Contents

A. Read the article and reproduce the results (Accuracy, Precision, Recall, F-Measure) for Chicago Crime dataset using classification method.

- **Same set of features used by the authors:** Based on the paper, "The final attributes considered for this study were named as ID, date, crime primary type, description of the crime, location, year, zip code and police district.". Therefore, to reproduce the results, I used **Date, Primary Type, Description, Location, Year, Zip Codes (column ID contains unique values so no meaning to add into model).**
- **Same classifier with exact parameter values:** As in the paper, **accuracy, precision, recall and f1-score** are the main parameters, I apply **Logistic regression, SVM, Naïve Bayes, MLP, KNN, Random forest, XGBoost** to reproduce closest results based on these parameters.
- **Same training/test splitting approach as used by the authors:** The data is divided into **test sets (30%) and training set (70%)**, so I use the same approach.
- **Same pre/post processing, if any, used by the authors:**
    - Based on the paper, "Initially there were **7019734 crime instances** within the Chicago dataset, and **16913 crimes were removed due to invalid formatting (missing data, fates, values etc.)**.". And "The dataset of Chicago city contains the crime history (reports and social factors) from **2001 to November 2019.**" Due to the reference, "[18] Chicago Data Portal. Accessed: Nov. 2, 2019. [Online]. Available: https://data.cityofchicago.org/Public-Safety/Crimes-2001-topresent-Dashboard/5cd6-ry5g". So, I extracted the data from **1/1/2001 to 1/11/2019**. I got **7013381** instances, and **6893117** instances after preprocessing.
    - "The selection of meaningful data is necessary to eliminate anomalies against the outliers, noise, missing values, and other discrepancies, and thus change over the unfeasible data into possible is manageable to accomplish information handling."
        - Missing values and outliers were removed.
        - Primary Type and Description were encoded after being fixed with spelling errors and heterogeneous.
        - Dates were separated into Day, Month, Year, Hour, Minute.
        - Location was splitted into Latitude and Longitude.
    - "For implementation, Python (version 3.6.3) framework was used with different libraries mainly for data transformation e.g., imblearn and sklearn." As there is no specific use or guide of the libraries, in this reproduction, I used imblearn for RandomUnderSampling and SMOTE (Synthetic Minority Oversampling Technique), along with sklearn in LabelEncoding.

I. Preprocessing

1. Remove "DOMESTIC VIOLENCE" in "Primary Type".

```
data = data[data['Primary Type'] != 'DOMESTIC VIOLENCE']
```

I remove "DOMESTIC VIOLENCE" as there is only 1 record and this is hard to predict.

2. Choose columns based on paper

```
# Choose columns based on paper
data = data[["Date", "Primary Type", "Description", "Location", "Year", "Zip Codes", "Police Districts"]]
```

The authors chose "ID", "Date", "Primary Type", "Description", "Location", "Year", "Zip Codes", "Police Districts" as the parameters using bootstrap random sampling method. However, in my perspective, as the column "ID" contains a unique value of each record, I did not use it.

3. Drop NaN values

```
# Drop NaN rows
data = data.dropna().reset_index().drop(columns=["index"])
```

I dropped NaN values, the same as the authors.

4. Encode "Primary Type"

```
sorted(data['Primary Type'].unique())
```

```
['ARSON',
 'ASSAULT',
 'BATTERY',
 'BURGLARY',
 'CONCEALED CARRY LICENSE VIOLATION',
 'CRIM SEXUAL ASSAULT',
 'CRIMINAL DAMAGE',
 'CRIMINAL SEXUAL ASSAULT',
 'CRIMINAL TRESPASS',
 'DECEPTIVE PRACTICE',
 'GAMBLING',
 'HOMICIDE',
 'HUMAN TRAFFICKING',
 'INTERFERENCE WITH PUBLIC OFFICER',
 'INTIMIDATION',
 'KIDNAPPING',
 'LIQUOR LAW VIOLATION',
 'MOTOR VEHICLE THEFT',
 'NARCOTICS',
 'NON - CRIMINAL',
 'NON-CRIMINAL',
 'NON-CRIMINAL (SUBJECT SPECIFIED)',
 'OBSCENITY',
 'OFFENSE INVOLVING CHILDREN',
 'OTHER NARCOTIC VIOLATION',
 'OTHER OFFENSE',
 'PROSTITUTION',
 'PUBLIC INDECENCY',
 'PUBLIC PEACE VIOLATION',
 'RITUALISM',
 'ROBBERY',
 'SEX OFFENSE',
 'STALKING',
 'THEFT',
 'WEAPONS VIOLATION']
```

Column "Primary Type" has spelling errors, such as:

- 'CRIM SEXUAL ASSAULT' and 'CRIMINAL SEXUAL ASSAULT'.
- 'NON - CRIMINAL' and 'NON-CRIMINAL'.

So I replaced it with same values:

```
data["Primary Type"] = data["Primary Type"].replace('CRIM SEXUAL ASSAULT', 'CRIMINAL SEXUAL ASSAULT')
data["Primary Type"] = data["Primary Type"].replace('NON - CRIMINAL', 'NON-CRIMINAL')
```

After that, I used LabelEncoder to encode the column:

```python
# Label Encoding for Primary Type
le = LabelEncoder()
data["Primary Type"] = le.fit_transform(data["Primary Type"])
```

5. Encode “Description”

```python
sorted(data['Description'].unique())
```

```
['$300 AND UNDER',
 '$500 AND UNDER',
 'ABUSE / NEGLECT - CARE FACILITY',
 'ABUSE/NEGLECT: CARE FACILITY',
 'ADULTRY',
 'AGG CRIM SEX ABUSE FAM MEMBER',
 'AGG CRIMINAL SEXUAL ABUSE',
 'AGG PO HANDS ETC SERIOUS INJ',
 'AGG PO HANDS NO/MIN INJURY',
 'AGG PRO EMP HANDS SERIOUS INJ',
 'AGG PRO.EMP: HANDGUN',
 'AGG PRO.EMP: OTHER DANG WEAPON',
 'AGG PRO.EMP: OTHER FIREARM',
 'AGG PRO.EMP:KNIFE/CUTTING INST',
 'AGG RIT MUT: HANDS/FIST/FEET NO/MINOR INJURY',
 'AGG RIT MUT: HANDS/FIST/FEET SERIOUS INJURY',
 'AGG RITUAL MUT:HANDGUN',
 'AGG RITUAL MUT:KNIFE/CUTTING I',
 'AGG RITUAL MUT:OTH DANG WEAPON',
 'AGG SEX ASSLT OF CHILD FAM MBR',
 'AGG. DOMESTIC BATTERY - HANDS, FISTS, FEET, SERIOUS INJURY',
 'AGG. PROTECTED EMPLOYEE - HANDS, FISTS, FEET, SERIOUS INJURY',
 'AGG: FINANCIAL ID THEFT',
 'AGG: HANDS/FIST/FEET NO/MINOR INJURY',
 'AGG: HANDS/FIST/FEET SERIOUS INJURY',
 'AGGRAVATED',
 'AGGRAVATED - HANDGUN',
 'AGGRAVATED - HANDS, FISTS, FEET, NO / MINOR INJURY',
 'AGGRAVATED - HANDS, FISTS, FEET, SERIOUS INJURY',
 'AGGRAVATED - KNIFE / CUTTING INSTRUMENT',
 'AGGRAVATED - OTHER',
 'AGGRAVATED - OTHER DANGEROUS WEAPON',
 'AGGRAVATED - OTHER FIREARM',
 'AGGRAVATED COMPUTER TAMPERING',
 'AGGRAVATED CRIMINAL SEXUAL ABUSE',
 'AGGRAVATED CRIMINAL SEXUAL ABUSE BY FAMILY MEMBER',
 'AGGRAVATED DOMESTIC BATTERY',
 'AGGRAVATED DOMESTIC BATTERY - HANDGUN',
 'AGGRAVATED DOMESTIC BATTERY - KNIFE / CUTTING INSTRUMENT'
```

Column “Description” has spelling errors, such as: 'SEX OFFENDER - FAIL TO REGISTER NEW ADDRESS' and 'SEX OFFENDER: FAIL REG NEW ADD', etc.
So, I created a csv file, including error column and fixed column and map these to create new column for the dataset.

Then, I encoded the column using LabelEncoding.

```
description = pd.read_csv("description.csv")
merged_df = data.merge(description, on='Description', how='left')
merged_df['Description'] = merged_df.apply(lambda row: row['Alternatives'] if not pd.isna(row['Alternatives']) else row['Description'], axis=1)
merged_df.drop(columns='Alternatives', inplace=True)
data = merged_df
```

```
data["Description"] = le.fit_transform(data["Description"])
data
```

	Date	Primary Type	Description	Location	Year	Zip Codes	Police Districts
0	07/10/2005 03:00:00 PM	2	42	(41.781002663, -87.652107119)	2005	21559.0	17.0
1	08/12/2005 11:00:00 PM	12	237	(41.779492755, -87.605912536)	2005	22260.0	18.0
2	02/01/2019 12:01:00 AM	2	311	(41.802924631, -87.687367104)	2019	22248.0	23.0
3	02/26/2006 04:00:00 PM	31	298	(41.937919992, -87.649024588)	2006	4449.0	5.0
4	08/30/2008 08:15:00 PM	2	311	(41.775891618, -87.75556842)	2008	22268.0	13.0
...	...	...	...	...	...	...	...
6924757	10/16/2019 06:00:00 PM	8	148	(41.845287889, -87.720915275)	2019	21569.0	21.0
6924758	01/01/2019 11:20:00 AM	8	150	(41.967923893, -87.656212265)	2019	22616.0	5.0
6924759	04/09/2019 12:00:00 AM	21	17	(41.73099572, -87.563409251)	2019	21202.0	19.0
6924760	02/01/2019 09:00:00 AM	8	150	(41.951016616, -87.707938347)	2019	21538.0	1.0
6924761	05/06/2018 04:58:00 PM	21	105	(41.851826431, -87.700973609)	2018	21569.0	21.0

6. Split Location into Latitude and Longitude

Location
(41.781002663, -87.652107119)
(41.779492755, -87.605912536)
(41.802924631, -87.687367104)
(41.937919992, -87.649024588)
(41.775891618, -87.75556842)
...
(41.845287889, -87.720915275)
(41.967923893, -87.656212265)
(41.73099572, -87.563409251)
(41.951016616, -87.707938347)
(41.851826431, -87.700973609)

With the format of column “Location” as in image, I splitted it into columns “Latitude” and “Longitude”.

```
data[['Latitude', 'Longitude']] = data['Location'].str.extract(r'\((.*),\s*(.*)\)')

# Convert the new columns to float
data['Latitude'] = data['Latitude'].astype(float)
data['Longitude'] = data['Longitude'].astype(float)
```

7. Process Date

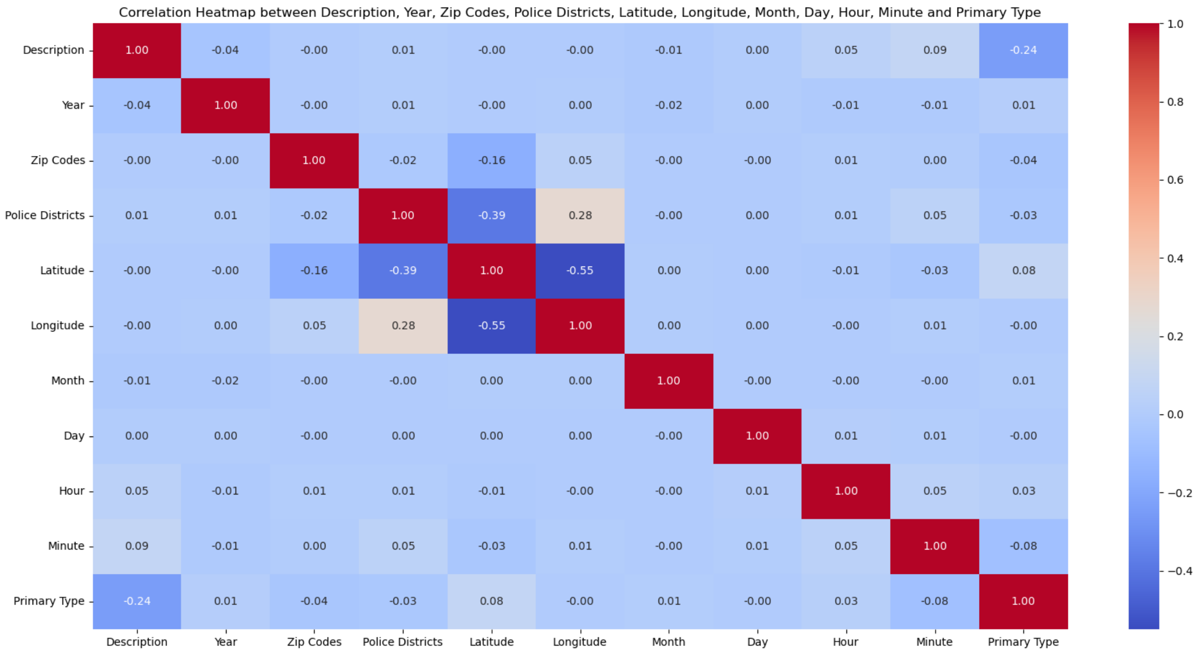
	Date
	07/10/2005 03:00:00 PM
	08/12/2005 11:00:00 PM
	02/01/2019 12:01:00 AM
	02/26/2006 04:00:00 PM
	08/30/2008 08:15:00 PM
	...
	10/16/2019 06:00:00 PM
	01/01/2019 11:20:00 AM
	04/09/2019 12:00:00 AM
	02/01/2019 09:00:00 AM
	05/06/2018 04:58:00 PM

With the format of column “Date” as in image, I splitted it into columns “Year”, “Month”, “Day”, “Hour”, “Minute”.

```
data['Date'] = pd.to_datetime(data['Date'])
```

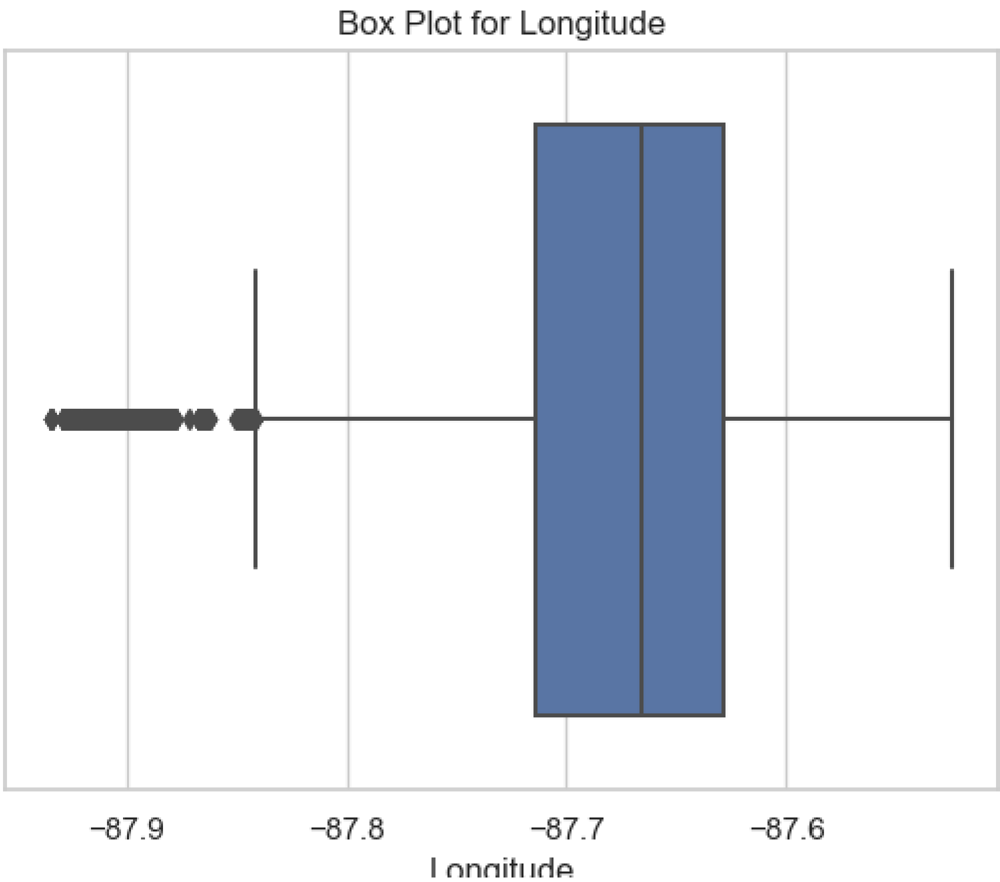
```
data['Year'] = data['Date'].dt.year
data['Month'] = data['Date'].dt.month
data['Day'] = data['Date'].dt.day
data['Hour'] = data['Date'].dt.hour
data['Minute'] = data['Date'].dt.minute
```

8. Check outliers



I checked the correlation of the features with the target, and concluded that column “Description” has the most effect on column “Primary Type”.

Then, I visualized the box plot for Longitude and saw that there are several outliers in the column.



Therefore, I removed those outliers using IQR.

```
# Calculate the IQR
Q1 = data['Longitude'].quantile(0.25)
Q3 = data['Longitude'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers from the 'V17' column and create a new DataFrame without outliers
data = data[(data['Longitude'] >= lower_bound) & (data['Longitude'] <= upper_bound)]

# Optionally, you can reset the index of the new DataFrame
data.reset_index(drop=True, inplace=True)
```

After doing some preprocessing, the dataset contained 6893117 instances.

II. Machine Learning Models

I counted the number of classes in target “Primary Type” and realized this is an imbalance dataset.

```
data['Primary Type'].value_counts()
```

```
THEFT                                  1479906
BATTERY                                1280642
CRIMINAL DAMAGE                         797811
NARCOTICS                               726448
ASSAULT                                 438755
OTHER OFFENSE                           435718
BURGLARY                                397652
MOTOR VEHICLE THEFT                     323139
DECEPTIVE PRACTICE                      285815
ROBBERY                                 263552
CRIMINAL TRESPASS                       200037
WEAPONS VIOLATION                        76783
PROSTITUTION                             68979
PUBLIC PEACE VIOLATION                   49319
OFFENSE INVOLVING CHILDREN               49171
CRIM SEXUAL ASSAULT                      27399
SEX OFFENSE                              26946
INTERFERENCE WITH PUBLIC OFFICER         16637
GAMBLING                                 14563
LIQUOR LAW VIOLATION                     14293
ARSON                                    11511
HOMICIDE                                  9995
KIDNAPPING                                6826
INTIMIDATION                              4109
STALKING                                  3613
CRIMINAL SEXUAL ASSAULT                   2017
OBSCENITY                                  642
CONCEALED CARRY LICENSE VIOLATION          495
PUBLIC INDECENCY                           174
NON-CRIMINAL                               170
OTHER NARCOTIC VIOLATION                   128
HUMAN TRAFFICKING                           65
NON - CRIMINAL                              38
RITUALISM                                   23
NON-CRIMINAL (SUBJECT SPECIFIED)             9
DOMESTIC VIOLENCE                            1
Name: Primary Type, dtype: int64
```

Also, the authors mentioned using imblearn, so I tested for 3 situations: Undersampling, Oversampling and No sampling.

1. Undersampling

a. Undersampling dataset

Undersampling is a technique to balance uneven datasets by keeping all of the data in the minority class and decreasing the size of the majority class. In this situation, as the class with the smallest number of instances has 9 instances, the total number of instances after sampling will be 9 x 33 = 297 instances. For this approach, I used RandomUnderSampler from imblearn.under_sampling.

```python
from imblearn.under_sampling import RandomUnderSampler

# Separate features and target
X = data.drop(columns="Primary Type")
y = data['Primary Type']

# Initialize and fit the RandomUnderSampler
undersampler = RandomUnderSampler(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = undersampler.fit_resample(X, y)
```

```python
X_resampled
```

	Description	Year	Zip Codes	Police Districts	Latitude	Longitude	Month	Day	Hour	Minute
0	97	2007	22618.0	1.0	41.939364	-87.734582	5	11	15	15
1	97	2001	14920.0	15.0	41.854900	-87.669379	12	27	1	10
2	12	2016	22535.0	6.0	41.924534	-87.722580	7	9	5	27
3	97	2014	22257.0	17.0	41.769715	-87.665163	5	28	5	5
4	97	2004	4299.0	6.0	41.906246	-87.729304	5	22	14	56
...	...	...	...	...	...	...	...	...	...	...
292	346	2003	21569.0	21.0	41.856289	-87.712694	3	5	11	21
293	346	2010	22257.0	17.0	41.789649	-87.658403	8	29	1	10
294	357	2001	22535.0	7.0	41.918512	-87.683803	2	7	10	0
295	346	2008	14926.0	14.0	41.904192	-87.647001	9	9	22	0
296	347	2014	4451.0	11.0	41.990345	-87.658226	1	21	10	31

297 rows × 10 columns

b. Models' results using undersampling (in percentage %)

	Logistic Regression	Gaussian Naive Bayes	MLP	KNN	Random Forest	XGBoost	SVM
Accuracy	3.33	30	5.56	12.22	31.11	52.22	1.11
Precision	1.13	36.3	1.17	7.12	28.67	51.06	0.01
Recall	3.33	30	5.56	12.22	31.11	52.22	1.11
F1 - score	1.51	29.72	1.57	8.26	27.46	49.45	0.03

Overall, Undersampling is definitely not the method they used to deal with imbalanced data, as the scores turned out to be really low. This could be a result of low training data for model, as in this situation, we only have 297 instances.

2. Oversampling

a. Oversampling dataset

Oversampling techniques maybe used to duplicate these results for a more balanced amount of positive results in training. In this situation, I used SMOTE (Synthetic Minority Over-sampling Technique), which creates synthetic samples by randomly sampling the characteristics from occurrences in the minority class and got a dataset of 33,439,494 instances.

```
from imblearn.over_sampling import SMOTE
```

```
# Separate features and target
X = data.drop(columns="Primary Type")
y = data['Primary Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y,random_state=42)
```

```
smote = SMOTE(sampling_strategy='auto', k_neighbors=3, random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

```
X_resampled
```

	Description	Year	Zip Codes	Police Districts	Latitude	Longitude	Month	Day	Hour	Minute
0	155	2011	4299.000000	6.000000	41.906611	-87.773086	11	27	18	0
1	311	2004	21869.000000	1.000000	41.968170	-87.734544	12	31	1	50
2	287	2008	22257.000000	17.000000	41.775130	-87.668978	3	28	11	30
3	346	2011	21861.000000	10.000000	41.668187	-87.620581	4	30	16	9
4	339	2018	21190.000000	5.000000	41.929889	-87.652543	10	4	1	30
...	...	...	...	...	...	...	...	...	...	...
33439489	346	2019	21202.000000	19.000000	41.731652	-87.556395	9	5	14	32
33439490	346	2005	4299.000000	16.000000	41.898150	-87.720013	5	27	0	28
33439491	279	2016	21572.000000	16.000000	41.880289	-87.732617	2	12	22	11
33439492	346	2006	4299.129843	15.610472	41.878309	-87.724214	9	17	14	30
33439493	346	2015	22257.000000	17.000000	41.790675	-87.659203	10	14	20	0

33439494 rows × 10 columns

However, due to lack of resources, I cannot run this large dataset, so I chose 10000 instances each class to train, which I had a total of 330000 instances.

```python
from collections import Counter

class_counts = Counter(oversampling_data['Primary Type'])

# Create a balanced subset
subset = pd.DataFrame()

# Set the desired number of rows per class
desired_rows_per_class = 30000

for class_label, count in class_counts.items():
    if count >= desired_rows_per_class:
        # Randomly sample 30,000 rows from this class
        class_subset = oversampling_data[oversampling_data['Primary Type'] == class_label].sample(desired_rows_per_class, random_state=42)
    else:
        # If there are fewer than 30,000 rows for this class, include all of them
        class_subset = oversampling_data[oversampling_data['Primary Type'] == class_label]

    subset = pd.concat([subset, class_subset], ignore_index=True)

# The 'subset' DataFrame now contains 30,000 rows of each class
subset
```

	Description	Year	Zip Codes	Police Districts	Latitude	Longitude	Month	Day	Hour	Minute	Primary Type
0	155	2001	22260.000000	18.000000	41.769422	-87.586392	2	28	17	0	3
1	155	2002	14912.652667	22.000000	41.874964	-87.628646	4	19	3	6	3
2	155	2010	21546.000000	19.831696	41.751218	-87.608204	1	24	11	45	3
3	155	2005	21202.000000	19.000000	41.725525	-87.570492	1	5	1	5	3
4	343	2011	21849.000000	1.000000	41.972314	-87.713474	11	15	9	30	3
...	...	...	...	...	...	...	...	...	...	...	...
989995	231	2012	15767.842178	21.418632	41.788028	-87.667409	10	7	22	27	19
989996	231	2016	20218.149997	13.315540	41.718621	-87.610568	2	13	17	17	19
989997	231	2016	14002.614661	23.306251	41.810060	-87.639195	1	6	16	34	19
989998	231	2014	14921.676391	23.419098	41.824358	-87.645820	1	16	19	25	19
989999	231	2016	11689.732105	21.564787	41.830995	-87.660510	1	6	15	31	19

990000 rows × 11 columns

b. Models' results using Oversampling (in percentage %)

	Logistic Regression	**Gaussian Naive Bayes**	**MLP**	**KNN**	**Random Forest**	**XGBoost**	**SVM**
Accuracy	12.53	12.43	0.66	64.71	40.76	80.71	18.06
Precision	17.61	33.05	0.04	79.15	75.80	90.58	28.89
Recall	12.53	12.43	0.66	64.71	40.77	80.71	18.06
F1 - score	12.26	11.94	0.09	69.22	46.40	83.81	19.22

Overall, using Oversampling resulted in a better score comparing to Undersampling method. Having balanced data for each target class may be a reason why the score became better. However, due to lack of resources, I cannot run all 33 million instances, the scores were not as high as the authors.

3. No sampling

a. No sampling dataset

```
data
```

	Primary Type	Description	Year	Zip Codes	Police Districts	Latitude	Longitude	Month	Day	Hour	Minute
0	2	42	2005	21559.0	17.0	41.781003	-87.652107	7	10	15	0
1	12	237	2005	22260.0	18.0	41.779493	-87.605913	8	12	23	0
2	2	311	2019	22248.0	23.0	41.802925	-87.687367	2	1	0	1
3	31	298	2006	4449.0	5.0	41.937920	-87.649025	2	26	16	0
4	2	311	2008	22268.0	13.0	41.775892	-87.755568	8	30	20	15
...	...	...	...	...	...	...	...	...	...	...	...
6893112	8	148	2019	21569.0	21.0	41.845288	-87.720915	10	16	18	0
6893113	8	150	2019	22616.0	5.0	41.967924	-87.656212	1	1	11	20
6893114	21	17	2019	21202.0	19.0	41.730996	-87.563409	4	9	0	0
6893115	8	150	2019	21538.0	1.0	41.951017	-87.707938	2	1	9	0
6893116	21	105	2018	21569.0	21.0	41.851826	-87.700974	5	6	16	58

6893117 rows × 11 columns

No sampling dataset contains 6,893,117 instances.

b. Models' results using no sampling dataset (in percentage %)

	Logistic Regression	**Gaussian Naive Bayes**	**MLP**	**KNN**	**Random Forest**	**XGBoost**
Accuracy	33.74	33.31	21.00	81.82	66.29	89.31
Precision	19.98	22.43	4.41	81.09	66.35	88.28
Recall	33.74	33.31	21.00	81.82	66.29	89.31
F1 - score	24.38	25.72	7.29	81.16	60.09	97.73

Overall, using No sampling dataset resulted in best scores, as we counted all real current data into the model. However, the scores for Logistic Regression, Naïve Bayes, MLP were still very close. This could be because the dataset is imbalanced.

4. Some suggestions for future improvement

a. Process Longitude and Latitude

Latitude and Longitude are not suitable to use in any machine learning models, as they can have correlation with each other. In normal cases with enough resources, I suggest using them to find the Area of the location, including City, Quarter, Zipcode, State, Municipality, County, Road. Then encode these columns to get numeric values. Below is an example of 10 rows, how I suggest retrieving data.

```python
from geopy.geocoders import Nominatim

def reverse_geocode(latitude, longitude, geolocator):
    location = geolocator.reverse(f"{latitude}, {longitude}", language="en", exactly_one=True)
    if location:
        return location.raw['address']
    else:
        return None

# Initialize the geolocator
geolocator = Nominatim(user_agent="reverse_geocode")

# Apply reverse geocoding to each row and create new columns for city, quarter, and zipcode
location_10['address'] = location_10.apply(lambda row: reverse_geocode(row['Latitude'], row['Longitude'], geolocator), axis=1)

# Extract and create new columns for city, quarter (if available), and zipcode
location_10['City'] = location_10['address'].apply(lambda x: x.get('city', ''))
location_10['Quarter'] = location_10['address'].apply(lambda x: x.get('quarter', ''))
location_10['Zipcode'] = location_10['address'].apply(lambda x: x.get('postcode', ''))
location_10['State'] = location_10['address'].apply(lambda x: x.get('state', ''))
location_10['Municipality'] = location_10['address'].apply(lambda x: x.get('municipality', ''))
location_10['County'] = location_10['address'].apply(lambda x: x.get('county', ''))
location_10['Road'] = location_10['address'].apply(lambda x: x.get('road', ''))

location_10
```

	Latitude	Longitude	address	City	Quarter	Zipcode	State	Municipality	County	Road
0	41.781003	-87.652107	{'house_number': '6211', 'road': 'South Aberde...	Chicago	Englewood	60620	Illinois	Lake Township	Cook County	South Aberdeen Street
1	41.779493	-87.605913	{'building': 'Strand Hotel', 'house_number': '...	Chicago	Woodlawn	60637	Illinois	Hyde Park Township	Cook County	South Cottage Grove Avenue
2	41.802925	-87.687367	{'house_number': '2512', 'road': 'West 50th St...	Chicago	Gage Park	60632	Illinois	Lake Township	Cook County	West 50th Street
3	41.937920	-87.649025	{'house_number': '3101', 'road': 'North Clark ...	Chicago	Lake View	60657	Illinois	Lake View Township	Cook County	North Clark Street
4	41.775892	-87.755568	{'house_number': '6412', 'road': 'South Lorel ...	Chicago	Clearing	60638	Illinois	Lake Township	Cook County	South Lorel Avenue
5	41.974346	-87.656361	{'historic': 'West Argyle Street Historic Dist...	Chicago	Uptown	60640	Illinois	Lake View Township	Cook County	West Carmen Avenue
6	41.865214	-87.727590	{'house_number': '1233', 'road': 'South Karlov...	Chicago	North Lawndale	60623	Illinois	West Chicago Township	Cook County	South Karlov Avenue
7	41.749500	-87.601157	{'house_number': '936-942', 'road': 'East 80th...	Chicago	Chatham	60619	Illinois	Hyde Park Township	Cook County	East 80th Street
8	41.896215	-87.728572	{'house_number': '826', 'road': 'North Karlov ...	Chicago	Humboldt Park	60639	Illinois	West Chicago Township	Cook County	North Karlov Avenue
9	41.911574	-87.789972	{'house_number': '1723', 'road': 'North Nashvi...	Chicago	Austin	60707	Illinois	Jefferson Township	Cook County	North Nashville Avenue

In this report, due to lack of resources, I cannot apply the method. Requesting API too much can be blocked and below is the error after running 1000 rows.

```
Cell In[48], line 14, in <lambda>(row)
     11 geolocator = Nominatim(user_agent="reverse_geocode")
     13 # Apply reverse geocoding to each row and create new columns for city, quarter, and zipcode
---> 14 location['address'] = location.apply(lambda row: reverse_geocode(row['Latitude'], row['Longitude'], geolocator), axis=1)
     16 # Extract and create new columns for city, quarter (if available), and zipcode
     17 location['City'] = location['address'].apply(lambda x: x.get('city', ''))

Cell In[48], line 4, in reverse_geocode(latitude, longitude, geolocator)
      3 def reverse_geocode(latitude, longitude, geolocator):
----> 4     location = geolocator.reverse(f"{latitude}, {longitude}", language="en", exactly_one=True)
      5     if location:
      6         return location.raw['address']

File ~\anaconda3\Lib\site-packages\geopy\geocoders\nominatim.py:372, in Nominatim.reverse(self, query, exactly_one, timeout, language, addressdetails, zoom, namedetails)
    370 logger.debug("%s.reverse: %s", self.__class__.__name__, url)
    371 callback = partial(self._parse_json, exactly_one=exactly_one)
--> 372 return self._call_geocoder(url, callback, timeout=timeout)

File ~\anaconda3\Lib\site-packages\geopy\geocoders\base.py:368, in Geocoder._call_geocoder(self, url, callback, timeout, is_json, headers)
    366 try:
    367     if is_json:
--> 368         result = self.adapter.get_json(url, timeout=timeout, headers=req_headers)
    369     else:
    370         result = self.adapter.get_text(url, timeout=timeout, headers=req_headers)

File ~\anaconda3\Lib\site-packages\geopy\adapters.py:472, in RequestsAdapter.get_json(self, url, timeout, headers)
    471 def get_json(self, url, *, timeout, headers):
--> 472     resp = self._request(url, timeout=timeout, headers=headers)
    473     try:
    474         return resp.json()

File ~\anaconda3\Lib\site-packages\geopy\adapters.py:494, in RequestsAdapter._request(self, url, timeout, headers)
    492         raise GeocoderServiceError(message)
    493     else:
--> 494         raise GeocoderUnavailable(message)
    495 elif isinstance(error, requests.Timeout):
    496     raise GeocoderTimedOut("Service timed out")

GeocoderUnavailable: HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Max retries exceeded with url: /reverse?lat=41.900037654&lon=-87.717685239&format=json&accept-
language=en&addressdetails=1 (Caused by ReadTimeoutError("HTTPSConnectionPool(host='nominatim.openstreetmap.org', port=443): Read timed out. (read timeout=1)"))
```

Instead, I checked the correlation between Latitude and Longitude, and they did not have correlation with each other, so I still put them in the models.

b. Using GridSearchCV to get optimized hyperparameters.

GridSearchCV is a technique for hyperparameter tuning in machine learning that exhaustively searches over a predefined hyperparameter grid to find the best combination of hyperparameters for a given model and dataset. However, it can be computationally expensive, especially for large hyperparameter grids and large datasets.

c. Apply Oversampling to get balanced dataset.

The primary advantage of using oversampling is that it helps improve the predictive performance of machine learning models in such imbalanced scenarios. By increasing the number of instances in the minority class, the algorithm will have more examples to learn from, which can help it better distinguish between the classes. Also, unlike some Undersampling techniques (which reduce the size of the majority class), Oversampling retains all the original instances of the majority class. This means losing potentially valuable information from the majority class will not happen.

B. Design and develop your own ML solution for this problem.

I. Motivation behind the proposed solution.

Chicago has faced persistent issues with crime over the years, and there is a pressing need to reduce crime rates for the safety and well-being of its residents. In the era of big data, law enforcement agencies can benefit from harnessing the power of data to make informed decisions. Machine learning models can process vast amounts of historical and real-time data to identify patterns and trends that might not be apparent through traditional analysis methods. In response to this pressing concern, this proposal seeks to introduce a groundbreaking machine learning solution designed to revolutionize crime prediction and prevention within the city of Chicago.

II. How the proposed solution is different from existing ones.

To gain a comprehensive understanding of how my proposed crime prediction system distinguishes itself, it is essential to first survey the existing landscape of crime prediction solutions in the city of Chicago.

Stec et al. took advantage of deep neural networks to make next day crime count predictions in a fine-grain city partition. This paper also shows the value of using external datasets in addition to standard crime data. They made predictions using Chicago crime data covering with additional datasets such as weather, census data, and public transportations.

Kang et al. proposed a feature-level data fusion method with environmental context based on a deep neural network (DNN). In order to enhance crime prediction models, they considered environmental context information, such as broken windows theory and crime prevention through environmental design. Prior to generating training data, they selected crime-related data by conducting statistical analyses.

Aldossari et al. applied Decision Tree and Naïve Bayes on the Chicago Police Department's Citizen Law Enforcement Analysis and Reporting system dataset to predict the potential crime category in a particular geographic area. Also, they manually reduced the dataset records to achieve a balanced distribution of the class labels.

Using the same datasets, Yuki et al. proposed a classification of which category of crime is most probably to take place at a detailed time and places in Chicago. This paper uses a different algorithm like Random Forest, Decision Tree and different ensemble methods such as Extra Trees, Bagging and AdaBoost to evaluate the accuracy given by each algorithm

Safat et al. applied different machine learning algorithms, namely, the logistic regression, support vector machine (SVM), Naïve Bayes, KNN, Decision Tree, Multilayer Perceptron, Random Forest, and XGBoost model to better fit the crime data.

My proposed solution is different from the existing ones among preprocessing and implementing the model. I used SelectKBest to find top 4 features that should be useful for model. Then, I applied XGBoost on the cleaned data.

III. Detail description of the model including all parameters so that any reader can implement your model.

Before training the data, SelectKBest with chi-squared (chi2) statistical test was applied to choose appropriate features for the model. Chi-squared is used to determine the independence between categorical variables, making it a suitable choice for feature selection when dealing with categorical data. Data was scaled, then applied to SelectKBest, choosing top 5 features. Finally, top 6 features were 'Description', 'Location Description', 'Arrest', 'Domestic', 'Hour', 'Minute'.

I applied XGBoost, with train/test ratio of 0.2, on these features with several hyperparameters: **n_estimators = 100, max_depth = 3, learning_rate = 0.1, random_state = 42**. n_estimators determines the number of boosting rounds or decision trees that will be used in the XGBoost ensemble. In this case, I set it to 100, meaning my XGBoost classifier will consist of 100 decision trees. max_depth controls the maximum depth of each decision tree in the ensemble. A lower value, such as 3 in my model, creates shallow trees, which can help prevent overfitting and improve model generalization. The learning rate (or shrinkage) controls the step size at each iteration while moving toward a minimum of the loss function. A smaller learning rate requires more boosting rounds to converge but may result in a more accurate model. For my model, I chose 0.1, a common value. random_state sets the random seed for reproducibility. It ensures that my results will be consistent if I run the same code multiple times. For my model, I set it at 42.

IV. Description of experimental protocol.

The experimental protocol consisted of formatting data, data extraction, quality check, multiclass classification, and results evaluation. Data formatting involved several steps. All the meaningless columns, which are 'ID', 'Case Number', 'Block', 'IUCR', 'FBI Code', 'X Coordinate', 'Y Coordinate', 'Year', 'Updated On', 'Location' were dropped firstly. These columns either have unique values each row or does not have any meaning to predict the target. All missing values were also dropped, same as the authors. Then, column "Primary Type" and "Description" were encoded, along with column "Date" into column "Year", "Month", "Day", "Hour", "Minute" being splitted, same as question 1. There appeared some spelling errors in column "Location Description", so I created a csv file including error column and fixed column, then merge to create the correct column in the dataset. Column "Arrest" and "Domestic" were also encoded as they were binary columns. Finally, I chose SelectKBest to choose appropriate features for the model. The selected features are 'Description', 'Location Description', 'Arrest', 'Domestic', 'Hour', 'Minute'.

V. Evaluation metrics.

To easily compare the results with the existed paper, I also chose "Accuracy", "Precision", "Recall", "F1 Score" to evaluate the model.

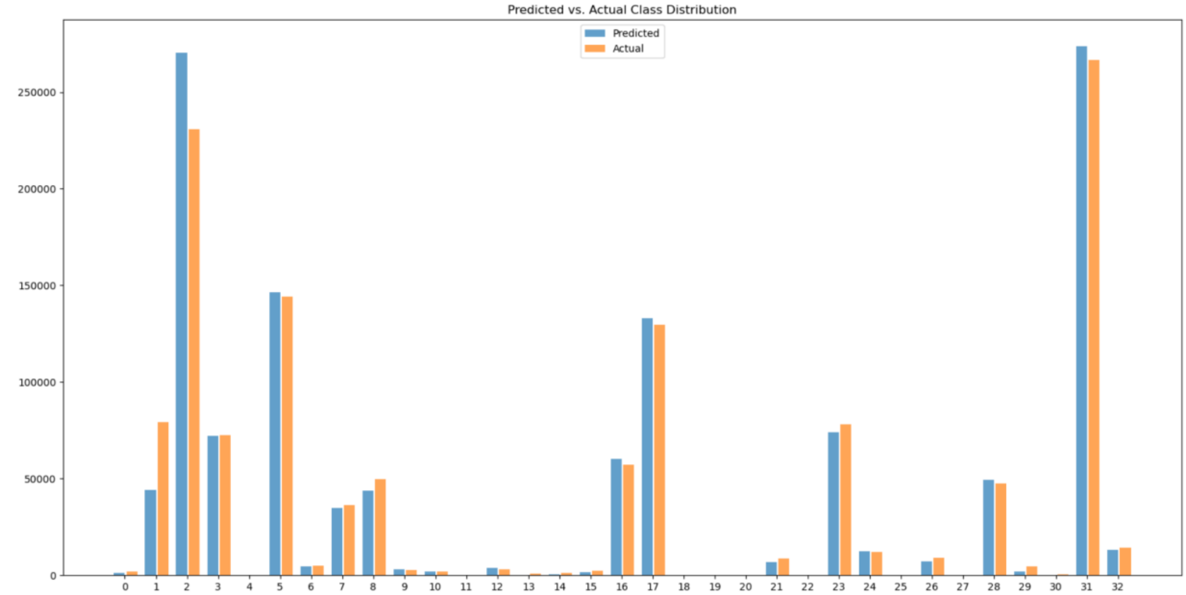
- Accuracy: The proportion of correctly predicted or classified instances out of the total.
- Precision: The ratio of true positive predictions to the total positive predictions (true positives + false positives).

- Recall: The ratio of true positive predictions to the total actual positives (true positives + false negatives).
- F1 Score: The harmonic mean of precision and recall, providing a balanced measure of both metrics.

VI. Present results using tables and graphs.

Accuracy	Precision (weighted)	Recall (weighted)	F1 – score (weighted)
91.23%	90.69%	91.23%	90.49%

The accuracy, precision, recall and F1 scores of the fitted model was calculated. With the accuracy of 91.23%, it means that the model correctly predicted the target variable in 91.23% of the cases. The weighted precision is 90.69%, indicating that, on average, when the model predicts a positive class, it is correct about 90.69% of the time. A weighted recall suggests that the model correctly identified 91.23% of all positive cases in the dataset. Lastly, A weighted F1 score of 90.49% indicates a good balance between precision and recall for your model. Overall, these metrics collectively provide a picture of how well the XGBoost model performed.



The chart showed how well the model predicted for each class. Overall, XGBoost worked quite well for most classes, except class 1 and 2. This could be due to the heterogeneous data. To solve this problem, oversampling or manually adding data can reduce the error in predicting.

VII. Compare and discuss results with respect to existing literatures.

Yuki et al. 's paper accuracies were 95.99% for Random Forest, 99.88% for Decision Tree, 74.78% for AdaBoost, 99.92% for Bagging and 97.10% for Extra Tree. Highest accuracy is acquired with the implement of Bagging because ensemble method combines several tree classifiers and gives much better predictive results, AdaBoost generally works best for binary classification, resulting in the lowest accuracy.

Meanwhile, Aldossari et al. applied backward feature selection and run models for each features subset. Overall, Decision Tree classifier gives better accuracy than Naïve Bayes with 9 features and same accuracy was achieved with all the features, at 91.68%.

Using 8 different algorithms to investigate the detailed predictive accuracy of the trained models, the paper shows that XGBoost performs better than other algorithms with 94% for Chicago dataset, as multiple innovative algorithms work behind XGBoost.

My proposed solution resulted in 91.23% of accuracy, with high scores for precision, recall and F1 – score. However, as the results are still not as high as the other papers, improvement should be made further.

VIII. Appropriate references

- Aldossari, BS, Alqahtani, FM, Alshahrani, NS, Alhammam, MM, Alzamanan, RM, Aslam, N & Irfanullah 2020, 'A Comparative Study of Decision Tree and Naive Bayes Machine Learning Model for Crime Category Prediction in Chicago',.

- Kang, H-W & Kang, H-B 2017, 'Prediction of crime occurrence from multi-modal data using deep learning' K-KR Choo (ed), *PLOS ONE*, vol. 12, no. 4, p. e0176244.

- Safat, W, Asghar, S & Gillani, SA 2021, 'Empirical Analysis for Crime Prediction and Forecasting Using Machine Learning and Deep Learning Techniques', *IEEE Access*, pp. 1–1.

- Stec, A & Klabjan, D 2018, 'Forecasting Crime with Deep Learning', *arXiv:1806.01486 [cs, stat]*, retrieved from <https://arxiv.org/abs/1806.01486>.

- Yuki, JQ, Sakib, MdMQ, Zamal, Z, Habibullah, KM & Das, AK 2019, 'Predicting Crime Using Time and Location Data', *Proceedings of the 2019 7th International Conference on Computer and Communications Management*.