Final Report

Project Title

Crime Rate Inference with Big Data

Team 5

Yang Guo C17424832 yguo3@clemson.edu

Qi Xiao C74871900 qxiao@clemson.edu

Abstract:

Crime is a very important sociological problem. Understanding the factors that lead to higher crime is crucial for government to make policies and improve citizen's life quality. In this report, we are trying to replicate and tackle the problem raised by this paper [1]: predict crime rate in one area by information from the neighborhood. This crime rate inference problem has always been of interest to sociologists, but in the past, research has only used data like traditional demographical and geographical information. In this paper, we chose Chicago, IL to study and used new types of data: point-of-interest (POI) data and taxi flow data. The result shows great improvement of crime rate inference using new features of data.

Introduction and Background:

Understanding what leads to crime and how to control crime is of great significance for U.S policy makers and citizens. According to [2], in 2014, U.S ranks No.11 for burglary and housebreaking rate in the world and No.15 for assault rate. Although the overall crime rate of U.S has declined since 1990s, it still remains relatively high compared to other countries [3]. Crime has severe consequences for citizen life and also influence social and economic development. Thus understanding how to control crime is crucial and we are specifically interested in the neighborhood context of crime: how crime in one area is affected by neighborhood crime pattern.

We chose Chicago, IL to study the problem of crime rate inference of communities. Chicago has very severe crime problem compared to other big cities in the U.S. The city has substantially higher overall crime rate than US average, but the reason to this high crime rate remains unclear [4]. Since Chicago Police Department has been tracking crime since early 20th century, this city provides a good example to study crime rate inference problem due to relatively sufficient data resources.

Traditionally, researchers have used demographic and geographical information to tackle crime rate inference problem. But only using this two information has some limitations. Demographic data like population, poverty level, racial composition, etc has been used to estimate crime rate in an area. But it only provides a partial picture of the area and such data is often not up to date because U.S census is performed every 10 years. As for geographical data, researchers have found that adding geographical data to demographic data only minorly improve crime estimate, maybe because neighboring areas usually share similar demographic features.

Thus we are interested in using new data features to predict neighborhood crime rate. The paper [1] proposed two new features: point-of-interest (POI) data and taxi flow data. POI data provides venue information in an area, it includes places like schools, restaurants, government buildings, etc. And it is expected to provide more information about an area. Meanwhile, taxi flow data gives us insight about how people commute

across different neighborhoods. Traditional geographical data focuses on interactions between nearby neighborhoods, but we know sometimes criminalists can travel to areas not nearby to commit crimes. Although criminalists may not take taxis, taxi flow data can reflect the human trafficking patterns across the city.

The report will be organized as follows. First we will discuss the approaches we take to tackle crime inference problems. It will include two parts. One is data collection and data preprocessing, in which we will talk about where and how we collected different data features. The second part will cover the statistical models we are going to test to predict crime based on the data features collected. After the approaches section, results of different inference models will be shown and the implications will be discussed.

Approaches:

In this section, we will discuss the approaches we take for crime inference problem. In general, we want to predict crime rate in one area based on information from this area and other neighborhoods. The first part will cover our data collection process, and the second part will discuss different inference models to predict crime rate.

Data Collection and Preprocessing:

1. Chicago community areas and crime rate

We choose Chicago, IL to study the crime inference problem. And we use the term community area as the basic region unit to study crime. Chicago have 77 community areas as shown in Fig.1 [1]. The main reason we choose community area to study instead of county, census tract, etc is because Chicago police department has well documented incidents of crimes and each crime has community area information associated with it. It also has police district information, but it will be hard to find demographic data associated with police district.

We collected crime incidents data from [6]. We calculated the total crime counts in each community area and converted into crime rate (crime count / total population in each community area). Fig 2. shows the crime rate in each community area in 2013.

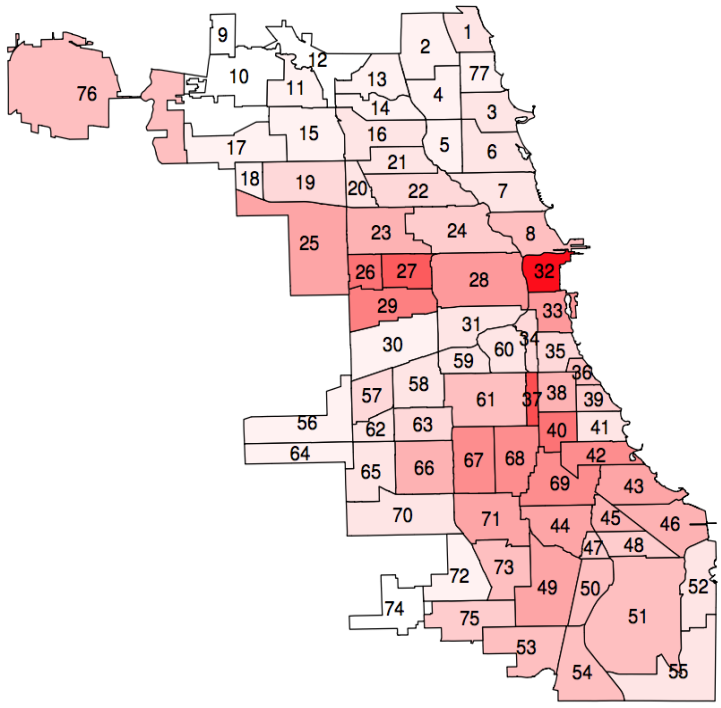


FIGURE 1. CHICAGO COMMUNITY AREA MAP

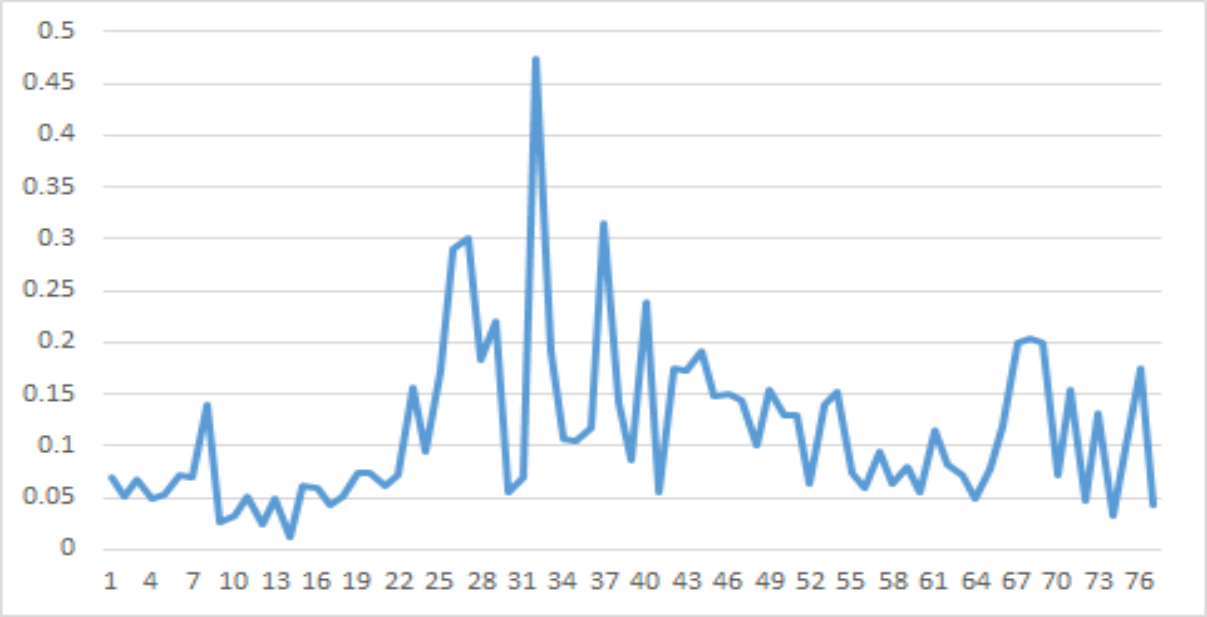


FIGURE 2. CRIME RATE IN 77 COMMUNITY AREAS IN 2013

2. Demographics:

We collect demographic information mainly from US Census Bureau's census website. To use demographic information as a predictor of crime rate, we use year 2000 data. We include these features in our demographics:

Total population, Poverty index, Percentage of main races

Table 1 shows correlation between demographic figures and crime rate at community area level. There are 77 communities in Chicago, it shows that crime rate of each community and poverty index has strong positive correlation while crime rate and percentage of NHW has negative correlation.

TABLE 1 CORRELATION BETWEEN DEMOGRAPHIC FEATURES AND CRIME RATE

Feature	Correlation with crime rate	p-value
Total population	-0.180000392490147	0.117234718388481
Poverty Index	0.522318571068164	1.10340078410944E-06
Percentage of NHB	0.62961506243669	8.57758210567115E-10
Percentage of NHW	-0.472578450495678	0.0000142734615258744
Percentage of Hispanic	-0.392972969	0.000407342672775797

To use demographic features in our inference model, we generate demographics matrix using numpy library. Here's the code:

```
def generate_demographics_feature(leaveOut = -1):
    f = open('data/demographics.csv')
    reader_demo = csv.DictReader(f)
    features = []
    flag = 0
    for row in reader_demo:
        if int(row['Community #']) > flag:
            features.append([row['Total_Population'], row['NHW_P'], row['NHB_P'],
                    row['Poverty_P']])
            flag += 1

    features = np.asarray(features, dtype=np.float)

    if leaveOut > 0:
        features = np.delete(features, leaveOut - 1, 0)

    return features
```

Because we saved demographic features and crime rate of each community in the file ‘data/demographics.csv’, when we want to add demographic feature to our feature set, we should extract specific features from the file and formatted them as numpy array.

3. Point-of-Interest (POI)

Point-of-interest data shows the venue information in an area. For example, how many restaurants, schools or public buildings are in an area. We collect POI data using foursquare venue services [7].

Foursquare explore function will return venues information in JSON format within a radius of a given GPS coordinates. Thus, we iterate over the whole Chicago area and try to crawl all POI data. The problem is each query is limited to a maximum of 50 results. So, if we get 50 venues, that probably means there are more than 50 venues and we need to divide the search region into smaller areas and search again. Also in venue information, only subcategory name is provided. But we’d like to categorize all venues into 10 main categories. So category hierarchy information is also crawled from Foursquare API. The main part of the codes is shown below:

```
public static void main(String[] args) throws Exception {
    // set start time
   startTime = System.currentTimeMillis();
   System.out.println("start: " + startTime);
   poi(chicago_lat_min, chicago_lat_max, chicago_lon_min, chicago_lon_max,100,100);
}


private static void poi(double lat_min, double lat_max, double lon_min, double lon_max, int
row, int col) throws Exception{
   double delta_lon = (lon_max - lon_min) / col;
   double delta_lat = (lat_max - lat_min) / row;
   for (int i = 0; i < row; i++) {
      System.out.println("row #: " + i + "\t" + System.currentTimeMillis() + "\t count: " +
requestCount);
      for (int j = 0; j < col; j++) {
         // (lat1,lon1) is upper left corner, (lat2,lon2) is lower right corner
         double lat1 = lat_max - i * delta_lat;
         double lon1 = lon_min + j * delta_lon;
         double lat2 = lat1 - delta_lat;
         double lon2 = lon1 + delta_lon;
```

```
        double radius = 0.5 * 1000 * distance(lat1,lon1,lat2,lon2,"K");  // unit is meters
        double lat_mid = lat1 - 0.5 * delta_lat;
        double lon_mid = lon1 + 0.5 * delta_lon;

        // if duration < 1hr, count < 4999, explore url
        // else if duration < 1hr, count >= 4999, sleep until 1hr
        // else if duration > 1hr, reset start time to now, reset count to 0
        long now = System.currentTimeMillis();
        if ((now - startTime) < 3600000) {
            if (requestCount >= 4998) {
                long wait = 3600000 - (now - startTime);
                try {
                    TimeUnit.MILLISECONDS.sleep(wait);
                    System.out.println("sleep for " + wait);
                    startTime = System.currentTimeMillis();
                    requestCount = 0;
                } catch (InterruptedException e) {
                    System.out.println("sleep is interrupted!");
                }
            }
        } else {
            System.out.println("a new hour start!");
            startTime = System.currentTimeMillis();
            requestCount = 0;
        }

        URL explore = new URL("https://api.foursquare.com/v2/venues/explore?ll=" +
                Double.toString(lat_mid) + "," + Double.toString(lon_mid) +
"&client_id=WUVLK1LZL5XM3HQVKIYHNM4CAUMRHVAXEWB3LZU2MCO5WXP2" +
"&client_secret=GEY31LO5AH0MOB54NW4S1DCYYKB130HMOG3ZO4BUZT5FDUTC" +
                "&v=20170101" +
                "&m=foursquare" +
```

```
                "&limit=50" +
                "&radius=" + Double.toString(radius));

        requestCount++;

        try (InputStream is = explore.openStream();
            JsonReader rdr = Json.createReader(is);
            Connection conn = DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/foursquare?useSSL=false", "qi",
"1234@Abcd");
            Statement stmt = conn.createStatement()) {

            JsonObject obj = rdr.readObject();
            JsonObject response = obj.getJsonObject("response");
            JsonArray groups = response.getJsonArray("groups");
            JsonObject groups_obj = groups.getJsonObject(0);
            JsonArray items = groups_obj.getJsonArray("items");

            if (items.size() < 50) {
               for (JsonObject item : items.getValuesAs(JsonObject.class)) {

                  JsonObject venue = item.getJsonObject("venue");
                  String name = venue.getString("name");
                  JsonObject location = venue.getJsonObject("location");
                  JsonNumber lat = location.getJsonNumber("lat");
                  JsonNumber lng = location.getJsonNumber("lng");
                  JsonArray categories = venue.getJsonArray("categories");
                  JsonObject category_obj = categories.getJsonObject(0);
                  String sub_category = category_obj.getString("name");

                  String main_category = findMainCategory(sub_category, stmt);

                  // insert ignore: no duplicate rows
                  String query = "insert ignore into poi values(" +
Double.parseDouble(lat.toString()) + ","
```

```
                        + Double.parseDouble(lng.toString()) + "," + "\"" + main_category + "\"" +
")" ;
                    stmt.executeUpdate(query);
                }
            } else {     // query result reaches limit of 50, need to further divide search area into
smaller range
                poi(lat2, lat1, lon1, lon2, 5, 5);
            }
        }
    }
  }
}
```

After crawling POI data from FourSquare and classify them into 10 main categories:

food, residence, travel, arts & entertainment, outdoors & recreation,

college & education, nightlife, professional, shops and event

We got about more than 40000 POIs. Since the location information of each POI is presented as coordinates, we should transform its location information to the id of community area where it's located using shapefile of Chicago. And then count the number of POI by its category in each community. Here's the code:

```
def generate_poiCount():
    f_shape = fiona.open('shapeFile_Chicago/geo_export_01.shp')

    f_poi = open('data/poi_original.csv')
    reader_poi = csv.reader(f_poi, delimiter = ',')

    matrix_poi = np.zeros((77,9))
    # only one event poi, so just use 9 poi features
    index_poi = {'Food':0, 'Residence':1, 'Travel & Transport':2, 'Arts &
        Entertainment':3, 'Outdoors & Recreation':4,
       'College & University':5, 'Nightlife Spot':6, 'Professional & Other Places':7,
        'Shop & Service':8}

    for row in reader_poi:
       point = Point(float(row[1]), float(row[0]))
       for community_shape in f_shape:
```

```
            s = asShape(community_shape['geometry'])
            if s.contains(point):
                matrix_poi[int(community_shape['properties']['area_numbe'])-1][index_poi[
                    row[2]]]+= 1
                break

    np.savetxt('data/poi_cnt_matrix.csv', matrix_poi, delimiter= ',')
    print(matrix_poi)
```

When we want to use POI feature in our inference model, we just need to retrieve it from previously saved POI matrix file.

4. Geographical Influence

We downloaded shapefile of Chicago from https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Community-Areas-current-/cauq-8yn6/data

It contains geometry information of each community. We can use this shapefile to determine which community area a specific place is located. But it's very hard to get the centroid coordinate through using shapefile. So we use a part of taxi trips data set to get centroid of every community area because in taxi trips data set it contains centroid coordinate and its paired community id number. Here's the code we used to generate centroid information:

```
import csv
f = open('data/taxiTripsPart.csv')
reader = csv.DictReader(f)


res = []
flag = 0
for row in reader:
  if int(row['Pickup Community Area']) > flag:
      res.append([row['Pickup Community Area'], row['Pickup Centroid Latitude'],
                  row['Pickup Centroid Longitude']])
      flag += 1


outfile = open('data/centroid_community.csv','w')
writer = csv.writer(outfile, delimiter=';', quotechar='"')
```

```
    writer.writerows(res)
    outfile.close()
```

After this step, we can generate geographical influence spatial matrix W. From the definition of matrix W, if region i and region j are not geospatially adjacent, $W_{ij} = 0$; otherwise, $W_{ij} \propto 1/dist(i,j)$.

While in practice, we calculate distance between region i and all other regions and generate a spatial matrix, then we only remain the largest 6 value in each row of the matrix and set all other value as 0. In this way we can easily get an approximate accurate spatial matrix W. Here is the code:

```
def generate_geographical_influence(leaveOut = -1):
    '''
    :param leaveOut: select community area and remove from the training set
    :return: 1/distance matrix
    '''
    f_Chicago = fiona.open('shapeFile_Chicago/geo_export_01.shp')
    centroids = csv.reader(open('data/centroid_community.csv'), delimiter = ';')
    cas = {}

    for row in centroids:
        cas[int(row[0])] = [float(row[1]), float(row[2])]

    cSet = list(range(1, 78))
    if leaveOut > 0:
        cSet.remove(leaveOut)
    # print(cSet)

    centers = {}
    for i in cSet:
        centers[i] = cas[i]

    W = np.zeros((len(cSet),len(cSet)))
    for i in cSet:
        for j in cSet:
            if i != j:
```

```
            if leaveOut < 0:
                W[i-1][j-1] = 1 / sqrt((1000*(centers[i][0]-centers[j][0]))**2 + (1000*(centers[i][1]-
                                        centers[j][1]))**2)
            elif leaveOut > 0:
                k = i
                l = j
                if i > leaveOut:
                    k -= 1
                if j > leaveOut:
                    l -= 1
                W[k - 1][l - 1] = 1 / sqrt( (1000 * (centers[i][0] - centers[j][0])) ** 2 + (1000 *
                                        (centers[i][1] - centers[j][1])) ** 2)

    for i in range(len(cSet)):
        # find largest 6 value
        threshold = heapq.nlargest(6, W[i, :])[-1]
        for j in range(len(cSet)):
            if W[i ][j] < threshold:
                W[i][j] = 0
    return W
```

5. Hyperlinks by Taxi Flow

We download taxi trips dataset between October and December in 2013 from:

https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew/data

To generate taxi flow feature for the inference model, first we count the total number of taxi trips from region i to region j. It will generate a 77x77 matrix. Here's the code:

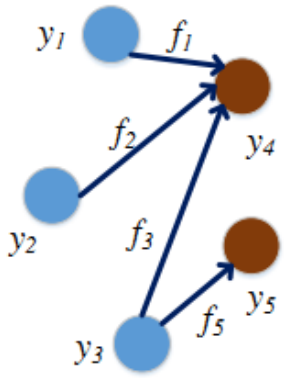
```
def generateTFCount():
    f = open('data/Taxi_Trips.csv')   # This file is too large, so we provide link of it in   report.
    reader = csv.DictReader(f)
    matrix_count = np.zeros((77,77))
    for row in reader:
        matrix_count[int(row['Pickup Community Area'])-1, int(row['Dropoff
                                CommunityArea'])-1] += 1
```

```
    outfile = open('data/taxi_flow_count.csv', 'w')
    writer = csv.writer(outfile, delimiter=';', quotechar='"')
    writer.writerows(matrix_count)
    outfile.close()

def generateTFMatrix():
    f = open('data/taxi_flow_count.csv')
    reader = csv.reader(f, delimiter = ';')
    matrixTF = np.zeros((77,77))
    rowNum = 0
    for row in reader:
        matrixTF[rowNum] = row
        matrixTF[rowNum][rowNum] = 0
        rowNum += 1

    outfile = open('data/taxi_flow_matrix.csv', 'w')
    writer = csv.writer(outfile, delimiter=';', quotechar='"')
    writer.writerows(matrixTF)
    outfile.close()
```

When we incorporate taxi flow feature into inference model, we can use raw count matrix. However, one issue with raw count is that most taxi trips is concentrated in downtown area. To address this issue, we can normalize the taxi flow either by source or by destination. The figure below shows how to normalize taxi flow.



Normalization by source:

$$F_4^t = \frac{f_1}{f_1} y_1 + \frac{f_2}{f_2} y_2 + \frac{f_3}{f_3+f_5} y_3$$

Normalization by destination:

$$F_4^t = \frac{f_1}{f_1+f_2+f_3} y_1 + \frac{f_2}{f_1+f_2+f_3} y_2 + \frac{f_3}{f_1+f_2+f_3} y_3$$

Here's the code for normalization:

```
def taxiFlowNormalization(tf_matrix, method='byDestination'):
```

```
    '''
    :param tf_matrix: taxi_flow_matrix, may be modified by leaveOut factor
    :param method: you can choose normalization method between: byDestination, bySource or
            byCount
    :return: normalized matrix
    '''
    n = tf_matrix.shape[0]
    tf_matrix = tf_matrix.astype(float)
    if method == "byDestination":
        tf_matrix = np.transpose(tf_matrix)
        fsum = np.sum(tf_matrix, axis=1, keepdims=True)
        fsum[fsum == 0] = 1
        assert fsum.shape == (n, 1)
        tf_matrix = tf_matrix / fsum
        np.testing.assert_almost_equal(tf_matrix.sum(), n)
        np.testing.assert_almost_equal(tf_matrix.sum(axis=1)[n - 1], 1)
    elif method == "bySource":
        fsum = np.sum(tf_matrix, axis=1, keepdims=True)
        fsum[fsum == 0] = 1
        tf_matrix = tf_matrix / fsum
        assert fsum.shape == (n, 1)
        np.testing.assert_almost_equal(tf_matrix.sum(axis=1)[n - 1], 1)
    elif method == "byCount":
        pass

    return tf_matrix
```

Inference Models:

Now that we have collected 4 different types of data in community areas, the next thing is to choose a proper inference model to predict crime rates. The paper suggested two models and we are going to build both.

Linear Regression (LR) Model:

Linear regression is one of the simplest prediction model. The input parameters should include nodal features like demographical and POI data in the community area we are trying to predict. Also, neighborhood crime rates information with proper weight matrix.

Equation below gives the linear regression formulation of our problem.

$$\vec{y} = \vec{\alpha}^T \vec{x} + \beta^f W^f \vec{y} + \beta^g W^g \vec{y} + \vec{\epsilon}$$

In our implementation, we use linear_model from sklearn library:

```
from sklearn import linear_model
def linearRegression(features, Y):
    model_linear = linear_model.LinearRegression()
    res = model_linear.fit(features, Y)
    return res
```

Negative Binomial (NB) Model:

Crime count is positive integers, so linear regression is not the optimal model for such data types. Poisson regression is a better model for such count data. However, poisson regression requires the mean and variance of dependent variable to be equal, which is not usually the case in reality. Thus we can use the Poisson-Gamma model, where the mean of y is a random variable with gamma distribution. Such model is also called negative binomial model.

In negative binomial regression, the link function is:

$$\mathrm{E}(y) = e^{Xw+\epsilon}$$

In our implementation, we import statsmodels.api and use it to construct NB model:

```
import statsmodels.api as sm


model_NB = sm.GLM(train_Y, train_features, family=sm.families.NegativeBinomial())
model_res = model_NB.fit()
```

Preparing features for inference model:

It's very important to construct features to train the model, since we may use different feature combination to train the model. And when we train the model, we will only use training set, the leaveOut argument indicates which community feature is used for testing. Here's the code:

```
def generateFeatures(features = ['demos', 'poi', 'geo', 'tf'], leaveOut = -1):
    Y = retrieve_crimerate()
    Y = Y.reshape((-1,1))

    if leaveOut > 0:
       Y = np.delete(Y, leaveOut-1, 0)
```

```
    if 'demos' in features:
        demos = generate_demographics_feature(leaveOut)
    if 'poi' in features:
        poi = getPOICount(leaveOut)
    if 'geo' in features:
        geo = generate_geographical_influence(leaveOut)
        f_geo = np.dot(geo, Y)
    if 'tf' in features:
        tf = getTaxiFlow(leaveOut, 'byDestination')
        f_taxi = np.dot(tf, Y)

    F = np.ones(f_geo.shape)
    if 'demos' in features:
        F = np.concatenate((F, demos), axis= 1)
    if 'poi' in features:
        F = np.concatenate((F, poi), axis= 1)
    if 'geo' in features:
        F = np.concatenate((F, f_geo), axis= 1)
    if 'tf' in features:
        F = np.concatenate((F, f_taxi), axis= 1)

    return F, Y
```

Results:

Since we have built LR and NB inference model. We can use them to estimate crime rate of each community and save the calculated crime rate value.

```
def getPredictedCrimeRate():
    '''
    Calculate and save predicted crime rate by LR and NBR.
    '''
    res = []
    F, Y = generateFeatures()
    for i in range(len(Y)):
```

```
        F_train, Y_train = generateFeatures(leaveOut=i+1)

        mod_LR = linearRegression(F_train, Y_train)
        y_predict_LR = mod_LR.predict(F[i,])
        model_NB = sm.GLM(Y_train, F_train, family=sm.families.NegativeBinomial())
        model_res = model_NB.fit()
        y_predict_NBR = model_NB.predict(model_res.params, F[i,])

        res.append([y_predict_LR[0][0], y_predict_NBR])

    outfile = open('result/crimeRate_predict.csv', 'w')
    writer = csv.writer(outfile, delimiter=',', quotechar='"')
    writer.writerows(res)
    outfile.close()
```

After running the program, we get the estimated crime rate result from LR and NBR model. Then plot them with the real crime rate in the same chart.

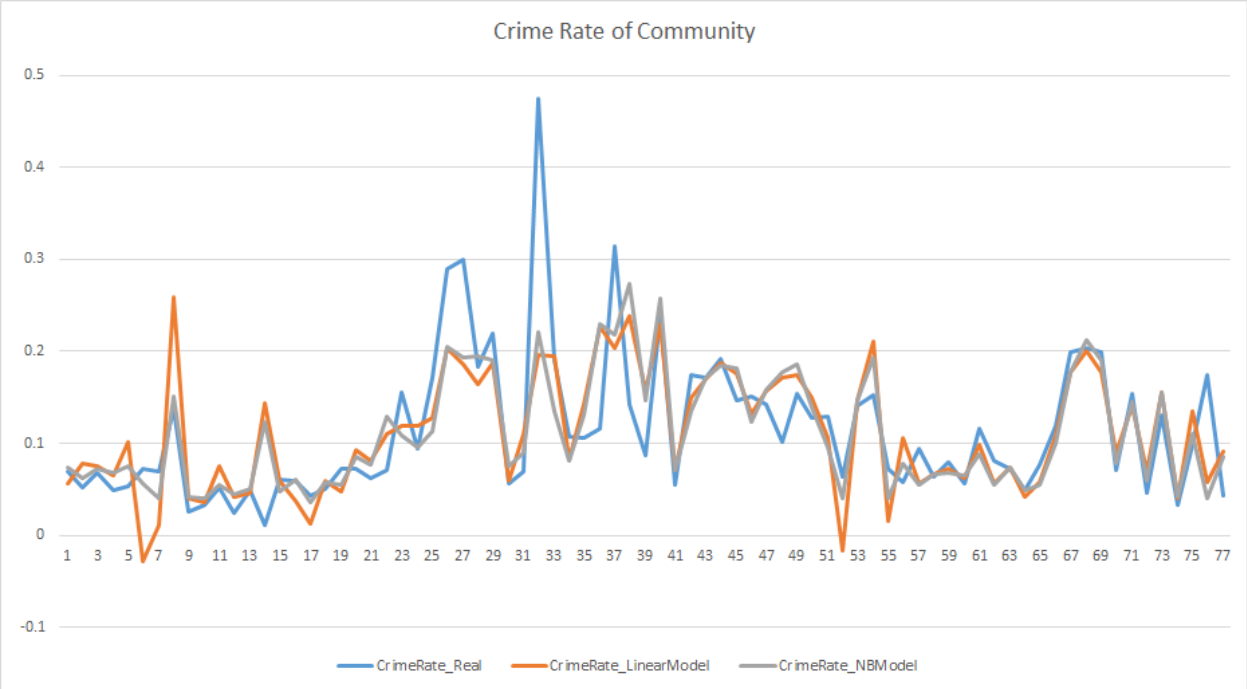


FIGURE 3. CRIME RATE COMPARISON BETWEEN REAL VALUE AND ESTIMATED VALUE

From Figure 3 we can see that in most areas except community area 27,32, the crime rate value of real world and crime rate calculated from our model is quite close. It shows that our model is appropriate to estimate crime rate.

The reason for the error in area 27 and 32 we think may be is because crime rates in these areas are too much higher than other area, so it's very hard to get an accurate result just using data from other area to train the model.

Performance Evaluation:

In the evaluation, the accuracy of estimation is evaluated by mean absolute error (MAE) and mean relative error (MRE).

- $\text{MAE} = \frac{\sum_i^n |y_i - \hat{y}_i|}{n}$
- $\text{MRE} = \frac{\sum_i^n |y_i - \hat{y}_i|}{\sum_i^n y_i}$

Here is the code for calculating MAE and MRE of Linear Regression Model:

```
def LRTraining(features, Y):
    errors = []
    loo = LeaveOneOut()
    for train_index, test_index in loo.split(Y):
        train_Y = Y[train_index]
        train_features = features[train_index]
        model_LR = linearRegression(train_features, train_Y)
        y_predict = model_LR.predict(features[test_index])
        errors.append(abs(y_predict - Y[test_index]))

    return np.mean(errors), np.mean(errors)/np.mean(Y)
```

And here's the code for calculating MAE and MRE of NB Regression Model:

```
def NBTraining(features, Y):
    '''
    Use statsmodels library.
    '''
    errors = []
    loo = LeaveOneOut()
    for train_index, test_index in loo.split(Y):
        train_Y = Y[train_index]
        train_features = features[train_index]
```

```
        model_NB = sm.GLM(train_Y, train_features,
                          family=sm.families.NegativeBinomial())
        model_res = model_NB.fit()
        y_predict = model_NB.predict(model_res.params, features[test_index])
        errors.append(abs(y_predict - Y[test_index]))

    return np.mean(errors), np.mean(errors)/np.mean(Y)
```

From above we can see that to calculate MAE and MRE, we split the data into training area and testing area. We use data of training area to construct and fit the model, then estimate crime rate for testing area. After having estimated crime rate for all the community area, we can get MAE and MRE value.

We use four kinds of feature combination to do the performance evaluation.

```
def generateMAE_MRE():
    F, Y = generateFeatures(features=['demos', 'geo'])
    mae1_DG, mre1_DG = LRTraining(F, Y)
    mae2_DG, mre2_DG = NBTraining(F, Y)

    F, Y = generateFeatures(features=['demos', 'geo', 'tf'])
    mae1_DGT, mre1_DGT = LRTraining(F, Y)
    mae2_DGT, mre2_DGT = NBTraining(F, Y)

    F, Y = generateFeatures(features=['demos', 'geo', 'poi'])
    mae1_DGP, mre1_DGP = LRTraining(F, Y)
    mae2_DGP, mre2_DGP = NBTraining(F, Y)

    F, Y = generateFeatures()
    mae1_ALL, mre1_ALL = LRTraining(F, Y)
    mae2_ALL, mre2_ALL = NBTraining(F, Y)

    res = [
        [mae1_DG, mae1_DGT, mae1_DGP, mae1_ALL],
        [mre1_DG, mre1_DGT, mre1_DGP, mre1_ALL],
        [mae2_DG, mae2_DGT, mae2_DGP, mae2_ALL],
        [mre2_DG, mre2_DGT, mre2_DGP, mre2_ALL],
    ]
```

```
# print(res)
outfile = open('result/Performance_evaluation.csv', 'w')
writer = csv.writer(outfile, delimiter=',', quotechar='"')
writer.writerows(res)
outfile.close()
```

The comparison result is shown below:

TABLE 2. PERFORMANCE EVALUATION

			Features Setting			
Year	Model	Error	Demo, Geo	Demo, Geo Taxi	Demo, Geo POI	Demo, Geo POI, Taxi
2013	LR	MAE	0.035001	0.035906	0.038541	0.037178
	LR	MRE	0.306853	0.314788	0.337886	0.325939
	NB	MAE	0.035441	0.036147	0.034202	**0.031129**
	NB	MRE	0.310715	0.316898	0.299848	**0.272905**

From table 2, we can see that when we add POI and Taxi flow into feature set and use NB regression model, the result is the best. And if we just add POI and use NB model, the result is the second best. This may be because POI reflects dynamics of population, it can complement the static residential population in demographics.

Our Conclusion:

- Negative binomial regression model is more appropriate for crime inference than linear regression.
- When POI and taxi flow features both are added to the feature set, the accuracy of estimation will improve a lot.
- In the best scenario, the combination of POI and taxi flow can reduce the estimation error by 11.06%.

Reference:

[1] Wang, Hongjian, et al. "Crime rate inference with big data." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

[2] https://knoema.com/atlas/ranks

[3] https://en.wikipedia.org/wiki/Crime_in_the_United_States

[4] https://en.wikipedia.org/wiki/Crime_in_Chicago

[5] United states census bureau. http://www.census.gov.

[6] https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2

[7] https://developer.foursquare.com/overview/venues.html

`