1)
gs = gs.fit(X_train, y_train)
print('Random Forest:')
print('BR: ', gs.best_score_)
print('BR: ', gs.best_params_)
est = gs.best_estimator_
est.fit(X_train, y_train)
print('Validation accuracy: %.3f' % est.score(X_validation, y_validation))
```

Model		RandomForest	GradientBoosting
Best score		0.997	0.997
Validation accuracy		0.999	0.999
Parameters	**max_depth**	**Best = 10** out of [3, 10, 20]	**Best = 3** out of [3, 10]
	n_estimators	**Best = 300** out of [10, 30, 50, 100, 300, 500, 1000]	**Best = 500** out of [10, 50, 100, 250, 500]
	learning_rate	**not a hyperparameter here**	**Best = 0.1** out of [0.1, 0.03]

Figure 18: Best Scores and Best Parameters

As shown above, the performance of the two models with their best parameters are both excellent. Thus, I will train two models:RandomForest and GradientBoosting at the same time and evaluate their RMSE to determine the best model.

4. Model training and selection

I will use the original train dataset (X,y) to train model rather than the 80% train dataset (X_train, y_train). It is because I have already split, got, and used the 20% train dataset (X_validation, y_validation) to evaluate and compare different models in the above steps. But now if we use the whole train dataset, the model will become more trained and accurate. The two candidate models are shown as below:

- RandomForestRegressor with max_depth = 10, n_estimators = 300
- GradientBoosting with learning_rate = 0.1, max_depth = 3, n_estimators = 500

Finally, I selected RandomForest as my model because after trial I find this model can lead to a reasonable RMSE value while GradientBoosting will lead to an unacceptably huge root mean squared error.

```
model = RandomForestRegressor(max_depth = 10, n_estimators =
        300,random_state = 3)
model.fit(X,y)
Pred = model.predict(X_test)
sample['Total'] = Pred
```

5. Summary for modeling process

The flowchart below shows my overall modeling process. Finally, the model and parameters I selected is: RandomForestRegressor with max_depth = 10, n_estimators = 300. I first apply the entire processed train dataset to train the model, then predict for the test dataset, and finally get the root mean squared error of 10.74. RMSE = 10.74 (MSE = 115) is within the reasonable range for this project.

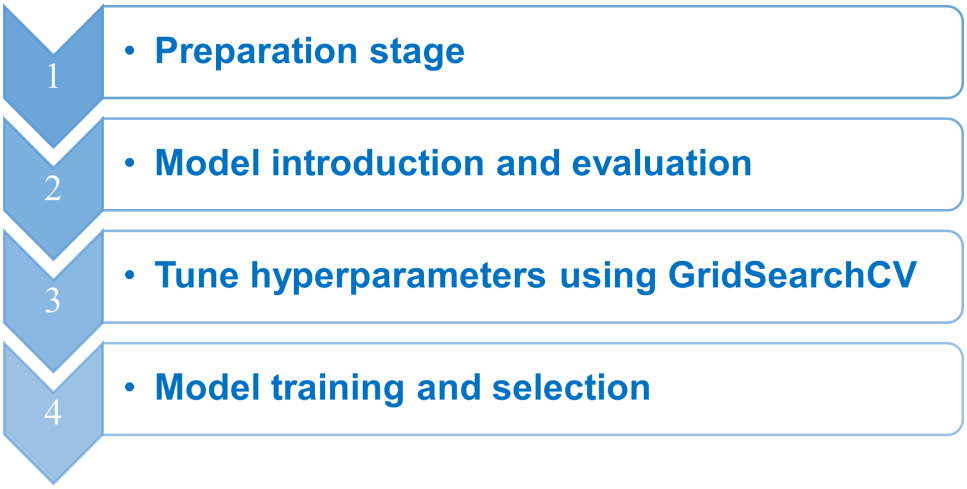


Figure 19: Flowchart of Modeling

Conclusion

All of the team members found random forest models to produce the best results with the lowest root mean square error. While each member used different parameters for her model, the final predictions had root mean square error values of less than 20. Based on our results, we conclude that machine learning can be used to approximate particulate matter with the variables we had available, but a better model will be needed to produce more accurate predictions.

References

1. **Predicting outdoor ultrafine particle number concentrations, particle size, and noise using street-level images and audio data**
   Kris Y. Hong, Pedro O. Pinheiro, Scott Weichenthal
   Environment International (2020-11) https://doi.org/ghnh6n
   DOI: 10.1016/j.envint.2020.106044 · PMID: 32805577

2. **Particle Pollution Estimation Based on Image Analysis**
   Chenbin Liu, Francis Tsow, Yi Zou, Nongjian Tao
   PLOS ONE (2016-02-01) https://doi.org/ghnjkc
   DOI: 10.1371/journal.pone.0145955 · PMID: 26828757 · PMCID: PMC4734658

3. **Ridge Regression**
   NCSS, LLC
   (2020-02-10) https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

4. **Lasso**
   Wikipedia
   (2020-11-17) https://en.wikipedia.org/w/index.php?title=Lasso&oldid=989142673

5. **A Beginners Guide to Random Forest Regression**
   Krishni
   Medium (2019-06-05) https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb

6. **Gradient boosting**
   Wikipedia
   (2020-12-05) https://en.wikipedia.org/w/index.php?title=Gradient_boosting&oldid=992489307