Udacity Machine Learning Nanodegree Capstone Report

Predicting Smog Pollution Rates in Beijing

By: Randall Tom

Project Overview

My project is loosely similar in that we are trying to understand if several features from historical data can help predict pollution smog level in one of the heaviest polluted cities in the world, Beijing, China [1]. We want to be able to predict smog levels because it will help people prepare for future days in terms of smog pollution defense. For instance, people would like to know the smog level prediction for a few days ahead of time in order to decide if they can safely hold a birthday party outdoors. Or, they may want to know if this weekend is a good time to go shopping without wearing a pollution mask. Having a prediction would be very helpful, similar to a weather prediction. Beijing, China has one of the higher rates of air pollution in the world. As this is a reality of modern life, the pollution level for future days is very useful for numerous reasons, such as forecasting days that are not safe to go outside as a senior citizen or young child or finding the right time to go out without needing to wear a protective mask. The ability to accurately predict the city's pollution has become a thriving business for some tech companies [2]. The input data for this project will come from the University of California at Irvine's Machine Learning Repository [3].

Problem Statement

Supervised learning algorithms assist us in using sets of examples to make predictions. Regression is a specific type of supervised learning algorithms where a value is being predicted. This project is a regression problem, and possibly linear regression problem, as we are trying to use the numerical data to predict another numerical value. In this

capstone project, I will specifically try to predict PM2.5 levels in Beijing based on several features. PM 2.5 is the level of airborne particulate matter that can infiltrate into the lungs.

Metrics

For metrics, we will be using R^2 to evaluate the models. R^2 is a metric that provides an indication of goodness of fit for a set of predictions to the actual values. This measure is called the coefficient of determination and the values are between 0 and 1 for horrible fit to perfect fit. [6] The closer it is to 1, the better. The closer to 0, the worse. This is the optimal choice because we are able to compare it against test data but we are not able to use it for predictions as we don't have any real data outside of the dataset. Comparing the results from predictions to test data makes the most sense here and R^2 is a great way to compare the models.

Data Exploration

I will be using a UC Irvine Dataset called **Beijing PM2.5 Data Set** [3], which contains 1 .csv file necessary to make a prediction. For this dataset, this a multivariate, time-series data with 13 attributes. One of the attributes is actually a row number. Here is a list of the attributes as indicated on the UCI description [3]:

No: row number
year: year of data in this row
month: month of data in this row
day: day of data in this row
hour: hour of data in this row
pm2.5: PM2.5 concentration (ug/m^3)
DEWP: Dew Point (â„f)
TEMP: Temperature (â„f)
PRES: Pressure (hPa)

cbwd: Combined wind direction

Iws: Cumulated wind speed (m/s)

Is: Cumulated hours of snow

Ir: Cumulated hours of rain

Here is a sample of three data points / lines inside this CSV file:

No	year	month	day	hour	pm2.5	DEWP	TEMP	PRES	cbwd	Iws	Is	Ir
25	2010	1	2	0	129	-16	-4	1020	SE	1.79	0	0
26	2010	1	2	1	148	-15	-4	1020	SE	2.68	0	0
27	2010	1	2	2	159	-11	-5	1021	SE	3.57	0	0

I

This data set has 43824 data points. The distribution has a max of 994 and a minimum of 0. Median is 72 and mean is 98.613. Standard deviation is 92.05. Mode is 16.

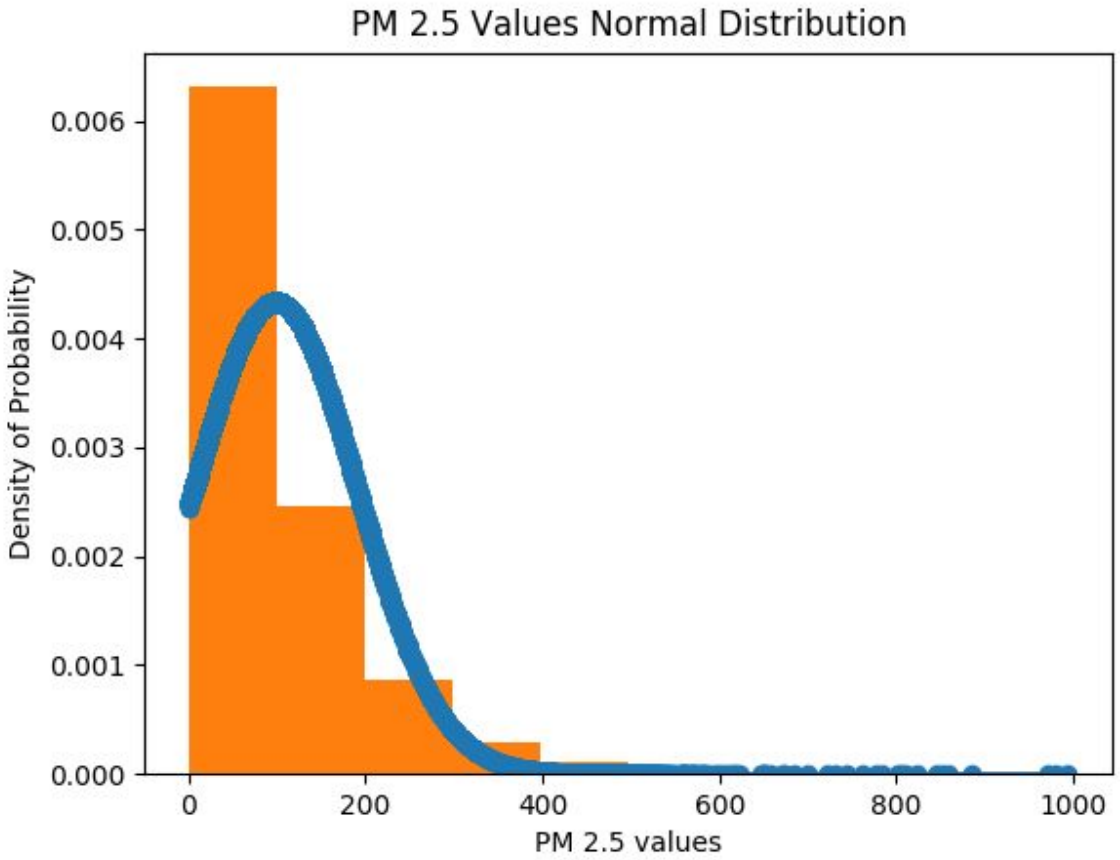
In terms of preprocessing, I will eliminate the points that are missing any of the attributes.

Exploratory Visualization

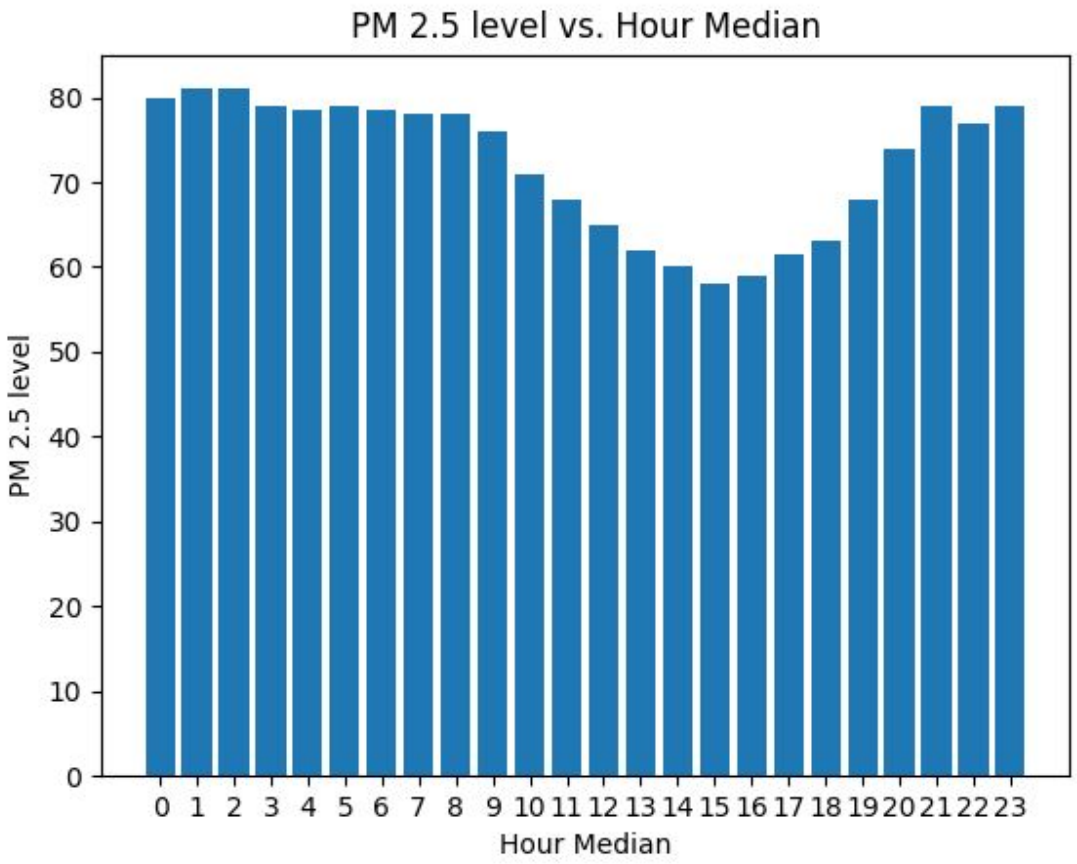
```
              year         month           day          hour         pm2.5  \
count  41757.000000  41757.000000  41757.000000  41757.000000  41757.000000
mean    2012.042771      6.513758     15.685514     11.502311     98.613215
std        1.415311      3.454199      8.785539      6.924848     92.050387
min     2010.000000      1.000000      1.000000      0.000000      0.000000
25%     2011.000000      4.000000      8.000000      5.000000     29.000000
50%     2012.000000      7.000000     16.000000     12.000000     72.000000
75%     2013.000000     10.000000     23.000000     18.000000    137.000000
max     2014.000000     12.000000     31.000000     23.000000    994.000000

              DEWP         TEMP         PRES         cbwd          Iws  \
```

count	41757.000000	41757.000000	41757.000000	41757.000000	41757.000000
mean	1.750174	12.401561	1016.442896	1.946692	23.866747
std	14.433658	12.175215	10.300733	1.604021	49.617495
min	-40.000000	-19.000000	991.000000	0.000000	0.450000
25%	-10.000000	2.000000	1008.000000	1.000000	1.790000
50%	2.000000	14.000000	1016.000000	1.000000	5.370000
75%	15.000000	23.000000	1025.000000	4.000000	21.910000
max	28.000000	42.000000	1046.000000	4.000000	565.490000

	ls	lr
count	41757.000000	41757.000000
mean	0.055344	0.194866
std	0.778875	1.418165
min	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
max	27.000000	36.000000



From the plot above, we can see the density of probability of PM 2.5 values is higher at the lower ends at around 0 - 100. The normal curve can be considered shifted towards the left.



From the plot above, we can see that in terms of median level per hour, the 10th hour to the 20th hour is a low dipping curve. This indicates that higher medians of PM 2.5 levels are seen from 9PM to 9AM, which means that PM 2.5 levels are actually worse during rest and non-standard working hours.

Algorithms and Technique

We would like to understand the relationship between 11 features and the PM2.5 level. I will choose a split of about 20% of the data for testing, and the rest for training. The main purpose of splitting the data is to allow for cross-validation. Similar to what was taught during the Boston Housing project, I will begin by using a Decision Tree algorithm to create a regressor first and then attempt to fit that data with Grid Search to tune the parameters. Also, we can try and look at PCA, linear regression, Random

Forest, or deep learning to check if predictions are better. Below are definitions of each algorithm.

Linear Regression: “Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable *x* is associated with a value of the dependent variable *y*.” [10]

Decision Tree: “Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data (but the resulting classification tree can be an input for decision making). This page deals with decision trees in data mining.” [11]

PCA: “Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components (or sometimes, principal modes of variation). The number of principal components is less than or equal to the smaller of the number of original variables or the number of observations. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the

highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables." [12]

Random Forest: "Random forests or random decision forests[1][2] are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set." [13]

Deep Learning / Keras: "Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, partially supervised or unsupervised.

Some representations are loosely based on interpretation of information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain.[5] Research attempts to create efficient systems to learn these representations from large-scale, unlabeled data sets." [14]

Benchmark Model

For this project, I will be using simple guessing as a benchmark to prove that the model has at least learned something from the data. Sklearn has a Dummy class for this that will help implement the simple guessing. If my model does not beat simple guessing, then it really did not learn much from the data.

http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html

Data Preprocessing

In terms of preprocessing, we will remove all data (any row) that is missing in any of the features. Also, we will convert wind direction to the following values:

cv - > 0

NW -> 1

NE -> 2

SW -> 3

SE -> 4

Implementation and Refinement

Data can be separated to 80% training and 20% for testing. Because data from the same period of time can be similar, we will be using time series split to separate the data. Also, this will help to avoid the problem also known as autocorrelation.

1) We will begin by using regular linear regression. Then, we will cross validate on the data set to test.

Estimated intercept coefficent: -31.0765857406

Number of coefficients: 11

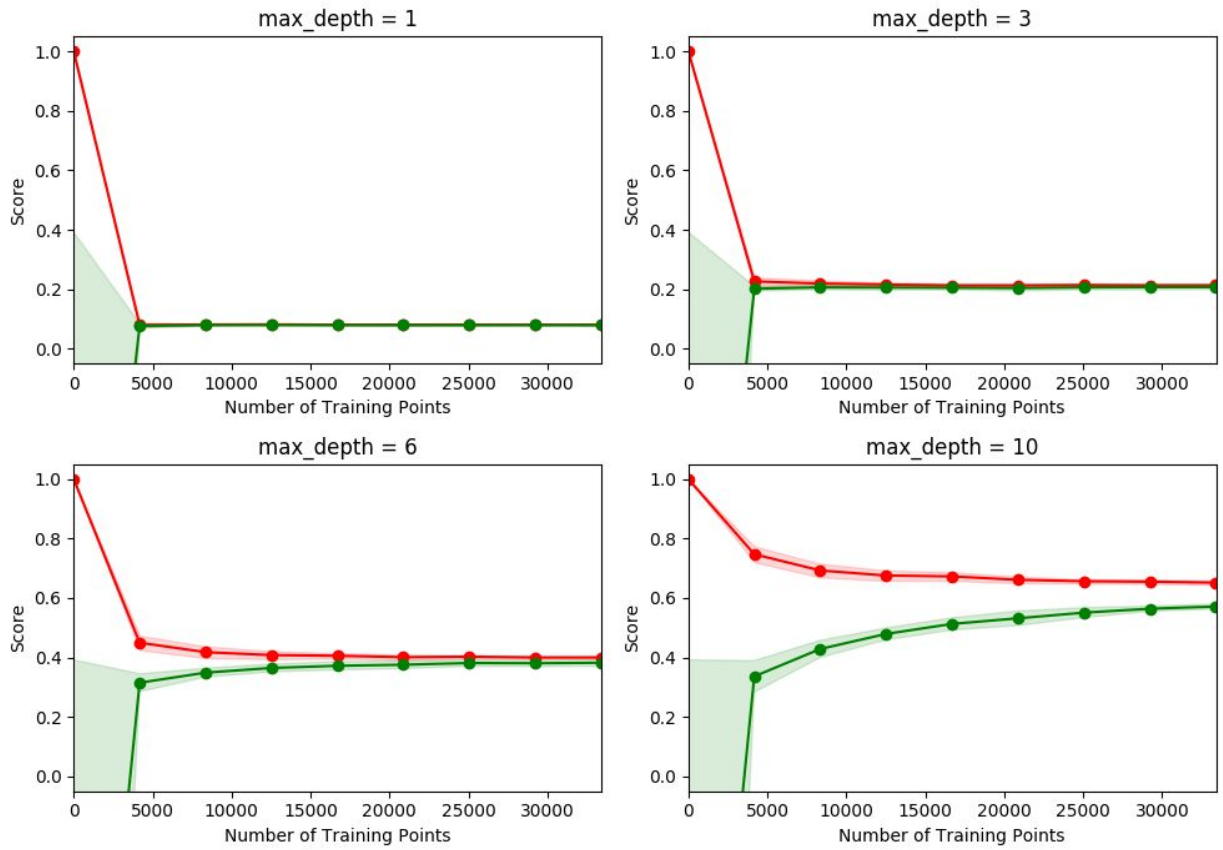
Variance score: 0.26

In terms of the code, here is what I did for this model. I performed a split for training and testing data with a 80/20 split. Then, I initialized the LinearRegression model, created the predictions given the test data, and called the r2_score function to get the R^2 regression score between predictions and test data.

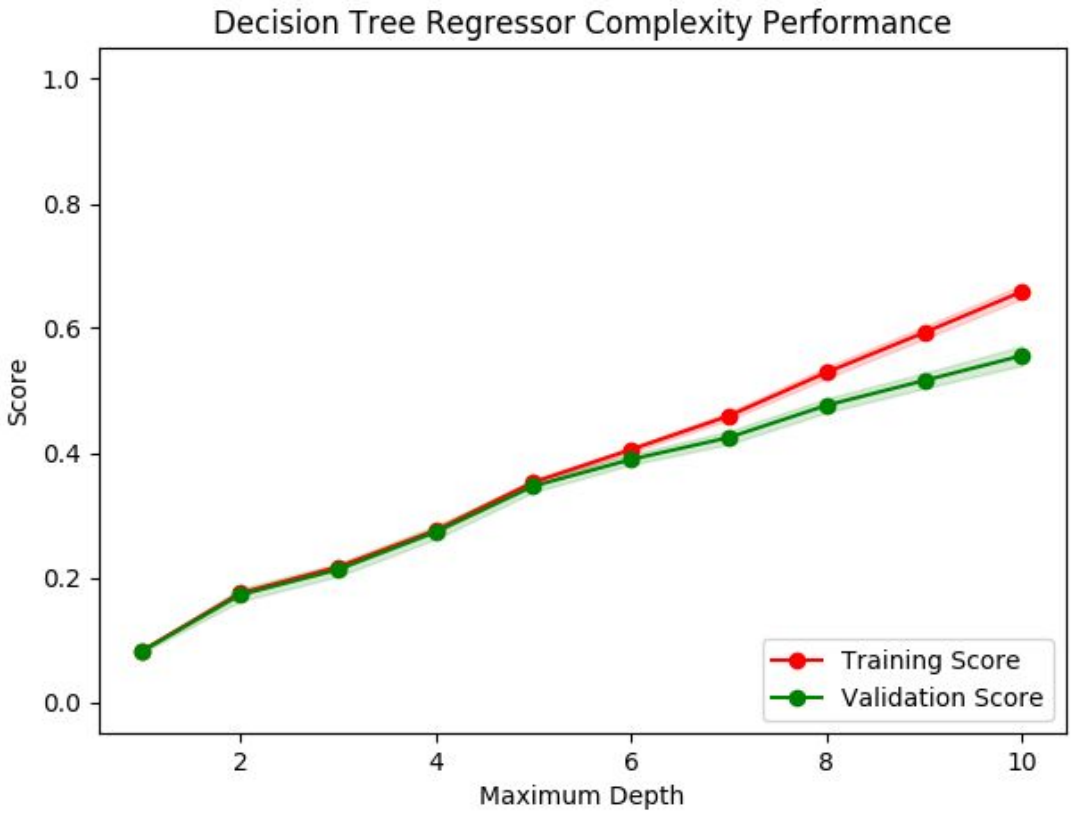
```
X_train, X_test, y_train, y_test = train_test_split(features, pm25, test_size=0.2, random_state=42)
```

```
def fit_model(X, y, X_test, y_test):
    """ Performs grid search over the 'max_depth' parameter for a
        decision tree regressor trained on the input data [X, y]. """

    regressor = LinearRegression()

    # Fit the grid search object to the data to compute the optimal model
    regressor.fit(X, y)

    y_pred = regressor.predict(X_test)

    score = metrics.r2_score(y_test, y_pred)

    # Return the optimal model after fitting the data
    return (regressor.intercept_, regressor.coef_, score)

# Fit the training data to the model using grid search
reg = fit_model(X_train, y_train, X_test, y_test)
```

2) We will use Decision Tree algorithm with Grid Search. Here is a chart of max-depth differences. From the chart below, we can see that max_depth of 10 is the best since both curves converge around score of 0.6.



For the decision tree regressor complexity performance chart below, we can see that the training score and testing score are about the same. The highest point is at maximum_depth of 10, where both scores are around 0.6.



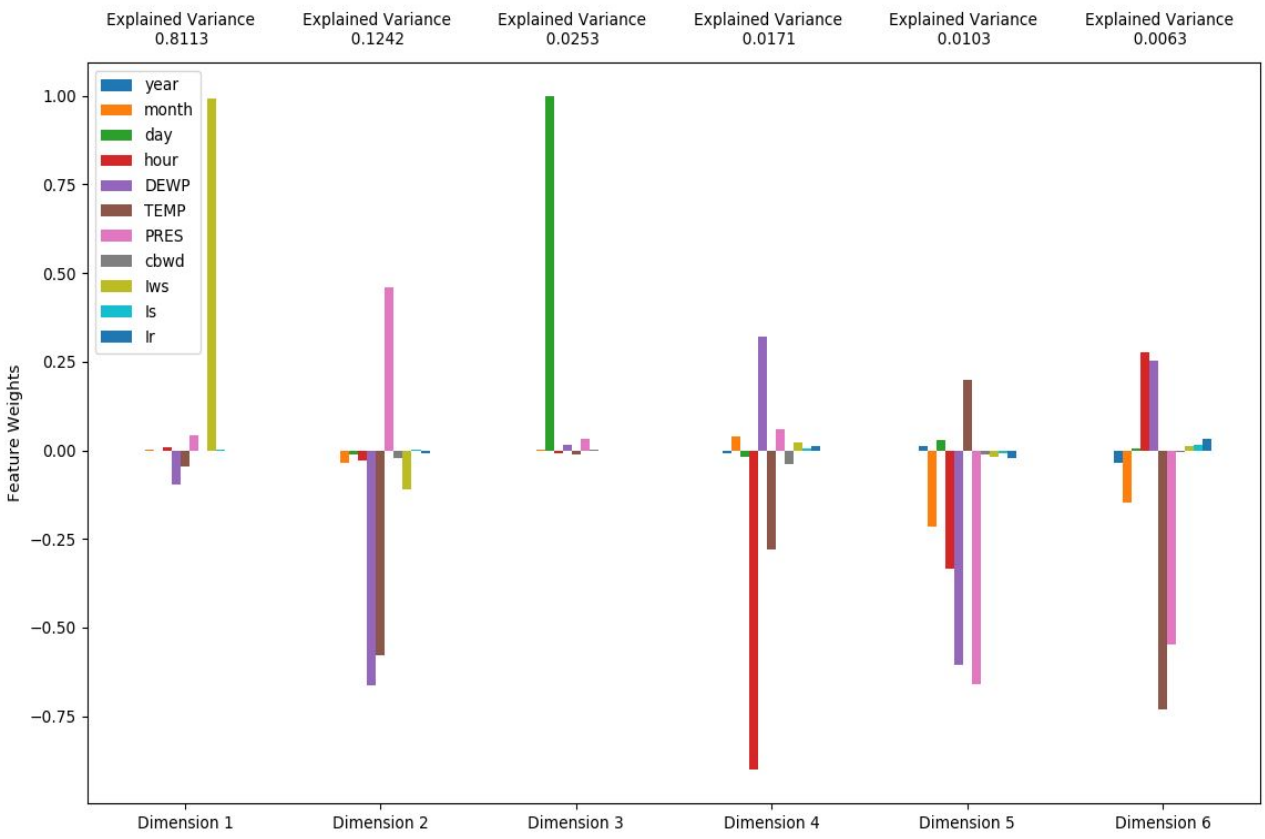
The regression score with the test data is: 0.55450033722, which is really high compared to the other models.

In terms of the code, here is what I did for this model. I performed a split for training and testing data with a 80/20 split. Since this will require Grid Search, I have to separately perform a shuffle split of the data to 80/20 again. Grid Search also requires max_depth parameters (we used from 1 to 10) and a scoring function to check the R^2 score. Then, I initialized the DecisionTreeRegressor model, fit the model to return the best estimator (the best max_depth level from the ones we provided). Afterwards, I get the KFold cross validation score at 10 folds. Finally, we created the predictions given the test data, and called the r2_score function to get the R^2 regression score between predictions and test data.

```
def performance_metric(y_true, y_predict):
    score = metrics.r2_score(y_true, y_predict)
    return score


X_train, X_test, y_train, y_test = train_test_split(features, pm25, test_size=0.2,
random_state=42)
X_full_train, _, y_full_train, _ = train_test_split(features, pm25, test_size=0,
random_state=42)

def fit_model(X, y):
    cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, random_state = 0)

    regressor = DecisionTreeRegressor()

    params = {'max_depth':[1,2,3,4,5,6,7,8,9,10]}
    scoring_fnc = make_scorer(performance_metric)
    grid = GridSearchCV(regressor, params, scoring=scoring_fnc, cv=cv_sets)

    grid = grid.fit(X, y)

    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(regressor, X_full_train, y_full_train, cv=10)
    print "KFold cross validation mean = ", scores

    return grid.best_estimator_

reg = fit_model(X_train, y_train)
score = reg.score(X_test, y_test)
```

3) We will then use PCA. Looking at the training data and transforming it to PCA if n_components = 6, we can see that some features have a closer relationship to other specific features. For example, we see that dew point and temperature have a similar variance in the second graph. Also, pressure and dew point also have a strong relationship in the 5th graph. One feature that does not correlate will with others is day and hour, as shown in the 3rd and 4th

graphs respectively. Also, cumulated wind speed in the 1st graph shows very little relationship with the other features.



In terms of the code, here is what I did for this model. Since this will require Grid Search, I have to separately perform a shuffle split of the data to 80/2. Also, I have to declare Logistic Regression. Grid Search also requires declaring PCA with the number of components (I chose 6) to create the fit and declared a pipeline for logistic regression. Then, I performed Grid Search and did a fit of the model. Finally, I made a plot of the best estimator appear on the screen.

```
def fit_model(X, y):

    cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, random_state = 0)

    logistic = linear_model.LogisticRegression()

    pca = decomposition.PCA(n_components=6).fit(X)

    pipe = Pipeline(steps=[('pca', pca), ('logistic', logistic)])

    n_components = 6

    estimator = GridSearchCV(pipe, dict())
    estimator.fit(X, y)

    plt.axvline(estimator.best_estimator_.named_steps['pca'].n_components,
            linestyle=':', label='n_components chosen')
    plt.legend(prop=dict(size=12))
    plt.show()
```

4) We will then try Random Forest. Using a RandomForestClassifier and the training data, the model that we created leads to a prediction scores:

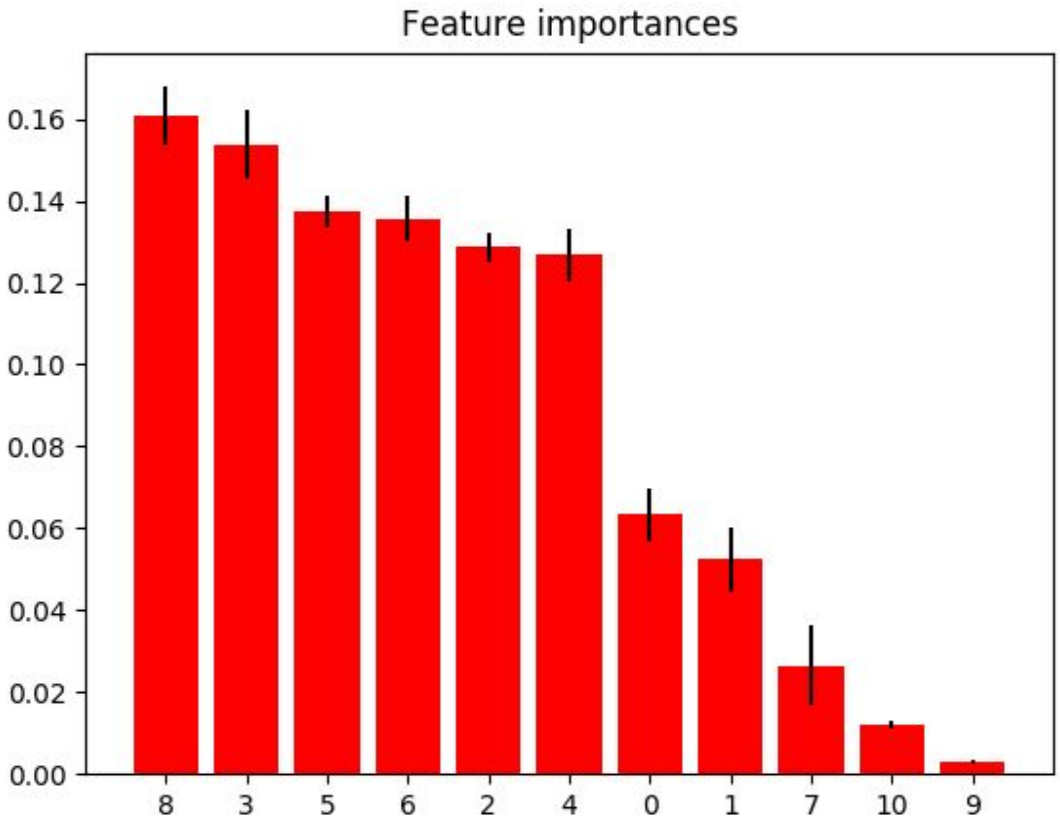
N_estimators = 10, Prediction score = 0.034003831417

N_estimators = 50, Prediction score = 0.0362787356322

N_estimators = 100, Prediction score = 0.0390325670498

We can see that higher N_estimators will lead to a better prediction score. However, my computer crashed several times as I tried to use a higher N_estimators value, so this the highest I can go.

Here is a diagram of feature importance with Random Forest of trees with N_estimators = 100. From the graph below, we can see that cumulated wind speed and hour of the day are the best informative features of the forest. The remaining features, temperature, pressure, day and dew point are also informative, but not as informative as wind speed and hour. The rest of the features are not very informative at all.



Feature Number to Feature chart:

Feature number	Feature	Feature Ranking
0	year: year of data in this row	(0.063300)
1	month: month of data in this row	(0.052394)
2	day: day of data in this row	(0.128633)

3	hour: hour of data in this row	(0.153788)
4	DEWP: Dew Point (â„ƒ)	(0.126792)
5	TEMP: Temperature (â„ƒ)	(0.137310)
6	PRES: Pressure (hPa)	(0.135616)
7	cbwd: Combined wind direction	(0.026554)
8	Iws: Cumulated wind speed (m/s)	(0.160707)
9	Is: Cumulated hours of snow	(0.002834)
10	Ir: Cumulated hours of rain	(0.012072)

In terms of the code, here is what I did for this model. I performed a split for training and testing data with a 80/20 split. Then, I initialized the RandomForestClassifier model with 100 n_estimators. I fit the model and performed predictions. Finally, I got the R^2 score for comparing prediction and test data.

For the feature ranking, I took the standard deviation of the tree's feature importances as y-indices and feature importances as x-indices to create the feature importances graph.

```
def fit_model(X, y):

    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X, y)
```

```
    return clf

trained_model = fit_model(X_train, y_train)

predictions = trained_model.predict(X_test)

score = trained_model.score(X_test, y_test)

print "Prediction score is: ", score

importances = trained_model.feature_importances_
std = np.std([tree.feature_importances_ for tree in trained_model.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

print("Feature ranking:")

for f in range(X_train.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train.shape[1]), indices)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```

5) We will finally try deep learning with Keras. The prediction score for the testing data was pretty low at 0.011973%. First, I had to convert the Pandas Data Frame into a Numpy Array. Then, despite attempting to tune the variables, with alternating the Dense layer units from the 1st one from 150 -> 12, 2nd one from 8 -> 50, and the 3rd one from 1 - >5, the result was still a prediction score of 0.011973%. Changing the nb_epoch of the fit and batch_size in the model evaluation did little to change the percentage either. From the results with Keras, we can conclude that the model created does not predict the PM 2.5 levels very well at all.

In terms of the code, here is what I did for this model. I performed a split for training and testing data with a 80/20 split. Then, I created a Sequential model along with 3 Dense layer units as described above. Finally, I compiled the model for 'binary_crossentropy' and fitted the model. Finally, I evaluated the X and y test values with the model's prediction and ended up with the prediction score of 0.011973%.

```
X_train, X_test, y_train, y_test = train_test_split(features, pm25, test_size=0.2,
random_state=42)

model = Sequential()

model.add(Dense(150, input_dim=11, init='uniform', activation='relu')) #<- 12 is0.011973, 150
is 0.11973
model.add(Dense(50, init='uniform', activation='relu')) #<- 8 is 0.11973, 50 is also 0.011973
model.add(Dense(1, init='uniform', activation='sigmoid'))

model.compile(loss='binary_crossentropy' , optimizer='adam', metrics=['accuracy'])

model.fit(X_train.values, y_train.values, nb_epoch=50, batch_size=len(data)) #<-150 vs 50 no
difference

loss_and_metrics = model.evaluate(X_test.values, y_test.values, batch_size=100) #<- adjust
doesnt make change

print("\n%s: %f%%" % (model.metrics_names[1], loss_and_metrics[1]*100))
```

In terms of challenges, I had great difficulty in the beginning trying to convert a numpy array to a Pandas Dataframe and manipulating the data to get rid of certain columns and rows. It took quite a lot of review of documentation and using StackOverflow online to figure out the correct way to do things. I also ended up using a lot of the code samples from the earlier projects to help structure my code. Otherwise, almost everything else was not that challenging once the data manipulation was complete because most of the modeling libraries used simple require plugging in the data and setting / experimenting with the parameters and options.

Tools and Libraries used:

1) Pandas
2) Numpy

3) Python
4) Jupyter
5) Scikit learn
6) Matplotlib
7) Keras and Tensorflow

Results

Looking at the benchmark model, which is the dummy random guessing classifier from SKLearn library, we get a regression score, or R^2 score as discussed previously, of 0.00634578544061. Here is a summary of the regression scores of the models used:

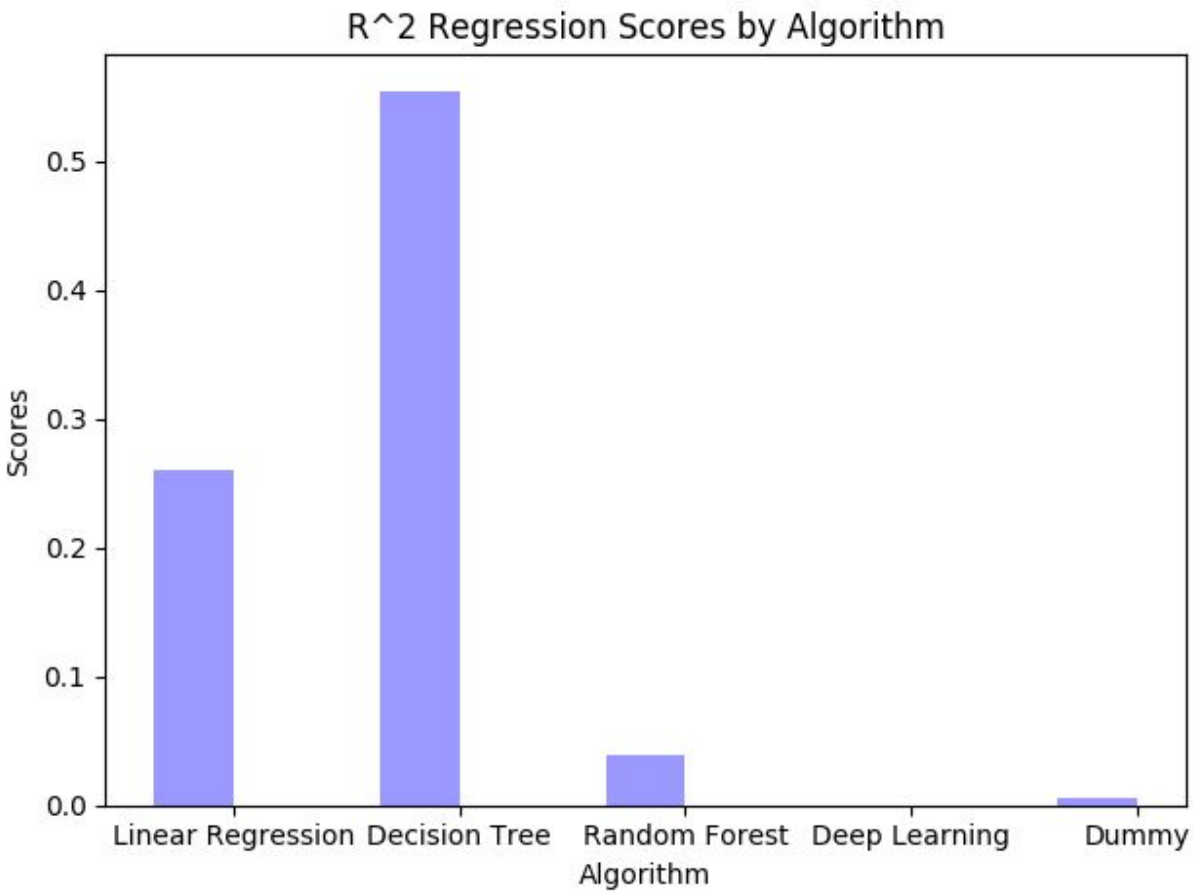
Model	Regression Score
Dummy	0.00634578544061
Linear Regression	0.26
Random Forest	0.0390325670498
Keras	0.00011973
Decision Tree	0.55450033722

Based on this information, I would use Decision Tree model to predict PM 2.5 levels. From a bigger picture, this model also makes sense in terms of deciding which variables / features to use and focus on. Some of the features that are not important to the PM 2.5 level would not be chosen for calculation. While this may not completely solve the problem of predicting PM 2.5 levels in Beijing, it is a good starting point to look at some of the bad and better machine learning models that can be utilized for prediction. If used by the public in Beijing, it can give values with a 0.554 regression score, which is much much better than the random dummy solution. Also, we confirmed through KFold cross-validation of the Decision Tree model that the mean KFold score is: 0.553955721965 at 10 folds, 0.552207365339 at 15 folds, and 0.551084951352 at 5 folds. This is very similar to the number obtained from the GridSearch model, which shows that coherence is demonstrated in the results.

In terms of robustness, it generates much better to unseen data compared to the other models we tested, as shown in the predictions of test data. Decision trees are also not sensitive to outliers since splitting occurs based on samples proportion within the split ranges but not on the absolute values. [15]

We can trust this model, but only with a grain of salt. From looking at the 0.554 regression score, we know through the testing of predictions with test data that it is not always correct, but it can be used as one of the tools to predict smog PM 2.5 level along with other tools and definitely not the sole tool. Perhaps other machine learning models can be used in conjunction, such as linear regression, or others that were not tested for this paper.

Conclusion



We started this project with a data set from UCI, preprocessed by deleting the data points with missing values, and used a variety of models such as Random Forest, Linear Regression, Decision Tree and Keras to attempt prediction of the PM 2.5 levels in Beijing. We also used PCA in graphs to determine which features were more important than others. From these tests and the graph above, we determined Decision Tree provided the best regression score.

I found it interesting that Keras did worse than random dummy classifier. Perhaps the neural network was no better at learning the results since more computing power and training time would be needed to get better. Also, another reason could be that there is not enough data for the neural network to be optimal since we are only looking at approximately 40,000 data points.

One way that this project can be improved is to evaluate more current data for Beijing. Also, we can add a few more cities in Northeastern China to compare if the Decision Tree model is as good over there as it is in this project in Beijing.

References

1) Science Magazine. 2017.
http://www.sciencemag.org/news/2017/03/here-are-some-world-s-worst-cities-air-quality

2) Reuters. 2015.
https://www.reuters.com/article/us-china-pollution/tech-giants-spot-opportunity-in-forecasting-chinas-smog-idUSKBN0UB1KB20151229

3) UCI Data Science Data Source. 2017.
https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data

4) Liang, X., Zou, T., Guo, B., Li, S., Zhang, H., Zhang, S., Huang, H. and Chen, S. X. (2015). Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC and winter heating. Proceedings of the Royal Society A, 471, 20150257.
http://www.mdpi.com/2073-4433/6/8/1243/pdf

5) Present live Beijing PM 2.5 data.
http://aqicn.org/city/beijing/

6) Machine Learning Metrics
https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

7) Feature importance for forest trees
http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

8) Develop Your First Neural Network in Python With Keras Step-By-Step
https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

9) Udacity Machine Learning Program
https://github.com/udacity/machine-learning

10) Multiple Linear Regression
http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm

11) Decision Tree
https://en.wikipedia.org/wiki/Decision_tree_learning

12) PCA:
https://en.wikipedia.org/wiki/Principal_component_analysis

13) Random Forest
https://en.wikipedia.org/wiki/Random_forest

14) Deep Learning
https://en.wikipedia.org/wiki/Deep_learning

15) Decision Tree
http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics