Big Data Analytics Exam and Project Work

San Francisco Crime Classification

Chiara Angileri, Giorgia Pavani, Livia Del Gaudio

Aim of the Project

The developement of this project involves exploring Big Data Analytics techniques for the analysis of San Francisco Crime dataset using the Pyspark library, following the methodology outlined by Abouelnaga.

The aim of this project is to perform **multi-class classification** to identify different categories of crime.

Using a set of features, the goal is to predict the corresponding category of the given crime.

Architecture

Apache Spark is a robust and adaptable distributed computing framework specifically designed for efficient and speedy processing and analysis of large-scale data.

Its in-memory computing engine allows for real-time data processing and iterative data analysis, making it well-suited for applications in **big data** analytics and **machine learning**.

In our project we used Apache Spark cluster built on Docker. Subsequently, we created a Spark session within a Jupyter Notebook.

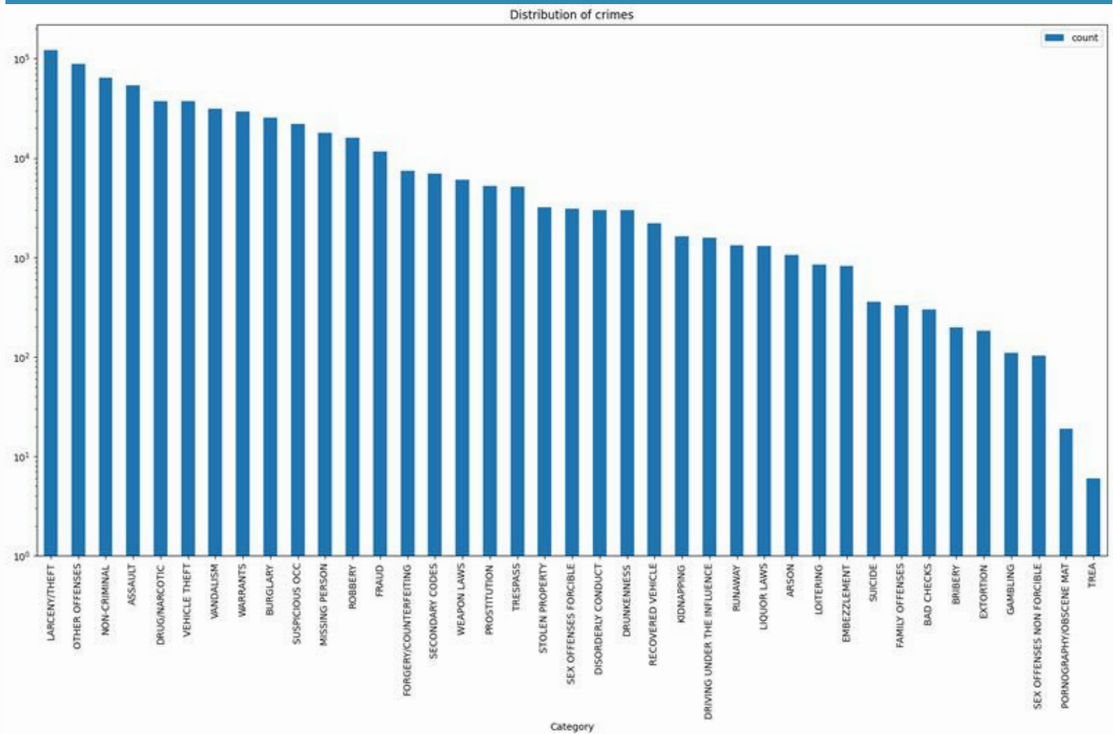
Dataset

The original dataset contains the following features:

- *Dates*: timestamp of the crime incident.
- *Category*: category of the crime incident. **Target variable** we are going to predict.
- *Descript*: detailed description of the crime incident.
- *DayOfWeek*: the day of the week.
- *PdDistrict*: name of the Police Department District.
- *Resolution*: how the crime incident was resolved.
- *Address:* the approximate street address of the crime.
- *X*: longitude.
- *Y*: latitude.

Dataset requirements:
9 features x 878.050 instances

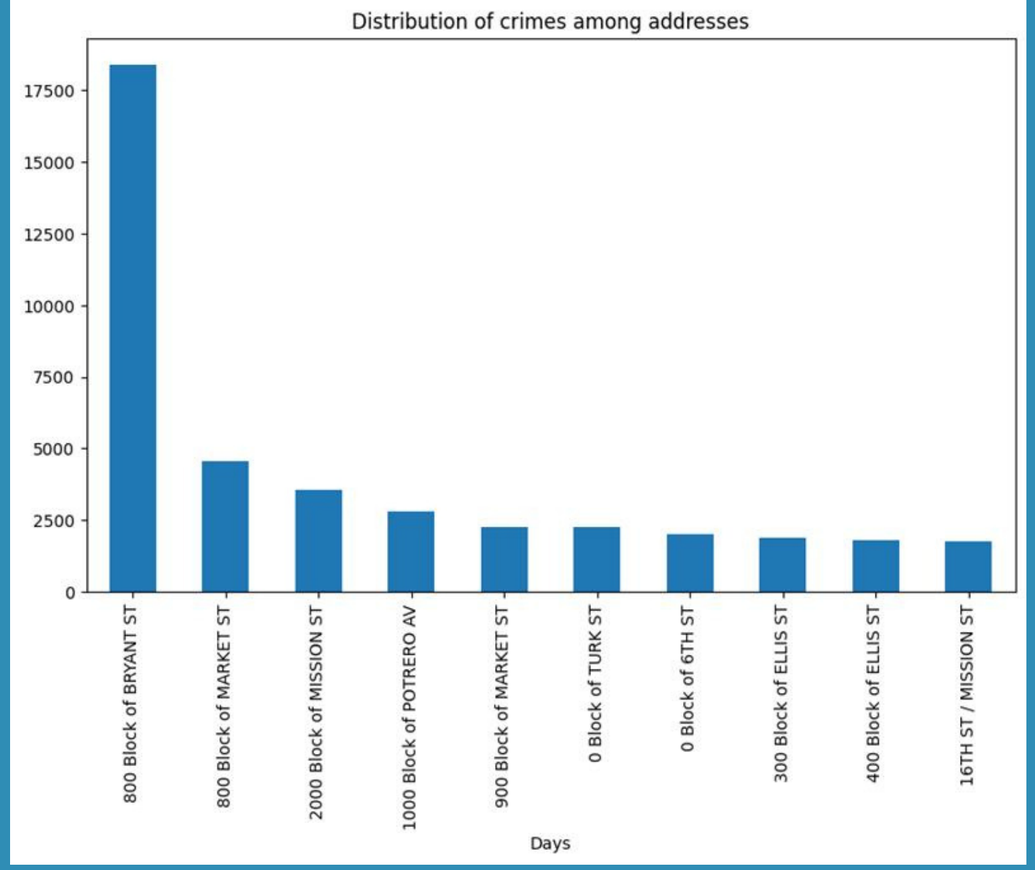
= 7.902.450

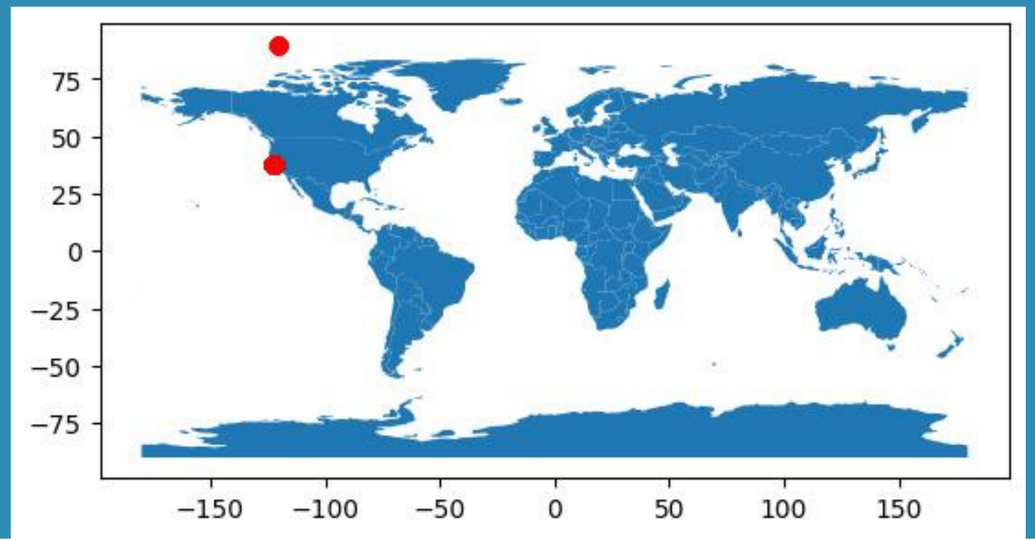


Dataset Preprocessing (1/2)

The majority of crimes occur in addresses containing the word '**Block**', so we take the decision to extract this information as an additional binary feature.

We notice some incorrect coordinates that don't match San Francisco's geographic location. These might be due to mistakes in labeling, so we treat them as **outliers** and exclude them from the dataset.





Dataset Preprocessing (2/2)

Further steps done in preprocessing:

- **Feature extraction**: split *Dates* column by selecting *Month*, *Hour* and *Year* informations.
- **Normalize coordinates**: normalize *X* and *Y* coordinates using *MinMaxScaler* to transform the values between 0 and 1.
- **Drop rows** with **nulls** and **duplicate** rows.
- **Drop unused columns**: drop the *address* column since we already have the coordinates and the *Block* feature.
- **Convert columns to categorical**: *Resolution*, *PdDistrict*, *Category*, and *DayOfWeek* are converted to categorical.
- **String processing pipeline**: the *Descript* column consists of strings and its values undergo two distinct pipelines for the *Bag-of-Word* and *TF-IDF* approaches.

Project Pipeline

Features preparation: we aggregate all features columns, except the category one, into a unique column. This can be done by using **pyspark.ml.feature.RFormula**, which produces a vector of features and a column of labels, representing the target (*i.e.* the *Category*). This step is crucial to perform training.

Train and test split: we randomly separate the dataset into train (70%) and test (30%).

Models definition and training: we use **pyspark.ml.classification** to define, train and evaluate the machine learning models.

- Decision Tree
- Random Forest
- Naive Bayes
- Logistic Regression
- Gradient-Boosted Tree
- MultiLayer Perceptron
- Logistic Regression with TF-IDF
- Factorization Machines

Evaluation: we compare the models in terms of *Accuracy*, *F1-score* and *LogLoss*.

Exam

Decision Tree

A decision tree is a tree-like model used for decision-making, where each node represents a decision based on specific features, leading to outcomes at the leaf nodes.

In order to improve model's results we perform parameter tuning via ***CrossValidation***.

Accuracy

F1-score

Log-loss

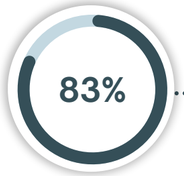
Parameters:

- *maxDepth* = 10
- maxBins = 256

Random Forest

Random forest is an **ensemble learning method** particularly well-suited for multi-label classification because it is able to handle complex relationships between features and labels. It is are also relatively robust to overfitting.

In order to improve model's performances we perform parameter tuning via ***CrossValidation***.



Accuracy

F1-score

Log-loss

Parameters:

- numTrees = 120
- maxDepth = 8
- maxBins = 256

Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem. It is commonly used for classification tasks and it typically performs well in real-world scenarios.

In order to improve model’s performances we perform parameter tuning via ***CrossValidation***.

Accuracy

F1-score

Log-loss

Parameters:

- *smoothing* = 0.0

Logistic Regression with BOW

Logistic regression is a statistical model that is used for binary classification but it can also be extended to handle multi-label classification problems.

In order to improve model's performances we perform parameter tuning via ***CrossValidation***.

Accuracy

F1-score

Log-loss

Parameters:

- *maxIter* = 10
- regParam = 0.1
- elasticNetParam = 0.0

Gradient-Boosted Tree

Gradient Boosted Trees (GBTs) are a type of **ensemble learning algorithm** that combines multiple decision trees to improve prediction accuracy.

To handle multi-label classification we employ the **One-vs-Rest** approach, which trains separate classifiers for each class, treating one class as positive and the rest as negative.

Accuracy

F1-score

Parameters:

- *maxIter* = 10

Comparison of Results

Model	Accuracy	F1-score	Log-loss
Decision Tree	0.7008	0.7056	1.0062
Random Forest	0.8330	0.8058	1.3347
Naive Bayes	0.9952	0.9955	**NaN**
Logistic Regression	0.9957	0.9958	0.2067
Gradient-Boosted Tree	**0.9958**	**0.9959**	---

Project Work

Dataset Preprocessing

In adddition to the preprocessing performed for the Exam’s section, it is necessary to add a further step in order to allow the training for two of the three models defined for the Project Work.

Standardize features: standardization is an important step in many machine learning algorithms, especially those that involve distance-based metrics or gradient-based optimization, as it helps ensure that all features have a similar scale.

We use **py*spark.ml.feature.StandardScaler*** to ensure that each scaled feature has 0 mean and 1 standard deviation.

MultiLayer Perceptron

A multiclassification multi-layer perceptron (MLP) is a neural network that categorizes input data into multiple classes. It uses multiple layers of interconnected nodes with non-linear activation functions to learn and recognize complex patterns for accurate classification.

Since neural networks employ gradient-based optimization algorithms to train the weights, we train the model on **scaled data** (i.e. with scaled features).

Accuracy

F1-score

Log-loss

Parameters:

- *maxIter* = 100
- blockSize = 32
- layers = [1095, 100, 39]

where 1095 is the number of features, and 39 is the number of classes

Logistic Regression with TF-IDF

We apply the Logistic Regression model to an alternative preprocessed dataset, distinct from the one previously discussed. In this dataset, the *Descript* column undergoes vectorization using the **TF-IDF Vectorizer**.
This method considers not only the frequency of a word within a document but also the significance of the word across the entire corpus.

Accuracy

F1-score

Log-loss

Parameters:

- *maxIter* = 10
- regParam = 0.1
- elasticNetParam = 0.0

Factorization Machines

Factorization Machines are a type of machine learning model that can be used to model the interactions between features. In order to perform multiclassification, the **One-vs-Rest** approach is employed, decomposing the multiclass problem into binary tasks.

The model is trained on the **scaled features**.

Accuracy

F1-score

Parameters:

- *maxIter* = 100
- stepSize = 10

Comparison of Results

Model	Accuracy	F1-score	Log-loss
MultiLayer Perceptron	**0.9996**	**0.9996**	**0.0012**
Logistic Regression with TF-IDF	0.9901	0.9891	0.2582
Factorization Machines	0.9970	0.9971	---