Udacity

Machine Learning Engineer Nanodegree

Capstone Project Report

Brindha Sivashanmugam

August 14th 2018

EMERGENCY – 911 CALLS PREDICTION

DEFINITION

Project Overview

We all know how significant the role of a 911 employee is. They are the ones who address our emergencies. The team works 24x7 non-stop. Any challenges they face in their job will directly affect the public in their life or death moment. Let us look at what happens when there is a flaw on their side, how it affects the public, and the challenges they face in their day-to-day job.

From the Headlines: (How a flaw in 911 affects the Public)

News 1 - Longer Response Time: Tragic death of Shanell Anderson in Atlanta because emergency responders arrived too late to save Shanell Anderson from drowning in her SUV.[1] **News 2 - *Equipment Outage*:** A cooling unit low on refrigerant, caused Montgomery County, MD 's 911 system to crash for nearly two hours on the night of July 10 2016, causing about 100 callers to get busy signals when they tried to report emergencies. Delays caused deaths of a 40-year-old dialysis patient and a 91-year-old woman. [2] ***News 3 - Human error:*** The death of Kokomo resident Tammy Ford on July 1 2015, due to operator error. In that case, a dispatcher sent first responders to the wrong address. [3]

Challenges Faced by 911 Operators and Dispatchers:

- Most of the 911 employees work in 12 hour shifts in a hectic, high stress environment, not able to plan their leisure hours or vacation. They are humans too. If they could know how to plan this, they could be stress free, and operator error will no longer be an issue.
- Not able to know when they could schedule a downtime to maintain their equipment, which is very crucial, so that the equipment would to be in good shape, ready for a dire emergency.
- Not able to predict the prime location where most of the calls are expected on a particular day and time. If they had known that, they could rotate employees accordingly and cover the emergencies in a shorter response time.

Research in such area has already been started, and the academic paper "Mining 911 Calls in New York City: Temporal Patterns, Detection and Forecasting" is an example. [4]

I have used "**Emergency - 911 calls**" dataset[5] for this problem. This dataset was provided by montcoalert.org, and donated to Kaggle by Mike Chirico.

1: https://www.ravemobilesafety.com/blog/911-system-failures

2: http://www.kokomotribune.com/news/dispatchers-discuss-challenges-of-the-job/article_85753cb0-379a-11e5-8a06-b74ef6ca24dc.html

3: https://www.washingtonpost.com/local/md-politics/montgomery-officials-try-to-explain-911-outage-that-should-never-happen/2016/08/02/798c77d8-58d2-11e6-9767-f6c947fd0cb8_story.html?utm_term=.cb08532efcaf

4 https://warwick.ac.uk/fac/sci/dcs/people/theo_damoulas/pubs/aaai15_911damoulas.pdf

5: https://www.kaggle.com/mchirico/montcoalert

Problem Statement

By minimizing the response time of the 911 team, lot of lives could be saved. Response time can be minimized if there are surplus amount of equipment and human resources ready to serve emergency, which is practically not possible, as it would consume a lot of money.

So, the alternate way is to effectively distribute available resources. This could be possible only when they know in advance about how many 911 calls are expected. As this is emergency call, no one knows what is expected until it actually happens. Here is where Machine Learning comes in rescue of the rescue team.

The goal is to create a regressor model which performs the following:

- Predict the number of emergency calls expected in emergency category – EMS
- Predict the number of emergency calls expected in emergency category – Fire
- Predict the number of emergency calls expected in emergency category - Traffic

The problem will be solved by following the below steps:

- Outlier and error record removal
- Forming grid based on latitude, longitude and partitioning the County into regions
- Extracting day, week, day of week, month, year and time information from timestamp
- Categorizing time by converting into 3 time-windows
- Scaling (numeric features) and one hot encoding (non-numeric features)
- Broadly classify “title” column into 3 categories – EMS, Fire and Traffic
- Rearranging the dataset to find the number of emergencies per category
- Split the dataset into training and testing sets in order as this is time-series data
- Implement benchmark model – Linear regression
- Implement better model, “The Solution Model”– XGBoost Regressor
- Fine tune various hyperparameters.

Metrics

R2 score is used for evaluating all the models developed in this project. R2 score is a very common regression metric. It is based on comparing our model to the simplest possible model (the mean of the data points).

The simplest possible model is the average of all data points, represented as a horizontal line that goes through them. The sum of squares of residuals of this simple model is calculated (SSmean). And, the sum of squares of residuals of our model, for example, linear regression model is calculated (SSlinear_reg). Then, R2 score would be defined as:

$$\text{R2 score} = 1 - (SS_{linear\_reg} / SS_{mean})$$

This will be clearly given in the below plot.

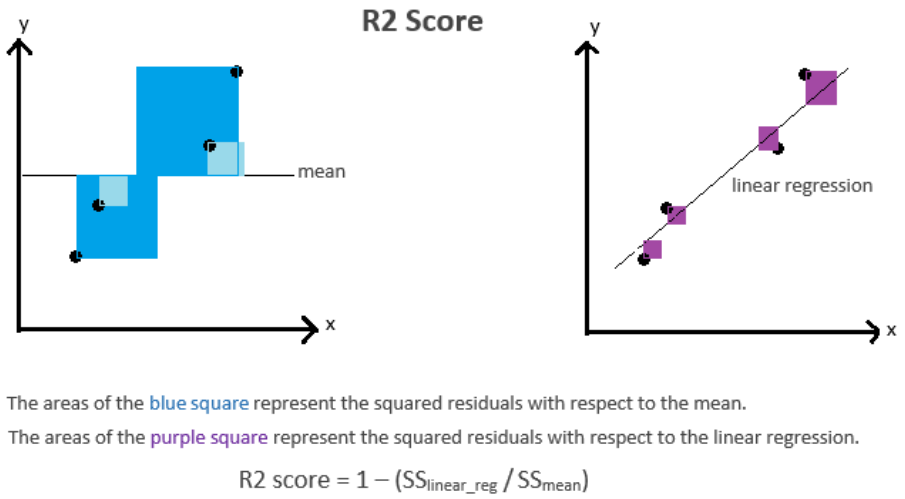


Fig. 1 *- R2 Score*

The value of R2 score lies between 0 and 1. In order for a model to be good, the R2 score should be closer to 1.

I have chosen R2 score because[6],

- R2 score says how much variability of the dependent variable has been explained.
- It indicates the goodness of the fit of the model.
- R-squared has the useful property that its scale is intuitive: it ranges from zero to one.

ANALYSIS

Data Exploration

The dataset has 326425 records and 9 columns. The columns are described below:

S.No	Column Name	Column Description
1.	“lat”	Latitude
2.	“lng”	Longitude
3.	“desc”	Description of emergency
4.	“zip”	ZIP code
5.	“title”	Title of emergency
6.	“timeStamp”	Date and time of the call
7.	“twp”	Town
8.	“addr”	Address
9.	“e”	This column is filled with a constant 1 for all rows

Table 1. *- Dataset columns and description*

6. https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models/

The first 2 rows from the dataset is provided below.

	lat	lng	desc	zip	title	timeStamp	twp	addr	e
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	12/10/2015 17:10	NEW HANOVER	REINDEER CT & DEAD END	1
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	12/10/2015 17:29	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	1

This dataset contains records from 10th December 2015 to 24th March 2018. The start and end date, and the number of records year-wise are given in the following screenshots.

```
Start Date in dataset : 2015-12-10 14:39:00
End date in dataset : 2018-03-24 13:15:00

Number of records in the year 2015 : 7916
Number of records in the year 2016 : 142360
Number of records in the year 2017 : 139617
Number of records in the year 2018 : 36532
```

Outlier Removal

The “twp” column which indicates the township is analyzed for unique town names. This is cross checked with google maps to identify that, 6 town names do not belong to Montgomery County at all. And those 6 values are neighbor to Montgomery county. They are: Delaware County, Chester County, Lehigh County, Bucks County, Berks County and Phila County. These are the outlier records (marked in red in the below screenshot) which needs to be removed.

```
print (data.twp.unique())
```

```
['NEW HANOVER' 'HATFIELD TOWNSHIP' 'NORRISTOWN' 'LOWER POTTSGROVE'
 'LANSDALE' 'HORSHAM' 'SKIPPACK' 'LOWER SALFORD' 'PLYMOUTH' 'MONTGOMERY'
 'UPPER MORELAND' 'CHELTENHAM' 'UPPER MERION' 'WHITEMARSH' 'UPPER GWYNEDD'
 'LOWER PROVIDENCE' 'UPPER DUBLIN' 'WHITPAIN' 'DELAWARE COUNTY'
 'FRANCONIA' 'WEST CONSHOHOCKEN' 'LOWER MERION' 'LIMERICK' 'TOWAMENCIN'
 'DOUGLASS' 'POTTSTOWN' 'BRIDGEPORT' 'AMBLER' 'CHESTER COUNTY'
 'UPPER HANOVER' 'SPRINGFIELD' 'ROCKLEDGE' 'ABINGTON' 'WEST NORRITON'
 'ROYERSFORD' 'UPPER SALFORD' 'LOWER MORELAND' 'CONSHOHOCKEN' 'PENNSBURG'
 'TELFORD' 'EAST NORRITON' 'UPPER FREDERICK' 'UPPER PROVIDENCE' 'SALFORD'
 'HATFIELD BORO' 'LEHIGH COUNTY' 'LOWER GWYNEDD' 'MARLBOROUGH'
 'BRYN ATHYN' 'HATBORO' 'WORCESTER' 'COLLEGEVILLE' 'SCHWENKSVILLE'
 'PERKIOMEN' 'SOUDERTON' 'UPPER POTTSGROVE' 'LOWER FREDERICK'
 'BUCKS COUNTY' 'RED HILL' 'WEST POTTSGROVE' 'EAST GREENVILLE'
 'BERKS COUNTY' 'NORTH WALES' 'JENKINTOWN' 'TRAPPE' nan 'NARBERTH'
 'GREEN LANE' 'PHILA COUNTY']
```

The total number of outlier records have been identified as 6664 records. These outlier records account for 2% of the dataset (6664 / 326425). The number of records present in each of the outlier towns is given in the below screenshot.

```
data['twp'][data['twp'].isin(['PHILA COUNTY',
                              'LEHIGH COUNTY',
                              'DELAWARE COUNTY',
                              'CHESTER COUNTY',
                              'BUCKS COUNTY',
                              'BERKS COUNTY'])].value_counts()
```

```
CHESTER COUNTY     3543
BUCKS COUNTY       1173
BERKS COUNTY        880
DELAWARE COUNTY     871
PHILA COUNTY        124
LEHIGH COUNTY        73
```

These outliers have been removed.

Error Record Removal

The range of latitude and longitude coordinates in the dataset is identified to be:

```
Minimum & Maximum latitude value: 30.333596000000004, 41.1671565
Minimum and Maximum longitude value: -95.5955947, -74.280113
```

But according to Google Maps, the rage of Montgomery County, PA ranges between, Latitude Range: (39.97, 40.46) and Longitude Range: (-75.71, -74.90). As we have already removed the town outliers, whatever town names remaining must obey the county's lat-lng bounds. Some points are far out of this range, but still having town names of Montgomery County as shown below:

```
print (list(lat_lng_outliers['twp']))
```

```
['HORSHAM', nan, 'EAST NORRITON', 'MONTGOMERY', 'UPPER PROVIDENCE', 'LOWER MERION', 'LOWER MERION', 'MONTGOMERY', 'SPRINGFIEL
D', 'LOWER MERION', 'UPPER PROVIDENCE', nan, nan, 'SPRINGFIELD', 'UPPER PROVIDENCE', 'NEW HANOVER', 'SPRINGFIELD', 'SPRINGFIEL
D', 'SPRINGFIELD', 'AMBLER', 'LOWER MERION', nan, 'SPRINGFIELD', 'ABINGTON', 'WEST NORRITON', 'CHELTENHAM', 'HATFIELD TOWNSHI
P', 'WEST CONSHOHOCKEN', 'LOWER MERION', nan, 'UPPER GWYNEDD', nan, 'LOWER SALFORD', 'WHITPAIN', 'AMBLER', 'LOWER SALFORD', 'UP
PER PROVIDENCE', 'LIMERICK', 'UPPER DUBLIN', 'HATFIELD TOWNSHIP', 'LOWER MERION', 'BRIDGEPORT', 'LOWER MERION', 'FRANCONIA']
```

```
print (len(lat_lng_outliers))
```

```
44
```

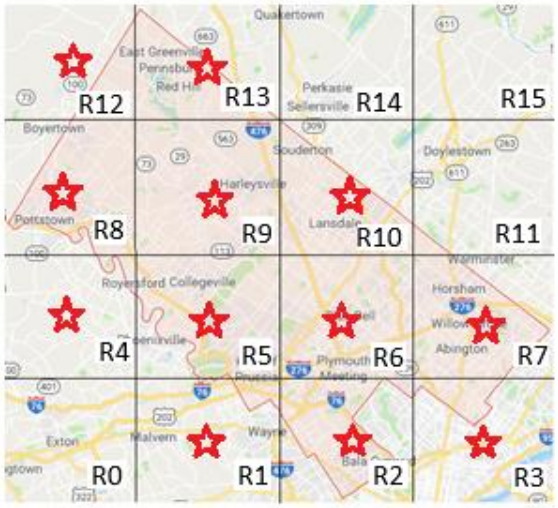
These are error records and have been removed.

Splitting into Regions

These latitude and longitude values are continuous values. So, these are converted to discrete values by assigning to a 4 x 4 grid and split into 16 regions.

The grid, regions, regions containing data points (star marked in red) and the number of emergency calls received per region are shown in the below figure. Map is obtained from Google Maps.

Based on the alignment of the Montgomery County from the below figure, it will be obvious that not all regions in the grid will have data points in them.




```
data['region'].value_counts()
```

R5	74554
R6	64712
R1	58806
R2	50647
R4	33316
R9	19319
R8	13998
R7	1515
R12	1461
R10	1238
R13	142
R3	9

Fig. 2 *–Partitioning into Regions (Red star – Regions containing datapoints)*

Calls on a sample day – 15th December 2015

For example, let us consider a sample day from the dataset - 15th December 2015.

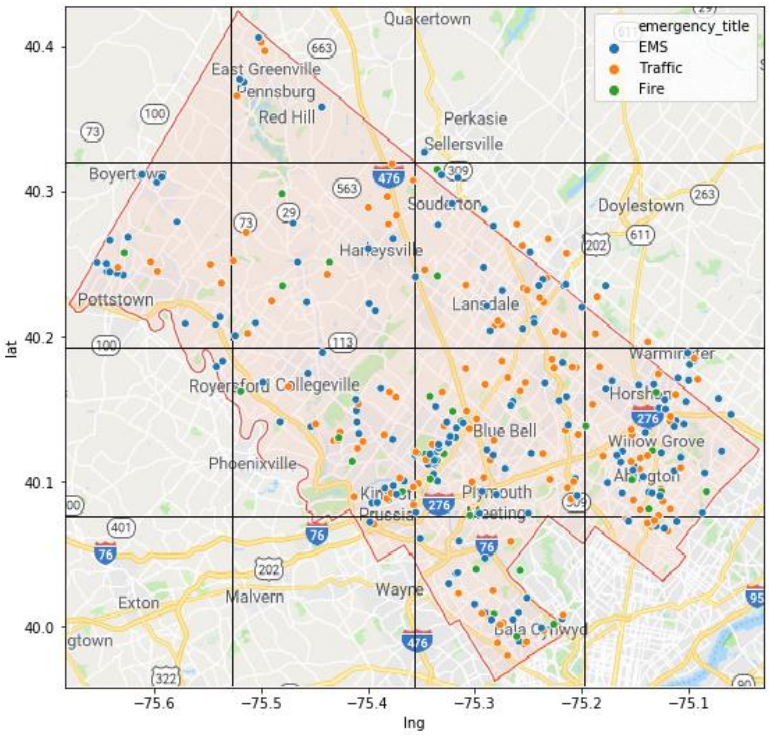


Fig. 3 *– Geographic Distribution of Calls on 15th Dec 2015*

The above figure shows the scatterplot of callers on 15th December 2015. Background image for the plot is obtained from Google Maps.

Missing Values

Column “zip” contains too many missing values and column “twp” contains few missing values as given below.

```
Missing Values in column - lat       : 0
Missing Values in column - lng       : 0
Missing Values in column - desc      : 0
Missing Values in column - zip       : 37009
Missing Values in column - title     : 0
Missing Values in column - timeStamp : 0
Missing Values in column - twp       : 109
Missing Values in column - addr      : 0
Missing Values in column - e         : 0
```

Columns to be Removed

Columns – “zip”, “twp”, “lat”, “lng”, “desc”, “addr”, “e” were removed and the reason is given below:

Reason for removing “zip” column: There are 2 reasons to remove this column. Reason 1: It has too many missing values (37009). Instead of removing the 37009 missing records and losing a lot of information, the zip column could be deleted. Reason 2: There are 138 unique zipcodes in total. The number of data points for each zipcode is given in the below screenshot.

zipcode	count	zipcode	count	zipcode	count	zipcode	count	zipcode	count	zipcode	count	zipcode	count
19401.0	22136	19001.0	5124	18074.0	1607	19128.0	383	19083.0	64	19082.0	5	36107.0	1	19073.0	1
19464.0	21588	19044.0	5087	19035.0	1607	18915.0	360	18932.0	37	18036.0	4	18104.0	1	19064.0	1
19403.0	16488	19010.0	4397	19041.0	1605	19453.0	293	19477.0	34	19520.0	3	17555.0	1	19063.0	1
19446.0	15793	18964.0	4265	19066.0	1539	19504.0	288	18092.0	26	19020.0	3	17506.0	1	19047.0	1
19406.0	10800	19004.0	4157	19465.0	1519	18070.0	222	19053.0	25	19348.0	3	8033.0	1	8361.0	1
19002.0	10305	19440.0	4127	19460.0	1468	19475.0	209	18951.0	24	19443.0	3	19018.0	1		
19468.0	9614	19473.0	3714	19405.0	1431	19009.0	198	18056.0	21	19404.0	3	77316.0	1		
19046.0	9043	19096.0	3703	18041.0	1363	19518.0	184	19115.0	19	18940.0	2	19607.0	1		
19454.0	8945	19003.0	3624	19075.0	1130	18914.0	173	19472.0	18	17752.0	2	19602.0	1		
19090.0	8638	19444.0	3513	18054.0	1092	19118.0	162	19423.0	17	18101.0	2	19486.0	1		
19038.0	8474	19095.0	3312	19087.0	1091	18976.0	162	19503.0	14	19057.0	2	19457.0	1		
19426.0	8184	19525.0	2967	18076.0	1069	19492.0	137	19138.0	14	19333.0	2	19450.0	1		
19006.0	7357	19034.0	2905	19151.0	987	18966.0	133	19355.0	10	19144.0	2	19382.0	1		
19428.0	7277	18073.0	2425	19085.0	940	19435.0	103	19116.0	9	19139.0	2	19380.0	1		
19438.0	6871	18969.0	2347	19150.0	881	19126.0	101	19119.0	7	19030.0	2	19127.0	1		
19040.0	6651	19072.0	2331	18936.0	831	18960.0	101	19474.0	6	19129.0	2	19026.0	1		
19462.0	6494	19012.0	2022	18974.0	775	19120.0	97	18958.0	6	18927.0	2	19121.0	1		
19422.0	6415	19031.0	2011	19512.0	706	19111.0	89	19107.0	6	18049.0	1	19104.0	1		
19027.0	6030	19025.0	1712	19131.0	437	19505.0	84	19490.0	5	18051.0	1	7203.0	1		

Out of a total of 138 zipcodes, 49 zipcodes have less than 10 entries in the dataset. There will be too few information to judge these zipcodes having too few entries. As the location information is already available through regions, the “zipcode” column has been removed.

Reason for removing “twp” column: Not all towns are of the same size. Some towns are too small to consider them as a single entity. Those small towns might have too few data points which would not be sufficient for training. So, instead of training the model town-wise, we have split the entire county into regions based on latitude-longitude coordinates. In that way we have good sized regions with enough data points in almost all regions for training. Another reason is that "twp" has a few missing values. So, this column "twp" has been removed.

Reason for removing columns “desc” and “addr”: Columns 'desc' and 'addr' contain description and address or similar information which is not needed for the problem in hand as we already have the regions derived from 'lat' and 'lng' column which indicates the location, and the “title” column to indicate the nature of emergency. So, “desc” and “addr” have been removed.

Reason for removing column “e”: Column 'e' is a constant that has value 1 for all rows. So, there is no use in retaining this column. So, column “e” has been removed.

Reason for removing “lat” and “lng”: Latitude and longitude values are continuous values, and it has already been categorized into regions to suit this problem. So, these “lat” and “lng” columns are no longer needed and has been removed.

“timeStamp” Column split

“timeStamp” column has continuous values. It is discretized by splitting it into the following: year, month, day, day_of_week and time. Time is further discretized by splitting it into 3 windows as: 0 => 12AM to 8AM; 1 => 8AM to 4PM; 2 => 4PM to 12AM.

“title” column encoding

This column indicates the type of emergency call. It has 133 unique values like 'EMS: BACK PAINS/INJUR Y', 'EMS: DIABETIC EMERGENCY', 'Fire: GAS-ODOR/LEAK', 'Traffic: DISABLED VEHICLE -', etc. Though ther e are 133 unique classes, they broadly fall under 3 classes: EMS, Fire or Traffic, and the corresponding ca tegory is noted down in the column “title_category”. The previous column “title” will no longer be referr ed, and therefore removed.

Exploratory Visualization

The below plot will give an intuition of what is available in the dataset. Here, date is plotted against the number of emergency calls received on that day. The category of emergency can be distinguished by their color (blue – EMS, orange – Fire, green - Traffic).

Fig. 4 – Calls in the Dataset – Date vs No. of calls

It is obvious that EMS (Emergency Medical Services) is the most occurred emergency type. The next is Traffic, and the least is Fire.

Another plot, a pie chart is given below which explains the same in a simple way.

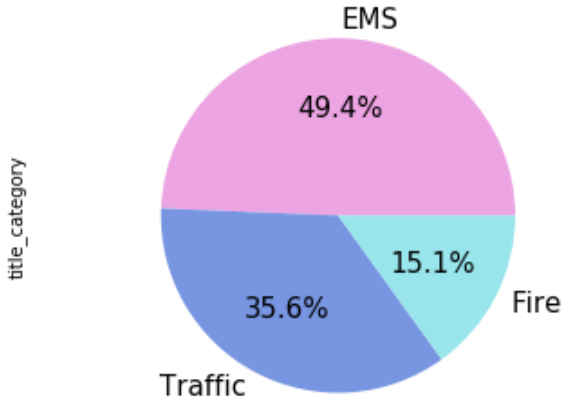


Fig. 5 *– Proportion of Various Emergency Categories in the Dataset*

Visualizing Calls on a Sample Day 15th December 2015:

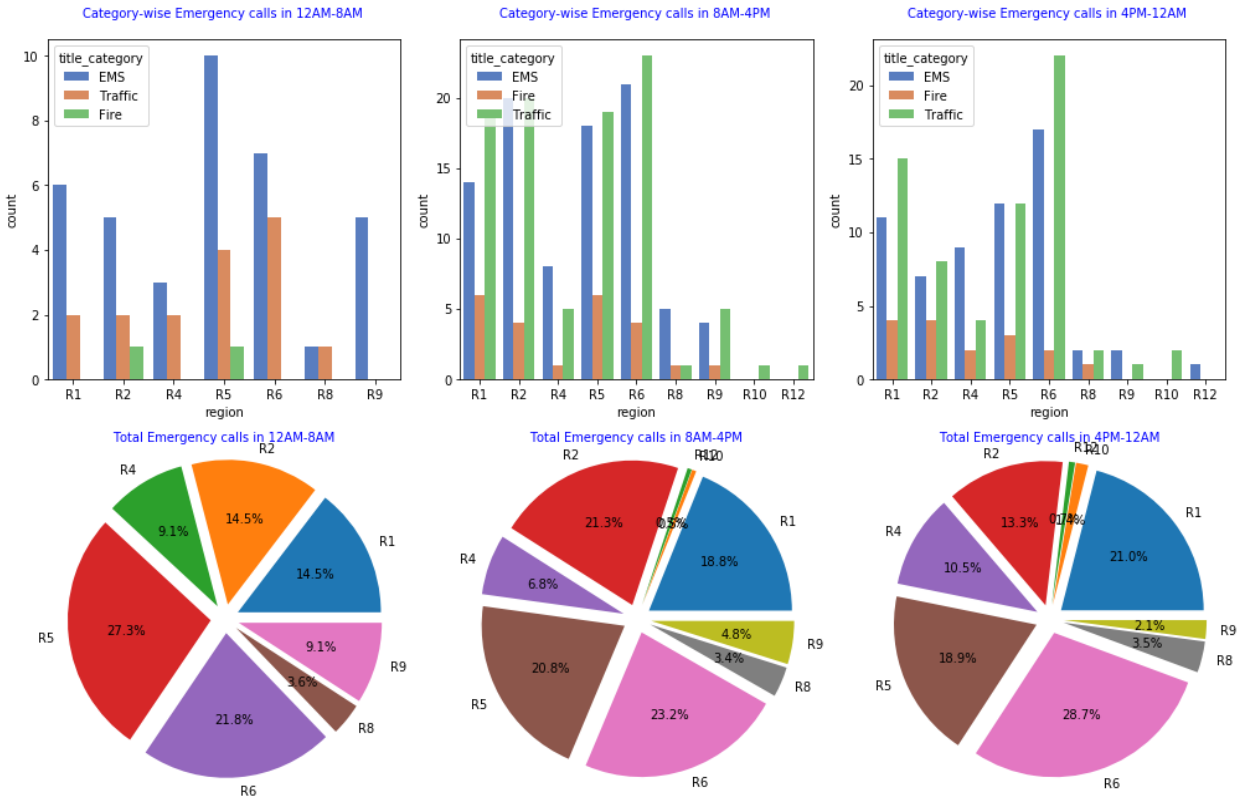


Fig. 6 *– Distribution of calls on 15th Dec 2015*

Analyzing the Feature - Month

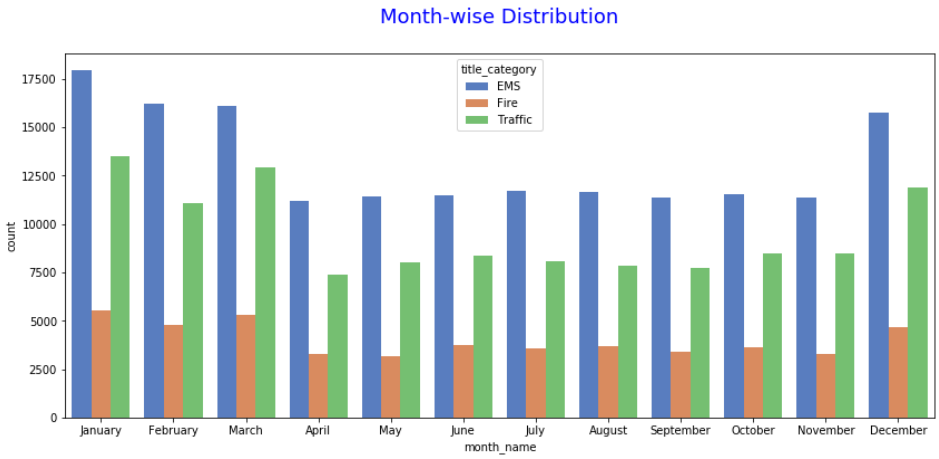


Fig. 7 *– Month-wise Emergencies Plot*

The above plot shows how many emergency calls are received in each month Jan to Dec. It looks like December, January, February, March have slightly higher values than other months. This is because the dataset ranges from 2015 December to 2018 March. In this duration, Dec, Jan, Feb and March occurs thrice, but other months occurs twice. That's why the count is higher in these months. Otherwise it is almost uniform.

Analyzing the Feature – Day

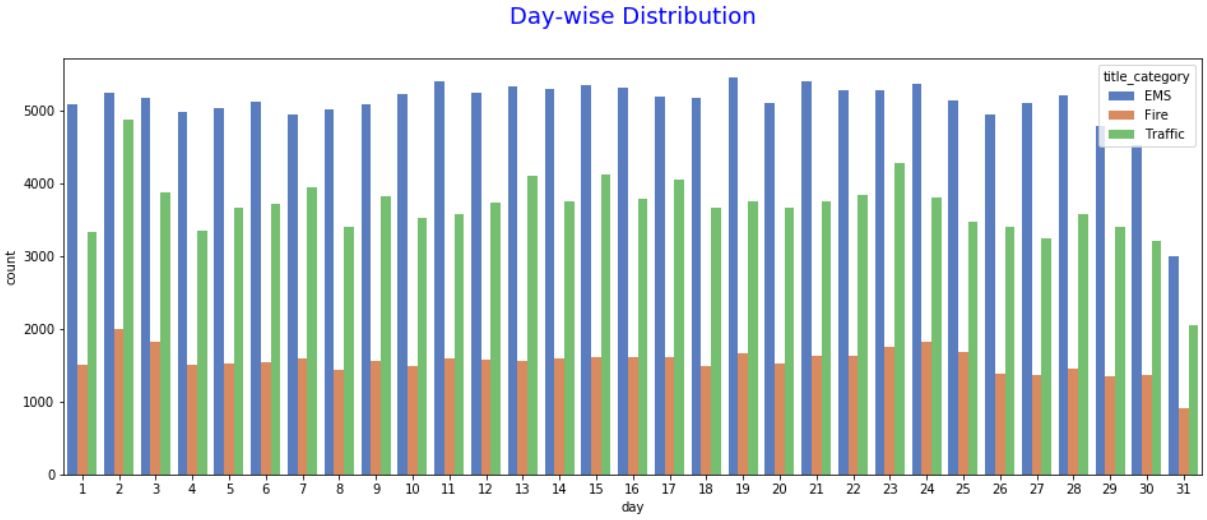


Fig. 8 *– Day-wise Emergencies Plot*

Traffic emergencies seems to be weirdly high on the 2nd of the month. Fire also shows increase on the same day. All emergencies seem to be on the low end on 31st which could be misleading, because, the day 31st doesn't occur in every month, thereby data will be few for that day compared to other days. So, being 31st doesn't actually mean emergencies are low on that day.

Analyzing the Feature – Weekday

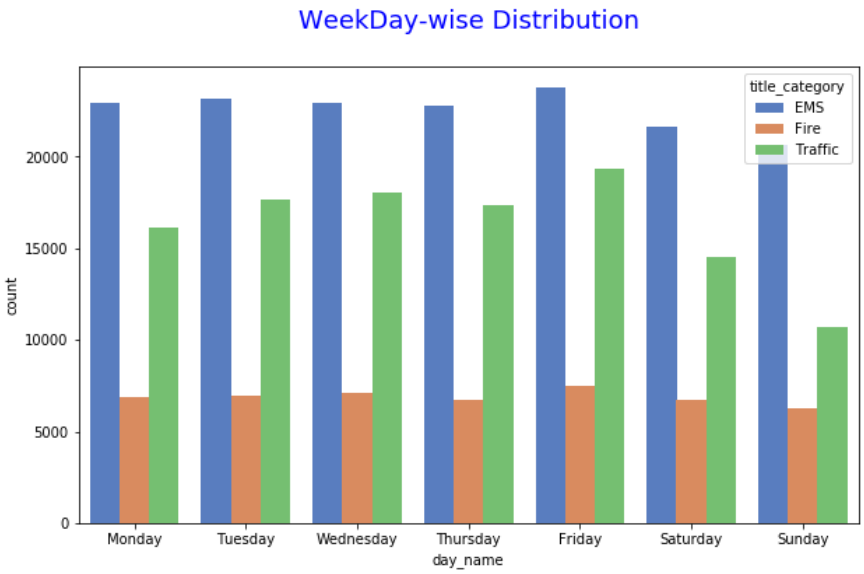


Fig. 9 *– Weekday-wise Emergencies Plot*

From this plot, Traffic (green bars) could be well described. Traffic incidents were high during weekdays mainly might be due to work traffic, highest on Friday where weekday meets weekend, and lowest on Sunday when most people relax at home.

Analyzing the Feature – Time window

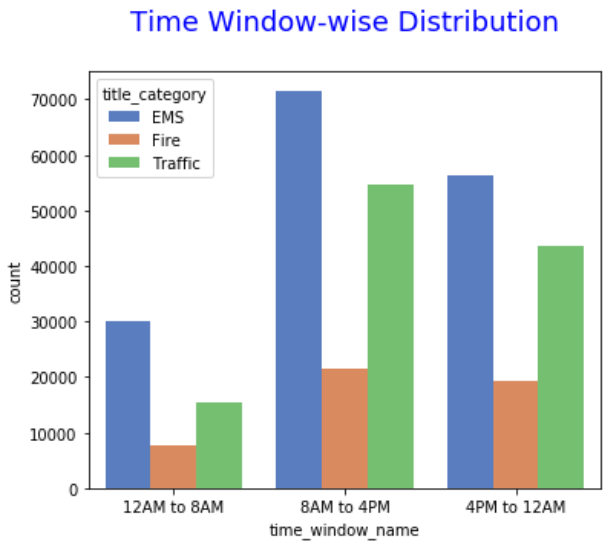


Fig. 10 – Time Window – wise Emergencies Plot

All the emergencies are highest during the busiest hours 8AM-4PM, medium in the evening, and the least in the early hours 12AM-8AM when most people are at home, sleeping.

Analyzing the Feature – Region

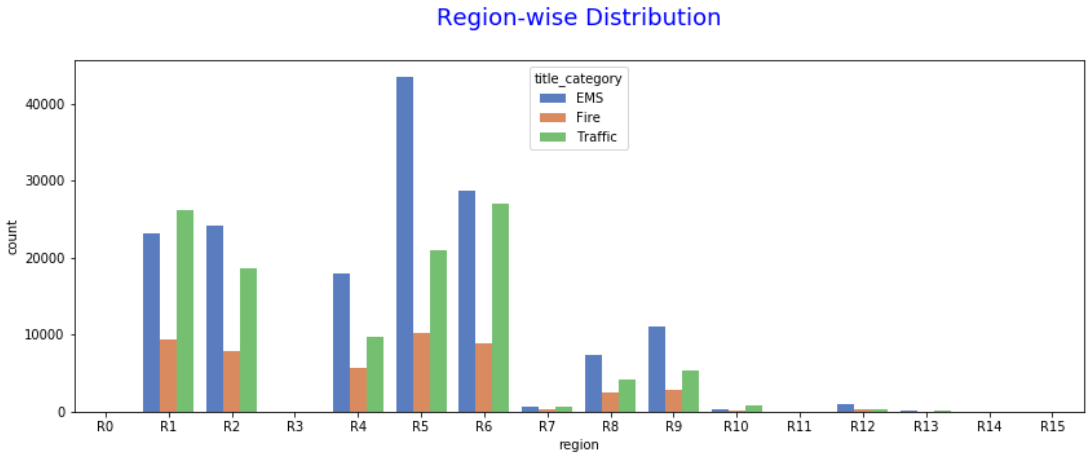


Fig. 11 – Region – wise Emergencies Plot

As already stated under "Splitting into Regions" section, all regions need not contain data points. The highest EMS emergencies occur in the region R5, and the highest traffic emergencies occur in the region R6 and R1.

Algorithms and Techniques

The algorithm used here is XGBoost Regression (xgboost.XGBRegressor). Its name stands for eXtreme Gradient Boosting[7], it was developed by Tianqi Chen and now is part of a wider collection of open-source libraries developed by the Distributed Machine Learning Community (DMLC). According to Tianqi Chen, "…xgboost used a more regularized model formalization to control over-fitting, which gives it better performance.". It was built and developed for the sole purpose of model performance and computational speed. XGBoost models dominate many Kaggle competitions.

The reason why this model is chosen for this context is that XGBoost is the leading model for working with standard tabular data. And also, it gives high performance and good at computational speed.

The data is already split for training and testing as 70:30 "in order" to not disturb the time series order of occurrence. Initially, an XGboost model with default set of parameters will be trained to find the r2 score on test set.

Then, this model will be tuned by playing around with the following hyperparameters:

- n_estimators
- early_stopping_rounds
- max_depth
- learning_rate

---

7. https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html

n_estimators is the number of boosted trees to fit. Too low a value causes underfitting, which is inaccurate predictions on both training data and new data. Too large a value causes overfitting, which is accurate predictions on training data, but inaccurate predictions on new data

The argument **early_stopping_rounds** offers a way to automatically find the ideal value. Early stopping causes the model to stop iterating when the validation score stops improving, even if we aren't at the hard stop for n_estimators. It's smart to set a high value for **n_estimators** and then use early_stopping_rounds to find the optimal time to stop iterating.

Max_depth is the maximum tree depth for base learners.

Learning_rate is the boosting learning rate.

Benchmark

Linear regression model (sklearn.linear_model.LinearRegression) has been implemented to predict the number of 911 calls expected in each emergency category on a given day, time window and location. This model is evaluated using R2 score as the evaluation metric. Given below are the R2 scores of the testing set using Linear Regression, and this will be used as the benchmark score.

	Linear Regression
EMS	0.645967834
Fire	0.224284911
Traffic	0.269204995

Table 2 *– R2 Scores on Benchmark Model (Linear Regression)*

METHODOLOGY

Data Preprocessing

As the latitude, longitude is already manipulated to give regions, and the timestamp is divided into many columns such as year, month, day, day_of_week, time_window, and the title column is broadly mapped to 3 categories such as EMS, Fire, Traffic, other steps will be discussed here.

Scaling

Given below are the various numerical feature and their corresponding range in the dataset.

- Year: 2015 to 2018
- Month: 1 to 12
- Day: 1 to 31
- Week: 1 to 53
- Day of week: 0 to 6
- Time window: 0 to 2

As the range of each feature is different from the other, they need to made equal so that they will be treated equally by the learning algorithm. So, scaling is applied by using *sklearn.preprocessing.MinMaxScaler* to numerical features.

One Hot Encoding

Given below are the various non-numeric features and their expected values in the dataset.

- Region: R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R12, R13
- Title category: EMS, Fire, Traffic

Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (categorical variables) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme by applying pandas.get_dummies(). One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature.

Rearranging the Dataset

As the number of emergencies in each title_category (EMS, Fire, Traffic) is not explicitly available, groupby is applied on the dataset to find the count of each title_category value when all other information in remaining columns remain the same.

Split the Dataset

The dataset is split for features and labels. The number of calls per Emergency title (title_category_EMS, title_category_Fire, title_category_Traffic) are the labels. The dataset is split “in order” (as this is time-series data) for training and testing - 70% of the data for training, and 30% of the data for testing.

Implementation

Sample records from the training features is given in the below screenshot:

	year	month	day	week	day_of_week	time_window	region_R8	region_R5	region_R1	region_R4	region_R6	region_R9	region_R2	region_R12
0	0.0	1.0	0.3	0.942308	0.5	0.5	0	1	0	0	0	0	0	0
1	0.0	1.0	0.3	0.942308	0.5	1.0	0	0	0	0	0	0	0	1
2	0.0	1.0	0.3	0.942308	0.5	1.0	0	0	0	0	0	0	1	0
3	0.0	1.0	0.3	0.942308	0.5	1.0	0	0	0	0	0	1	0	0
4	0.0	1.0	0.3	0.942308	0.5	1.0	0	0	0	0	1	0	0	0

region_R10	region_R7	region_R13	region_R3
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Sample records from the training labels is given in the below screenshot:

	title_category_EMS	title_category_Fire	title_category_Traffic
0	1	1	0
1	0	1	0
2	5	3	8
3	5	0	0
4	8	1	9

There are 3 cycles of training and predicting scores, one for each category of emergency. First an object of Xgboost regressor is created with default parameters as: **reg_xgb = xgb.XGBRegressor()**

The features and labels for the 3 cycles of training and prediction is given below:

	Training Feature	Training Label
For EMS	X_train	y_train['title_category_EMS']
For Fire	X_train	y_train['title_category_Fire']
For Traffic	X_train	y_train['title_category_Traffic']

Table 3 *– Training Features and Labels for Each Emergency Category*

For category EMS:

```
reg_xgb.fit(X_train, y_train['title_category_EMS'])
y_pred_ems = reg_xgb.predict(X_test)
score = r2_score(y_test['title_category_EMS'], y_pred_ems)
```

For category Fire:

```
reg_xgb.fit(X_train, y_train['title_category_Fire'])
y_pred_fire = reg_xgb.predict(X_test)
score = r2_score(y_test['title_category_Fire'], y_pred_fire)
```

For category Traffic:

```
reg_xgb.fit(X_train, y_train['title_category_Traffic'])
y_pred_traffic = reg_xgb.predict(X_test)
score = r2_score(y_test['title_category_Traffic'], y_pred_traffic)
```

The testing scores for all the 3 categories are noted down (provided in the next section).

Refinement

The default settings of Xgboost model has max_depth=3, learning_rate=0.1, n_estimators=100.

Now, n_estimators is set to be 1000. And, when training the model, early_stopping_rounds=100 (10% of n_estimators) is set so that if the validation loss does not improve over 100 iterations, the iteration process will be stopped. For this, 10% of the training set is already set aside as validation set.

Then, depth of the boosting model, and the learning rate is adjusted. Finally, it is found that, the below are the best possible parameters.

For category EMS:

xgb.XGBRegressor(n_estimators=1000, max_depth=4, learning_rate=0.01)

with early_stopping_rounds=100 during model fitting

For category Fire:

xgb.XGBRegressor(n_estimators=1000, max_depth=4, learning_rate=0.01)

with early_stopping_rounds=100 during model fitting

For category Traffic:

xgb.XGBRegressor(n_estimators=1000, max_depth=4, learning_rate=0.02)

with early_stopping_rounds=100 during model fitting

Given below are the results before and after tuning:

	XGB	**XGB Tuned**
EMS	0.793329909	0.80175959
Fire	0.271129041	0.270440526
Traffic	0.385370514	0.389280459

Table 4 *– R2 Scores of Xgboost – Default and Tuned Model*

RESULTS

Model Evaluation and Validation

For EMS and Traffic category, the tuning seems good. In the Fire category, tuning didn't help much.

Reason why tuning didn't help in category Fire:

This dataset consists of 49.4% EMS, 15.1% Fire and 35.6% Traffic emergencies. Already the data for Fire category in the dataset is less. For tuning process there is a need for validation set, and 10% of training set is set aside. So, the number of training records got still reduced. Therefore, there is no performance improvement in category Fire, due to insufficient training data in that category.

Comparing the raw version of XGB and tuned version of XGB, **the optimal model** for various categories are given below (highlighted scores in the above table "Table – 4"):

EMS: Tuned XGB model with n_estimators = 1000 (with early_stopping_rounds = 100), max_depth = 4, learning_rate = 0.01

Fire: Default XGB model with default set of parameters

Traffic: Tuned XGB model with n_estimators = 1000 (with early_stopping_rounds = 100), max_depth = 4, learning_rate = 0.02

The model is **tested with unseen data** which is the 30% testing data already kept aside. The model designed is a robust model. Here, **overfitting is avoided by using "early_stopping_rounds"** [8]. This uses a separate validation set and monitors the validation loss while training, and stopping the training process when the validation performance did not improve.

The model predicts EMS category (R2 score 0.80) better than the other two categories. For Traffic category (R2 score 0.39) and Fire category (R2 score 0.27), the model needs to be trained with more data in those categories to get good score.

---

8. https://www.kaggle.com/dansbecker/xgboost

Justification

The R2 scores of linear regression and XGBoost final model are given below:

	Benchmark	Final Model
	Linear Regression	**Optimal XGB model**
EMS	0.64	0.8
Fire	0.22	0.27
Traffic	0.27	0.39

Table 5 *– R2 Scores of Benchmark and Final model*

By comparing the R2 scores, the final model's performance is better than the benchmark model's performance. The final model plays a good role in predicting EMS emergencies, but still require more data to train for Fire and Traffic emergencies.

CONCLUSION

Free-form Visualization

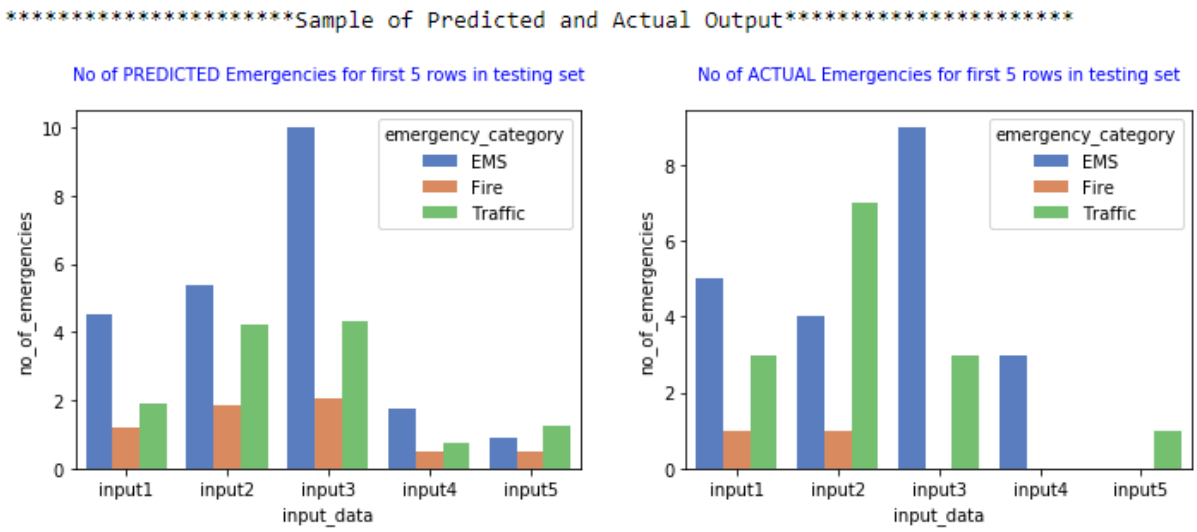


Fig. 13 *– Visualizing first 5 predictions (left) and corresponding actual values (right)*

The predicted number of emergencies, and the actual number of emergencies of the first 5 testing inputs are given in the above plot. 'input1', 'input2', etc in X axis corresponds to the test input rows from the testing set.

Reflection

- Selecting the dataset from Kaggle
- Analyzing the dataset for knowing what type of analysis to be done.
- Preparing project proposal based on the analysis
- Coding:
- Outlier and error record removal
- Forming grid based on latitude, longitude and partitioning the County into regions
- Extracting day, week, day of week, month, year and time information from timestamp
- Categorizing time by converting into 3 time-windows
- Data visualization plots
- Scaling (numeric features) and one hot encoding (non-numeric features)
- Broadly classify “title” column into 3 categories – EMS, Fire and Traffic
- Rearranging the dataset to find the number of emergencies per category
- Split the dataset into training and testing sets “in order” as this is time-series data
- Implement benchmark model – Linear regression
- Implement better model, “The Solution Model”– XGBoost Regressor
- Fine tune various hyperparameters.

The interesting part of the project was finding the outliers and error records. The difficult part was the problem I faced when splitting regions. I found the edge coordinates of the county from google maps, and then I set a relaxation of 2 for both the latitude and longitude so that I don’t miss any points in the margin. By this way I made region partition and found that all data points lie in only 3 regions in total 12 regions which is suspicious. So, I made a plot to find that the county is entirely zoomed out, with lot of extra space occupying other regions and that’s why those regions did not have data points. I realized that the entire county varies around 1 degree latitude and around 1 degree longitude, and setting a relaxation of 2 degrees caused the entire issue. Then I fixed this to find the correct regions.

This model could be used in a real setting to predict EMS emergencies, and to some extent Traffic emergencies. But for Fire emergencies, we need to add more training points to make the model good enough.

Improvement

I have been researching about Stacking models. This seems to be interesting, and could be used to improve the performance of the existing model. In stacking since it uses multiple model, prediction could be better than a single model.