Alma Mater Studiorum - Università di Bologna
Master's Degree in Artificial Intelligence

San Francisco Crime Classification

Big Data Analytics Project and Project work

Andrea Valente

Academic Year 2022-2023

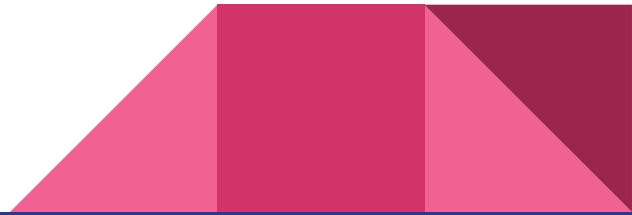
Project task

The **San Francisco Crime Classification** is a Kaggle competition where the goal is to predict the category of crimes that occurred in the city by the bay.

The dataset contains information about the crimes that occurred in San Francisco between 2003 and 2015, such as the date, time, location, description, resolution, etc.

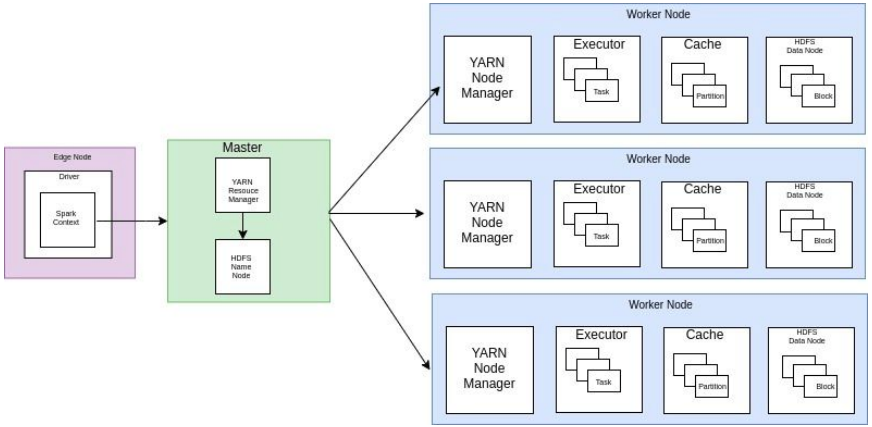
In total, there are **39 categories** of crimes, **8 features** and **878049 samples**, a mix of continuous and categorical features, but some are redundant (e.g. *Address* with *X* and *Y* coordinates) and some are not useful for the task (e.g. *Descript* and *Resolution*).

The goal of this project is to use the **Hadoop and Spark frameworks** to preprocess the data and train a model to predict the category of the crimes.



Hadoop & Spark Cluster architecture

- **Architecture**: 3-node cluster (one master and two slaves), each running on a different VM setup with Vagrant with 4GB of RAM
- **Hadoop**: a framework for distributed storage and processing of large data sets
- **Spark**: a unified analytics engine for large-scale data processing
- Spark runs on the master node and connects to the **HDFS** and the **YARN Resource Manager** to distribute the Spark jobs
- **PySpark**: the Python API for Spark that allows to write Spark jobs in Python



Source:
https://blog.knoldus.com/install-configure-hadoop-hdfsyarn-cluster-and-integrate-spark-with-it/

Dataset features

Features	Type	Unique	Null	Description
Dates	timestamp	389,257	0	Time of the crime
Descript	string	879	0	Detailed description of the crime *(train.csv only)*
DayOfWeek	string	7	0	Day of the week
PdDistrict	string	10	0	Name of the police department district
Resolution	string	17	0	How the crime was resolved *(train.csv only)*
Address	string	23,228	0	Approximate street address of the crime
X	double	34,243	0	Longitude
Y	double	34,243	0	Latitude
Category	string	39	0	**Target variable**, the category of the crime incident

Data analysis

There are some features that are not useful for the task, such as **Descript** and **Resolution** (not available at the time of the crime). Additionally, **Address** is redundant with **X** and **Y** coordinates, so we will drop it.

Additionally, we will create new features from **Dates**, such as **year**, **month** and **hour**, since they might be useful for the task.

Since no feature has null values, we don't need to impute or drop any rows. At the end of the preprocessing, we will have **8 features** and **878049 samples**.

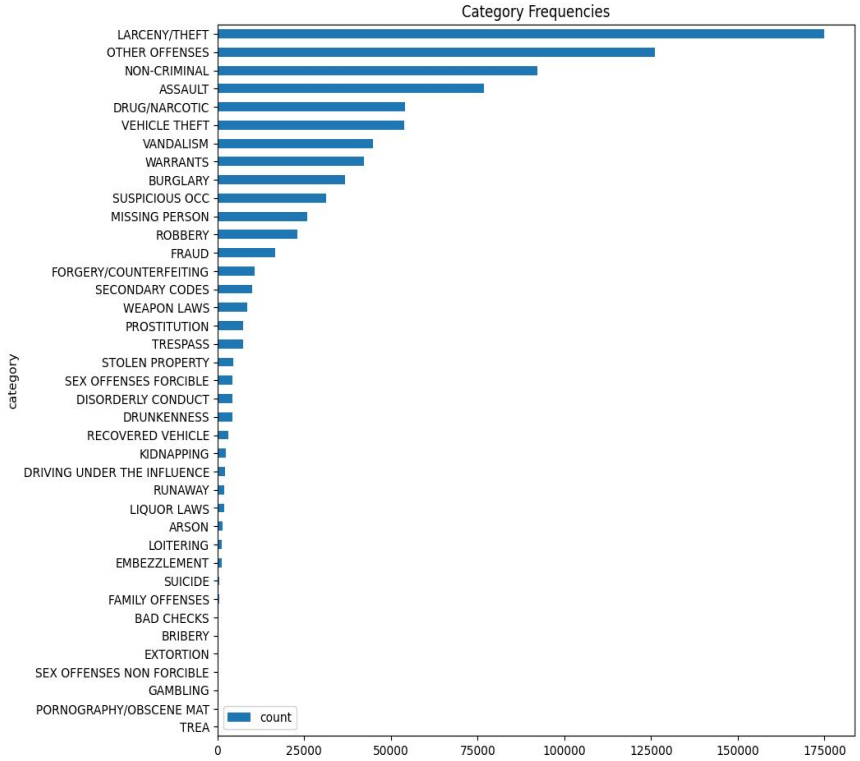
Category	DayOfWeek	PdDistrict	X	Y	Year	Month	Hour
Suspicious Occ	Saturday	Richmond	-122.4903	37.7759	2011	12	5
Assault	Saturday	Ingleside	-122.4410	37.7164	2010	9	21

Label analysis

The **distribution** of the categories is **skewed**: some categories having a lot more samples than others, with the **top-6 crimes** which cover almost 66% of the dataset.

The **most frequent** category is **LARCENY/THEFT**, which is almost 20% of the dataset.

The **least frequent** category is **TREA**, which is less than 0.01% of the dataset.



PdDistrict analysis

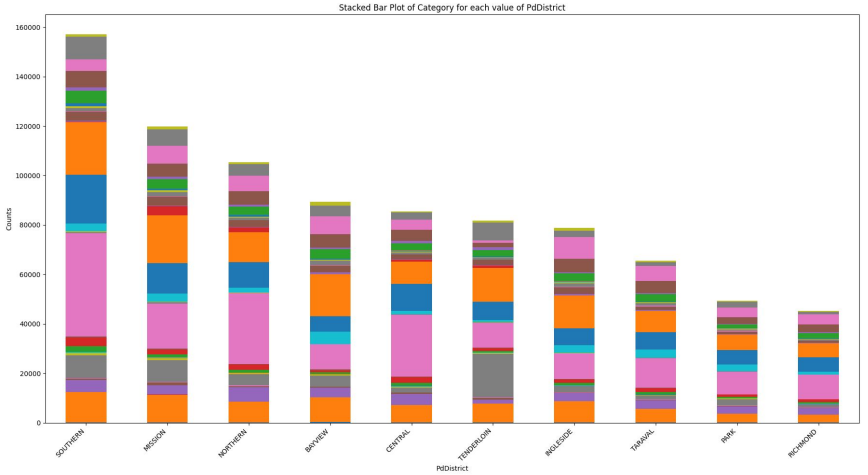
The **PdDistrict** feature has 10 unique values, which are the names of the police department districts.

The **most frequent** is **SOUTHERN**, which is almost 18% of the dataset, while the **least frequent** is **RICHMOND**, which is slightly more than 5% of the dataset.

The distribution of the crimes in the districts is not uniform, with some districts having a lot more crimes of a certain category than others.

For example, the most frequent category in SOUTHERN is **LARCENY/THEFT**, while the most frequent category in TENDERLOIN is **DRUG/NARCOTIC**.

However, in most districts, the most frequent category is LARCENY/THEFT (pink bar).



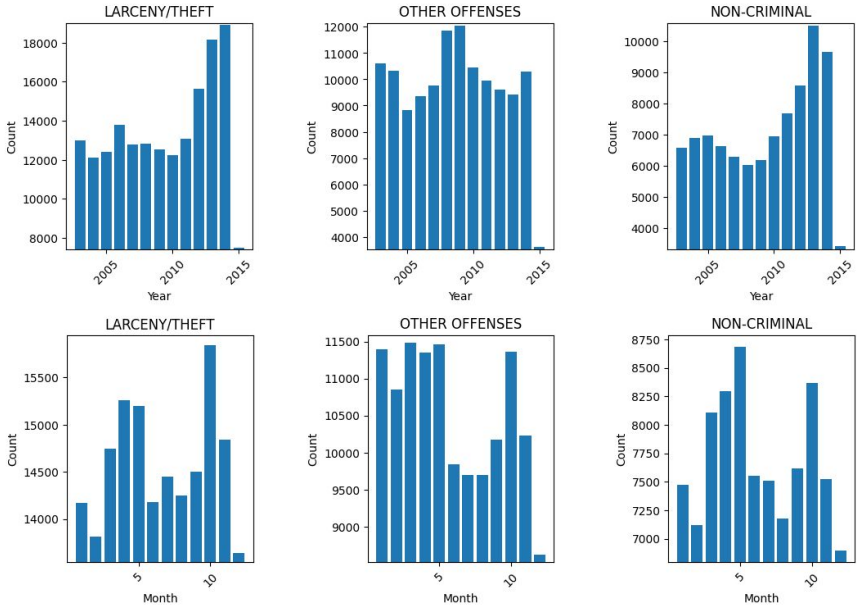
Year and month analysis

The crimes are distributed over 12 years, from 2003 to 2015. The most frequent year is 2013, while the least frequent year is 2015 since the dataset contains only data until May.

The distribution of the distinct crimes over the year and the month is not uniform.

For example the number of crimes of the category **LARCENY/THEFT** has started to increase since 2012, while the number of crimes of the category **VEHICLE THEFT** has dropped since 2005.

The same can be said for the months, with ups and downs in the number of crimes of the different categories.



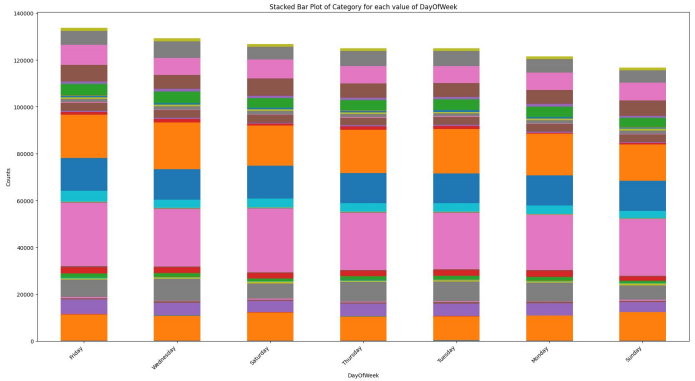
DayOfWeek and Hour analysis

The crimes are distributed over 7 days, from Monday to Sunday.

The most frequent day is **Friday**, while the least frequent day is **Sunday**, however there are some categories that are more frequent during the **weekend**, such as **ASSAULT**.

The crimes are distributed over 24 hours, from 0 to 23. The most frequent hour is **18**, while the least frequent hour is **5**.

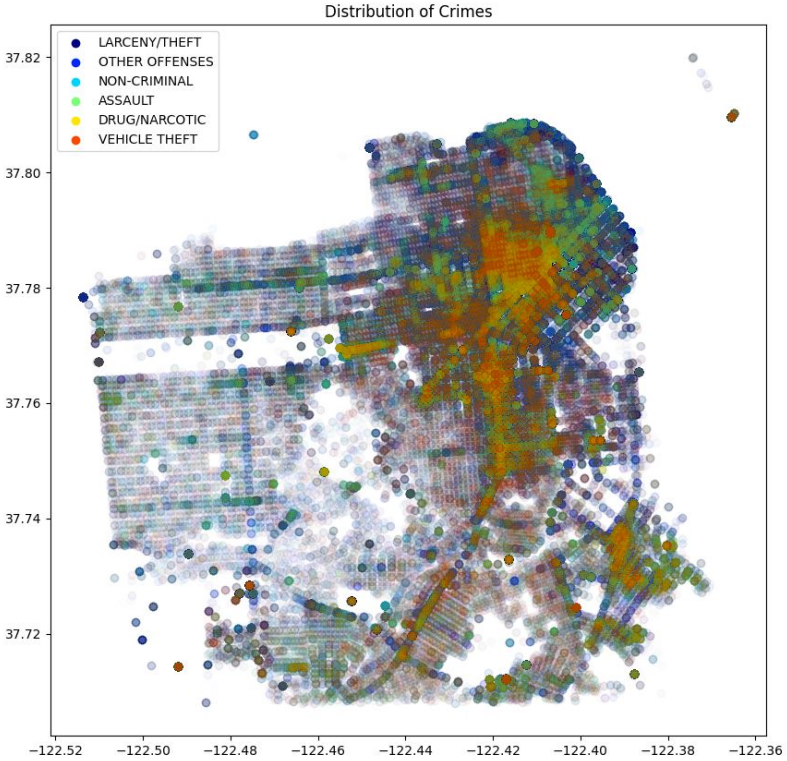
The number of crimes is higher during the day and lower during the night, however there are some categories that are more **frequent during the night**, such as **VEHICLE THEFT** and **VANDALISM**.



DayOfWeek and Hour analysis

The **X and Y coordinates** are distributed over San Francisco, with some areas having a lot more crimes than others. The most frequent area is the one around the coordinates (-122.4, 37.8) (upper left corner), while the least frequent area is in the west side of San Francisco.

There are **67 values** for X and Y that are **outside of San Francisco** (probably wrong values), so we will safely remove them since they are a very small percentage of the dataset.

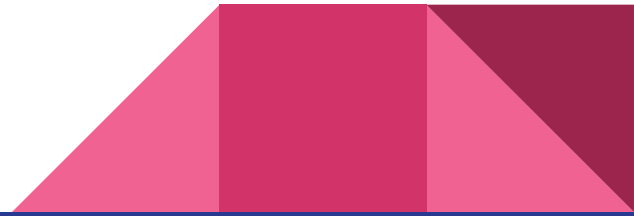


Feature engineering

We will encode the **categorical features** using **StringIndexer** and **OneHotEncoder**, while we will encode the X and Y coordinates using either **GridEncoder** (custom transformer) or **Clustering** (KMeans).

The main **difference** between StringIndexer and OneHotEncoder is that **StringIndexer** will assign a number to each category, while **OneHotEncoder** will create a new feature for each category and assign 1 to the category of the sample and 0 to the other categories.

The main **difference** between GridEncoder and Clustering is that **GridEncoder** will divide the map of San Francisco into a grid of equally sized cells and assign a number to each cell, while **Clustering** will divide the map of San Francisco into clusters and assign a number to each cluster.



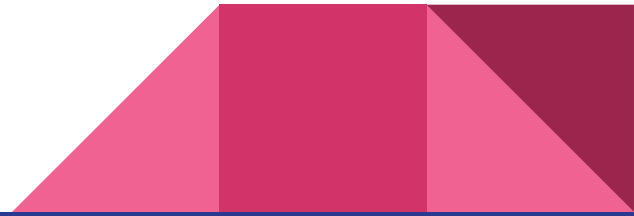
Model selection and evaluation

For the machine learning models, several models will be trained and compared, such as **Logistic Regression**, **Random Forest**, **Naive Bayes** and **Neural Network**.

The comparison will be done using the **multiclass log loss** and the **F1 score** of the models trained on the same set of features.

Additionally, the models will be trained using **incremental feature engineering**, i.e. adding new features one by one and comparing the results, in order to see which features are the most useful for the task.

The task training dataset will be split into a **training set** (60%), **validation set** (20%) and **test set** (20%), while the task test dataset won't be used since it doesn't contain the labels (available only during the Kaggle competition)



Training results - Baseline

The baseline model is a **Logistic Regression** model trained on the categorical features *DayOfWeek* and *PdDistrict*, which are encoded using **StringIndexer**. Then, the model is compared to **Naive Bayes** and **Random Forest**, which are trained on the same features.

Finally, these results are **compared** to the results of the same models trained on the categorical features encoded using **OneHotEncoder**.

The results show that the models trained on the categorical features encoded using **OneHotEncoder** perform better.

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Logistic Regression	0.0662	0.0663	2.6711	2.6727
Naive Bayes	0.0679	0.0680	2.6771	2.6784
Random Forest	**0.1217**	**0.1227**	**2.6222**	**2.6241**

StringIndexer encoding

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Logistic Regression	**0.1225**	**0.1236**	**2.6107**	**2.6129**
Naive Bayes	0.1224	0.1232	2.6149	2.6168
Random Forest	0.1085	0.1093	2.6278	2.6292

OneHotEncoder encoding

Training results - Dates features

After adding the new features *year*, *month* and *hour*, the models are trained again and compared to the previous results.

The results show that the models trained on the new features **perform better**, except for the Random Forest model, which performs slightly worse.

The **Random Forest** model worsens its performance due to **limitations** on the cluster: increasing the depth of the trees results in **out of memory errors**.

However, in a different setup, the Random Forest model would perform better.

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Logistic Regression	**0.1428**	**0.1444**	**2.5559**	**2.5605**
Naive Bayes	0.1417	0.1428	2.5627	2.5667
Random Forest	0.0826	0.0827	2.6279	2.6305

Year, Month, Hour features - OneHotEncoder encoding

Training results - X and Y features

The X and Y coordinates are encoded using two different methods: **GridEncoder** and **Clustering**.

The **GridEncoder** divides the map into a **grid of equally sized cells** and assigns a number to each cell (e.g. *8x8 grid, 64 cells*). Then, the X and Y coordinates are mapped to the cell they belong to.

The Clustering uses the **KMeans** algorithm (after appropriate **scaling** of the coordinates) and assigns a number to each cluster. The clusters are evaluated using the *silhouette score* (e.g. K=3)

The results show that the models trained on the X and Y coordinates encoded using **GridEncoder** perform better than the ones trained on the X and Y coordinates encoded using Clustering.

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Logistic Regression	0.1606	0.1599	2.5153	2.5242
Naive Bayes	**0.1610**	**0.1614**	**2.5623**	**2.5696**

XY features - GridEncoder encoding

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Logistic Regression	0.1455	0.1461	2.5539	2.5587
Naive Bayes	**0.1479**	**0.1484**	**2.5700**	**2.5738**

XY features - KMeans encoding

Training results - Neural network

The **Neural Network** model is trained on similar features, but rather than encoding the *X* and *Y* coordinates using GridEncoder, they are with **StandardScaler**, which scales the coordinates to have *mean 0* and *variance 1*.

The results show that the **Neural Network** model **performs worse** than the other models, however a different setup might be able to **improve its performance** (e.g. more layers, more neurons, different activation functions, etc.).

The model is **compared** to a **Random Forest** model trained on the same features, which **performs worse** for the same reasons as before.

Models	Train F1	Test F1	Train Log Loss	Test Log Loss
Neural network	**0.1467**	**0.1479**	**2.5847**	**2.5868**
Random Forest	0.1101	0.1107	2.6082	2.6108

Year, Month, Hour, X, Y features - OneHotEncoder and StandardScaler

Conclusion

In conclusion, the best model is the **Naive Bayes** model trained on the categorical features encoded using **OneHotEncoder** and the X and Y coordinates encoded using **GridEncoder**, which achieves a **multiclass log loss of 2.56** and an **F1 score of 0.16**.

The results show that the models **benefit from the new features**, such as year, month and hour, and the encoding of the X and Y coordinates.

In addition, the **OneHotEncoder** performs better than the StringIndexer, since it **does not introduce any order** between the categories, which is not useful for the task.

Due to **limitations** on the cluster, the **Random Forest** model performs worse than the other models, however in a different setup it would perform better, as shown by some **results** obtained in the **Kaggle competition**.