Grouping and Classifying Oil Platforms Accidents Causes using Supervised Machine Learning

José Geraldo de Lima Júnior
VIRTUS
Cônego Pequeno street, nº 490, Bela
Vista
Campina Grande, PB, Brazil
geraldo.lima@virtus.ufcg.edu.br

ABSTRACT

With increasingly modern security techniques and procedures, companies in all segments have invested vigorously in the control and prevention of incidents to protect their employees and properties. Not unlike, oil companies are turning to studies in Artificial Intelligence to improve such procedures. In 2010, more than 2,700 cases were recorded in the USA, which resulted in billionaire financial losses and socio-environmental damage. Previous studies related to this topic focus only on the probability of leaks and contamination by oil spills, or on the prediction of oil derivatives prices. This study analyzes the combinations between the types of fluids and pipes used in oil exploration activities that may cause major accidents during their execution. Therefore, using Supervised Machine Learning and Clustering techniques, the results of this research can be used as a tool to predict the cause and location of new accidents, making it possible to create specific prevention strategies.

Keywords

Oil exploration, accident risk, clustering, supervised machine learning, decision tree, python.

1. INTRODUCTION

Companies are always worried about their workplaces safety control, because it evolves its workers' lives at first, and the efficient execution of their lines of production. This includes how the company manages its raw materials and equipment combinations, which can lead to serious accidents when they do not match for some reason. Oil exploration entities are very familiar with this kind of problem since it causes a lot of financial, socio-environmental, and human losses.

This study hopes to contribute with the developed SML (Supervised Machine Learning) model in Python [1] for accident classification and all data analysis to the oil exploration engineers community, companies, and environment protection authorities. To achieve such results, it was used several well-known Python libraries for data analysis and artificial intelligence like *scikit-learn* [2], *pandas* [3], and *seaborn* [4]. With these libraries, it was possible to read, prepare, and process all the data. After reading the dataset, several pre-processing steps were executed in order to avoid processing invalid data and to discover which features from the dataset have more correlation with the target value (the accident cause) and research value. Also, distribution and correlation charts of key features were made available for discussion. Finally, it was verified which categorical features could be used to generate *dummie* columns, so the training and test sets could be generated.

In the final steps, it was applied clustering techniques to evaluate whether the dataset has a natural grouping, but unfortunately, it hasn't. Also, dimensionality reducing techniques were executed in order to find the most important features in the dataset, but since it has too much variance, this wasn't possible too. Finally, some classifiers (Decision Tree, K-Neighbors, Random Forest, and SVC - Support Vector Classification) were tested to find the best solution for this problem, based mainly on the accuracy, precision, and *f1-score* metrics. The *Decision Tree* type was the chosen one, having a performance of 100% in all the metrics.

This paper is organized as follows. Section 2 presents the Methods of Research where all the steps to reach the results are explained. Section 3 describes the Challenges in Machine Learning, to discuss the world of solutions in AI (Artificial Intelligence) and how to properly choose one of them. Finally, Section 4 summarizes the main contributions and presents future work.

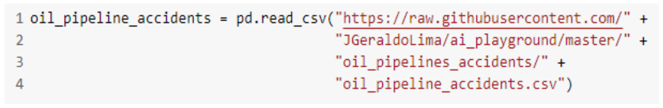
2. METHOD OF RESEARCH

In this study, the focus is to answer the following question: which fluid types and pipeline combinations are more dangerous? And to be able to train the SML model, a Kaggle dataset [5] proposed by the U.S. Department of Transportation and composed by a record for each oil pipeline leak or spill reported to the United States Pipeline and Hazardous Materials Safety Administration since 2010, was used in the entire research. These official records include detailed data regarding the accidents, which it turns to be a rich source of information that can be largely explored. But first, it is necessary to do some pre-processings to clean up the dataset, removing outliers, invalid data, and non-valuable features.

2.1 Importing Python Libraries and Reading the Dataset

Python has plenty of different tools to work with data analysis and visualization. For this article, beyond the libraries already mentioned earlier, there were used several other libraries, such as *numpy* [6], *matplotlib* [7], and *scipy* [8].

To make the dataset easily readable from an URL for anyone who wants to use it too, the .csv file was downloaded from Kaggle and stored in the author GitHub[1] account. With pandas available, it is possible now to get the data and convert it into a DataFrame object, which was named *oil_pipeline_accidents* in this paper:

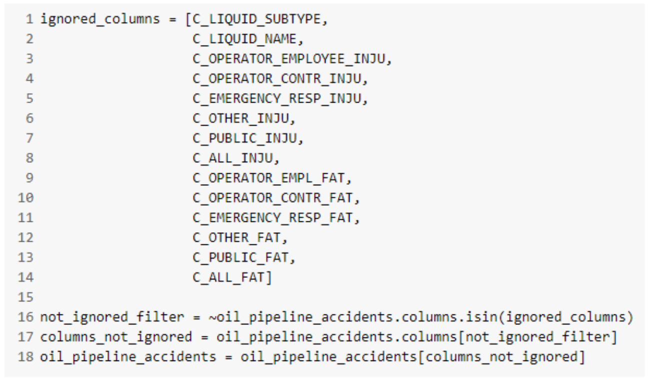


```python
oil_pipeline_accidents = pd.read_csv("https://raw.githubusercontent.com/" +
                                     "JGeraldoLima/ai_playground/master/" +
                                     "oil_pipelines_accidents/" +
                                     "oil_pipeline_accidents.csv")
```

Figure 1 - Reading the dataset with pandas

2.2 Getting Rid of Unuseful Data

First of all, based on the resumed Kaggle dataset features values distribution statistics, the columns which have mostly (50% or more) *NaN* (Not a Number) values are dropped from the read dataframe. The mentioned columns are shown in Figure 2. For more details, please see the columns description section.

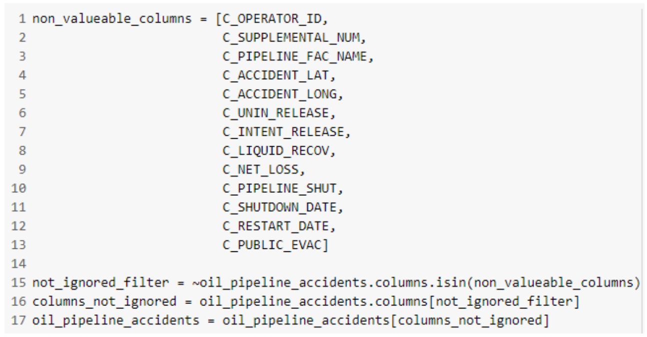


```python
ignored_columns = [C_LIQUID_SUBTYPE,
                   C_LIQUID_NAME,
                   C_OPERATOR_EMPLOYEE_INJU,
                   C_OPERATOR_CONTR_INJU,
                   C_EMERGENCY_RESP_INJU,
                   C_OTHER_INJU,
                   C_PUBLIC_INJU,
                   C_ALL_INJU,
                   C_OPERATOR_EMPL_FAT,
                   C_OPERATOR_CONTR_FAT,
                   C_EMERGENCY_RESP_FAT,
                   C_OTHER_FAT,
                   C_PUBLIC_FAT,
                   C_ALL_FAT]

not_ignored_filter = ~oil_pipeline_accidents.columns.isin(ignored_columns)
columns_not_ignored = oil_pipeline_accidents.columns[not_ignored_filter]
oil_pipeline_accidents = oil_pipeline_accidents[columns_not_ignored]
```

Figure 2 - Dropping columns which have mostly NaN values

Now, we need to drop those columns that do not have research value (due to lack of definition, non-related data with this research - like "Shutdown Date/time", too many fault data, etc), see Figure 3.

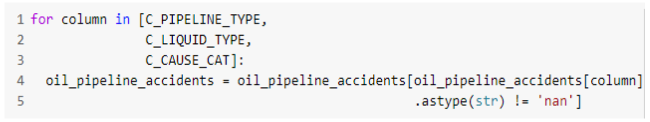


```python
non_valueable_columns = [C_OPERATOR_ID,
                         C_SUPPLEMENTAL_NUM,
                         C_PIPELINE_FAC_NAME,
                         C_ACCIDENT_LAT,
                         C_ACCIDENT_LONG,
                         C_UNIN_RELEASE,
                         C_INTENT_RELEASE,
                         C_LIQUID_RECOV,
                         C_NET_LOSS,
                         C_PIPELINE_SHUT,
                         C_SHUTDOWN_DATE,
                         C_RESTART_DATE,
                         C_PUBLIC_EVAC]

not_ignored_filter = ~oil_pipeline_accidents.columns.isin(non_valueable_columns)
columns_not_ignored = oil_pipeline_accidents.columns[not_ignored_filter]
oil_pipeline_accidents = oil_pipeline_accidents[columns_not_ignored]
```

Figure 3 - Dropping columns that do not have research value

The related dataset has several columns with too many *NaN* values, which can cause slow training and non-expected results. To avoid such problems, three steps are executed to clean these values, starting with the three main categorical columns:



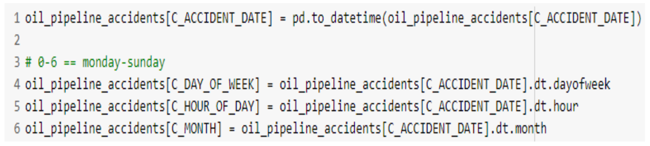
```python
for column in [C_PIPELINE_TYPE,
               C_LIQUID_TYPE,
               C_CAUSE_CAT]:
  oil_pipeline_accidents = oil_pipeline_accidents[oil_pipeline_accidents[column]
                                                  .astype(str) != 'nan']
```

Figure 4 - Dropping rows where the selected categorical columns have invalid data

Then, the other columns (including numeric types) are cleaned up using the same previous processes. Now, the dataset *oil_pipeline_accidents* is ready to be used.

2.3 Creating New Features Based on the 'Accident Date/Time' Column

A common practice in data analysis study with Python is to create new columns based on one or more columns available on the dataset. In this case, the feature 'Accident Date/Time' was used to extract new ones. For now, they were used only to improve the SML model training, since these date/time features are available for all columns and can be used to detect correlations between the variables.



```python
oil_pipeline_accidents[C_ACCIDENT_DATE] = pd.to_datetime(oil_pipeline_accidents[C_ACCIDENT_DATE])

# 0-6 == monday-sunday
oil_pipeline_accidents[C_DAY_OF_WEEK] = oil_pipeline_accidents[C_ACCIDENT_DATE].dt.dayofweek
oil_pipeline_accidents[C_HOUR_OF_DAY] = oil_pipeline_accidents[C_ACCIDENT_DATE].dt.hour
oil_pipeline_accidents[C_MONTH] = oil_pipeline_accidents[C_ACCIDENT_DATE].dt.month
```

Figure 5 - Extracting day of the week, the hour of day and month new features

The pre-processing steps were concluded, the dataframe can now be processed and data visualizations can be made.

2.4 Analysing Data

Before proceeding with the SML model creation, it is necessary to analyse the dataframe data to better understand the distributions, correlations and verify if the selected features ('Pipeline Type' and 'Liquid Type') for training and validation of the classification are the best options.

2.4.1 Checking Data Distributions

For the data distributions, it was used the library *seaborn* to plot charts. In the filtered data set, there are only Onshore pipeline location records. See the following figures to check the other main columns distributions.

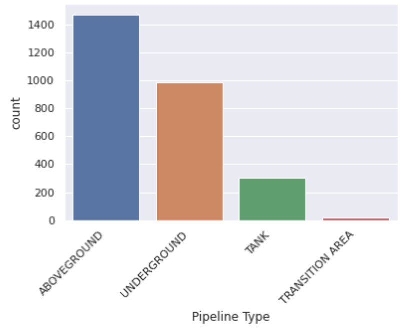


Figure 6 - Pipeline type distribution - most of the accidents were caused by underground and aboveground types

[1] https://raw.githubusercontent.com/JGeraldoLima/ai_playground/master/oil_pipelines_accidents/oil_pipeline_accidents.csv

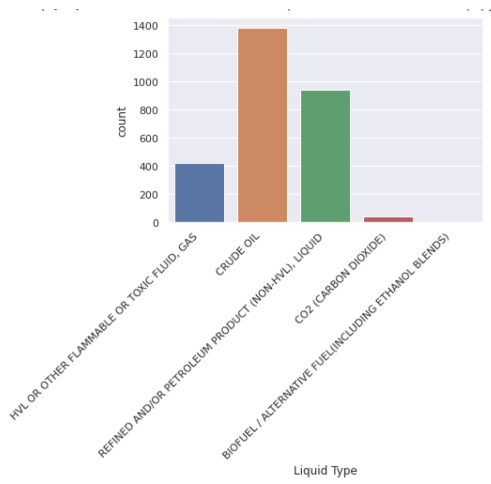


Figure 7 - Liquidy type distribution - most of the accidents were caused by crude oil and refined and or petroleum product liquid types

With these data now available, may it is possible to finally answer questions:

- Which causes categories had more classified accidents by?
- Are the pipeline and liquid types data relevant for the classification?
- Assuming that there is only these information available, without any specific Oil Engineering background, is it possible to say that the correlations make sense?

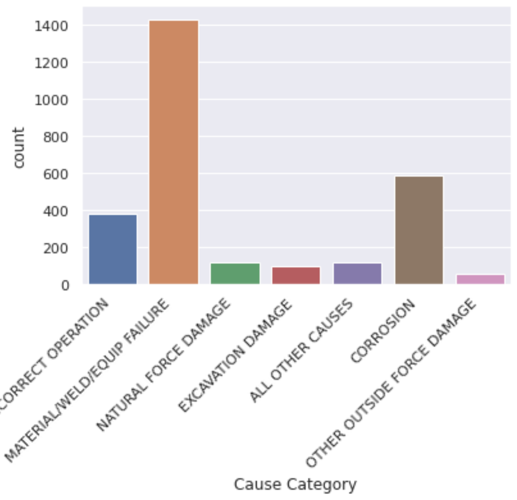
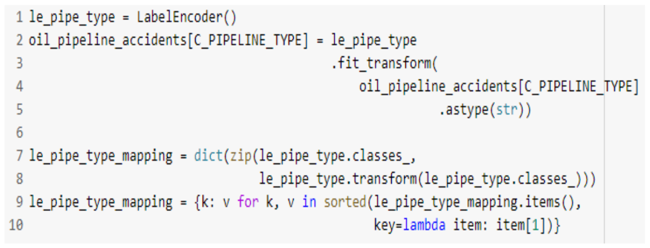


Figure 8 - Accidents cause distribution - most of the accidents were caused by material / bad weld / equipment failure and corrosion, with a significant amount for the incorrect operation

Indeed, since the main cause includes equipment failure, the pipeline and liquid type may interfere with it.

2.4.2 Checking the Correlations with the Target Variable ('Cause Category')

First of all, it is necessary to convert the independent ('Pipeline Type' and 'Liquid Type') and dependent ('Cause Category') variables values into numeric type, so the correlation charts can be generated easily.

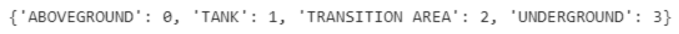


```
le_pipe_type = LabelEncoder()
oil_pipeline_accidents[C_PIPELINE_TYPE] = le_pipe_type
                                          .fit_transform(
                                              oil_pipeline_accidents[C_PIPELINE_TYPE]
                                                         .astype(str))

le_pipe_type_mapping = dict(zip(le_pipe_type.classes_,
                                le_pipe_type.transform(le_pipe_type.classes_)))
le_pipe_type_mapping = {k: v for k, v in sorted(le_pipe_type_mapping.items(),
                                                key=lambda item: item[1])}
```

Figure 9 - Using the *scikit-learn LabelEncoder* tool to convert the string labels into categorical (numeric) type.

The same code was used to convert 'Liquid Type' and 'Cause Category' columns values too. After converting them, the labels map is saved for later use and here is how it looks like:



```
{'ABOVEGROUND': 0, 'TANK': 1, 'TRANSITION AREA': 2, 'UNDERGROUND': 3}
```

Figure 10 - Columns label mapping example

Now, the correlation heatmaps are generated. Heatmaps were chosen due to their simple visualization and interpretation.

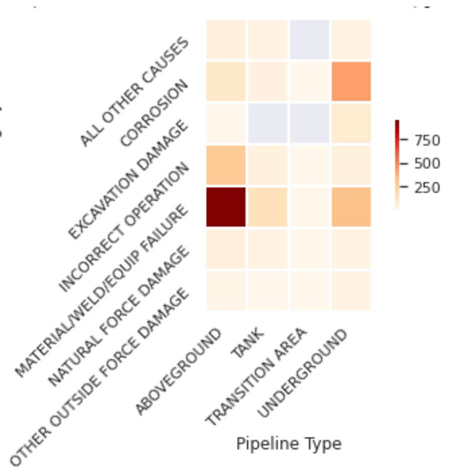


Figure 11 - Showing the correlation heatmap between cause category and pipeline type

From the previous chart, we have:

- **Aboveground** pipelines are the main cause of **material/weld/equip failure** accidents;
- **Underground** pipelines are the main cause of **corrosion** accidents;
- **Undergound** pipelines have an expressive blame rate in **material/weld/equip failure** accidents;
- **Aboveground** pipelines are closely related to **incorrect operation** accidents;
- **Tank** pipelines have a small portion of the blame in **material/weld/equip failure** accidents;
- **Aboveground** have a small portion of the blame in **corrosion accidents**;

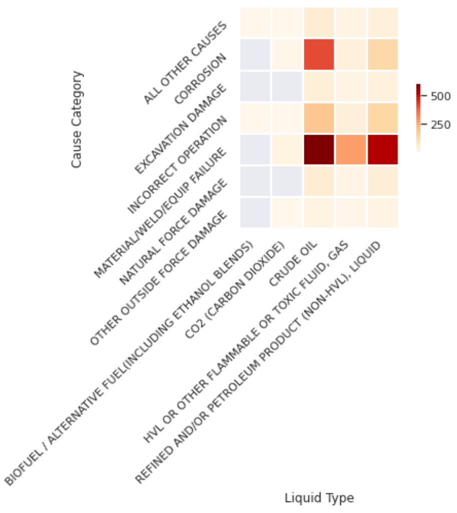


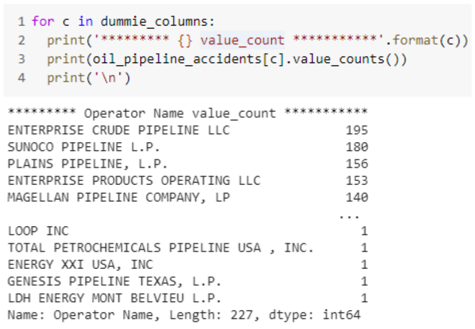
Figure 12 - Showing the correlation heatmap between cause category and liquid type

From the previous chart, we have:

- **Crude oil** fluid are the main cause of **material/weld/equip failure** accidents;
- **Refined / petroleum product** fluids are the main cause of **material/weld/equip failure** accidents;
- **Crude oil** fluid have an expressive blame rate in **corrosion** accidents;
- **HVL / other flammable / toxic** fluids are closely related to **material/weld/equip failure** accidents;
- **Crude oil** fluid have a small portion of blame in **incorrect operation** accidents;
- **Refined / petroleum product** fluids have a small portion of blame in **incorrect operation** accidents;
- **Refined / petroleum product** fluids have a small portion of blame in **corrosion** accidents.

2.5 Clustering Attempts

Before trying to execute the clustering techniques, *dummie* columns were needed to be generated in order to enhance cluster algorithm performance. But not all the categorical features were used in this process, first it was needed to verify the columns content and list all unique values for each one to check how many levels and its size there were on them (see Figure 13).



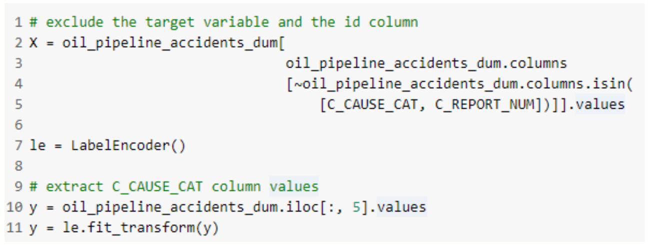
```
for c in dummie_columns:
  print('********* {} value_count ***********'.format(c))
  print(oil_pipeline_accidents[c].value_counts())
  print('\n')
```

```
********* Operator Name value_count ***********
ENTERPRISE CRUDE PIPELINE LLC                195
SUNOCO PIPELINE L.P.                         180
PLAINS PIPELINE, L.P.                        156
ENTERPRISE PRODUCTS OPERATING LLC            153
MAGELLAN PIPELINE COMPANY, LP                140
                                            ...
LOOP INC                                       1
TOTAL PETROCHEMICALS PIPELINE USA , INC.       1
ENERGY XXI USA, INC                            1
GENESIS PIPELINE TEXAS, L.P.                   1
LDH ENERGY MONT BELVIEU L.P.                   1
Name: Operator Name, Length: 227, dtype: int64
```

Figure 13 - List all unique values and its quantity for each column

After the analysis, the selected columns were used and the remaining ones were discarded, using the same solution as used in Section 2.2. Then, the *dummie* columns were generated using the *pandas get_dummies* tool.

Lastly, the dependent and independent variables are separated:



```
# exclude the target variable and the id column
X = oil_pipeline_accidents_dum[
                                oil_pipeline_accidents_dum.columns
                                [~oil_pipeline_accidents_dum.columns.isin(
                                    [C_CAUSE_CAT, C_REPORT_NUM])]].values

le = LabelEncoder()

# extract C_CAUSE_CAT column values
y = oil_pipeline_accidents_dum.iloc[:, 5].values
y = le.fit_transform(y)
```

Figure 14 - Extracting the dependent and independent variables to use on dataset split

Now, it is possible to generate the train and test sets. To do so, the *scikit-learn* train_test_split function was used. Then, with train and test sets available, the clustering techniques can be applied.

2.5.1 K-Means

Since the number of clusters is known (seven: "All Other Causes", "Corrosion", "Excavation Damage", "Incorrect Operation", "Material/Weld/Equip Failure", "Natural Force Damage", "Other Outside Force Damage"), it was used to run the solution, but the algorithm wasn't able to find natural grouping (see Figure 15).

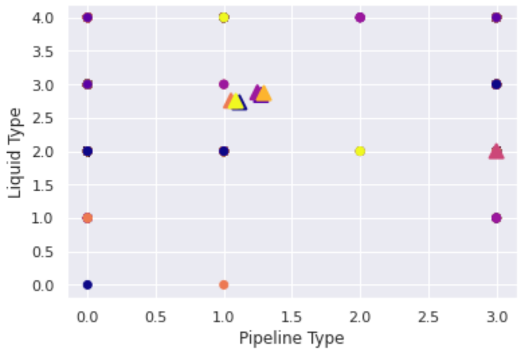


Figure 15 - K-Means solution generated chart

2.5.2 Hierarchical Clustering

The second attempt used the Hierarchical Clustering algorithm, generating the Dendrogram, and using the solution calculated ideal number of clusters (in this case, five - check Figure 16).

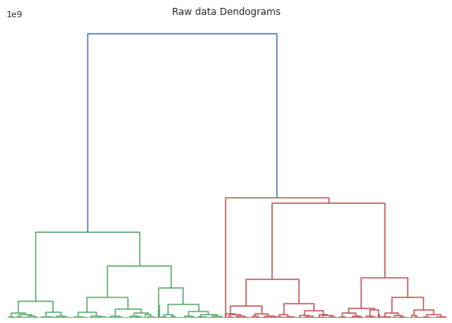
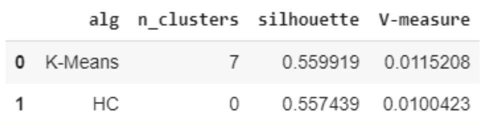


Figure 16 - Generated Dendrogram by *scipy* library

While running the algorithms, the *silhouette* and *v-measure* scores were calculated. See the results in Figure 17.



	alg	n_clusters	silhouette	V-measure
0	K-Means	7	0.559919	0.0115208
1	HC	0	0.557439	0.0100423

Figure 17 - Clustering algorithms results

As can be seen, these are bad results and cannot be considered as clustering solutions. From the analysis above, it is clear that this dataset does not have a natural clustering.

2.6 Dimensionality Reducing Attempts

Even though it is known that the dataset does not have a natural clustering, we tried two dimensionality reducing techniques: t-SNE and PCA.

2.6.1 t-SNE

The *scikit-learn* t-SNE tool expects a number of components (dataset dimensions) as its only parameter. Since the dataset has too many attributes, the maximum value for this parameter was chosen (three). The result is shown in Figure 18.

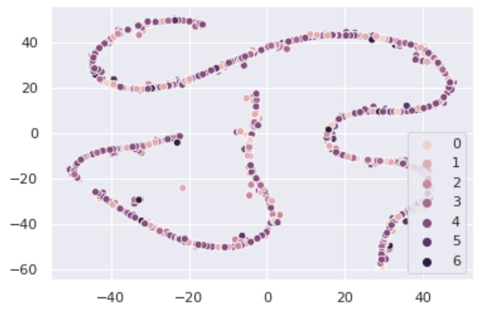


Figure 18 - t-SNE results

2.6.2 PCA

The *scikit-learn* PCA solution has the same dependency (number of components) and the same value was used for it, to try to maintain the consistency in this methodology. The results are shown in Figure 19.

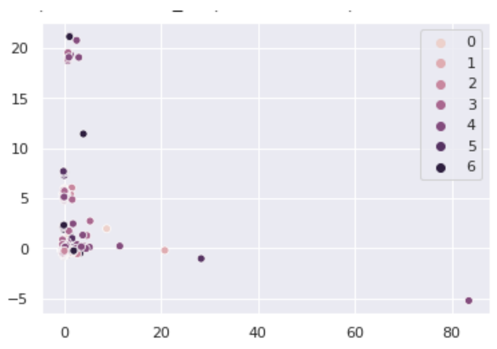
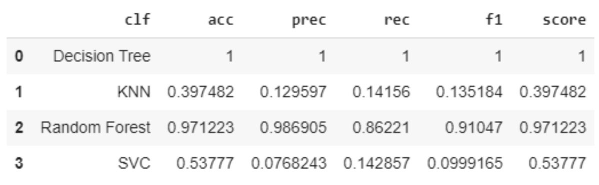


Figure 19 - PCA results

Based on the previous generated two approaches charts, it is possible to conclude that it is not possible to use dimensionality reducing with t-SNE or PCA in this dataset since there is too much variance on it.

2.7 Training Classification Models

Finally, the classification models are trained so the comparison between them can be made and the better solution chosen. For this paper, Decision Tree, K-Neighbors, Random Forest, and SVC models were selected. The validation results can be checked in Figure 20.



	clf	acc	prec	rec	f1	score
0	Decision Tree	1	1	1	1	1
1	KNN	0.397482	0.129597	0.14156	0.135184	0.397482
2	Random Forest	0.971223	0.986905	0.86221	0.91047	0.971223
3	SVC	0.53777	0.0768243	0.142857	0.0999165	0.53777

Figure 20 - Classification models validation results

Therefore, the Decision Tree proves itself to be the best solution for this dataset and to seal this decision, the confusion matrix was generated. See the results in Figure 21.

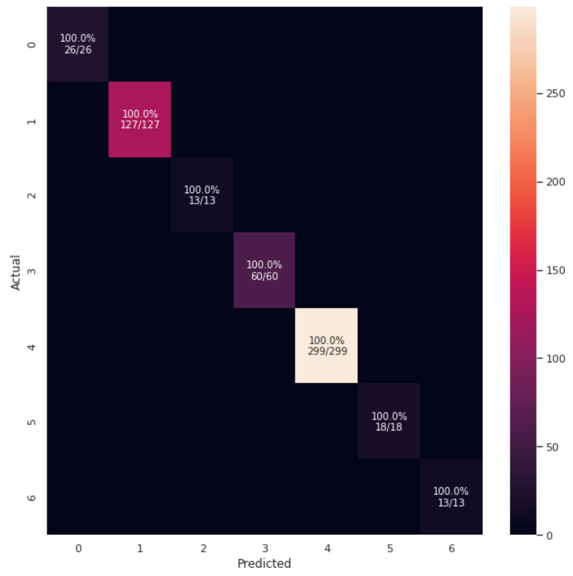


Figure 21 - The Decision Tree model confusion matrix

3. Challenges in Machine Learning

Machine Learning is a huge world with infinite possibilities. What makes an ML solution really useful is how the solution model is built, the parameters used for, how much you know the dataset, the relevance of its data, the correlation between the features, and so on.

Understand your dataset content is an important part of the process because the selection of the techniques depends on it. Also, pre-processing practices make all the difference.

Since this paper discusses a classification problem for a simple dataset, the model selection was pretty easy. However, as shown in the Method of Research section, sometimes it is not possible to use some techniques (in this case, Clustering and Dimensionality Reduction) and it can be either a problem or an opportunity to explore deeply.

4. Conclusion

In this paper, we show that the generated model can be used for the industry to classificate the oil exploration platforms accidents and then search for new combinations of pipeline and liquid types.

Also, the correlations make sense: the most common accidents cause is "material / bad weld / equip failure" (see Figure 8); the most problematic pipeline type is the aboveground (see Figure 6) and the liquid type evolved in the most accidents is the crude oil (see Figure 7), i.e., maybe this fluid does not fit with above ground pipelines for some reason and this may lead to equipment failure. An Oil Engineer can use these results and take technical decisions about the exploration installations to avoid financial, socio-environmental, and human losses.

There is more to explore in this dataset, to produce more results in different areas (e.g.: financial and environmental). As future work, we intend to relate the accident city/state and weather temperature/humidity with the cause category, as well as the hour of the day, day of week and month to discuss it. Moreover, we also aim at the costs (where it is higher) to the pipeline location, pipeline type, liquid type, and so on.

ACKNOWLEDGMENTS

My thanks, at prior, to God, to be so kind in my entire life; to Compal Electronics and VIRTUS for providing me the opportunity to participate in the AIE course; to all the professors and monitors that always answered my questions and made the Artificial Intelligence knowledge way easier to assimilate; my family for all the support; and my fiancé to understand my tight schedule until this finishes.

REFERENCES

[1] Willi Richert, W., Coelho, L.P., Building Machine Learning Systems with Python. Packetpub.com, 2013.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830.

[3] Reback J., Wes McKinney W., Van den Bossche J., Augspurger T., Cloud P., Mehyar M., Pandas 1.0.3 (Version v1.0.3), 2020.

[4] Michel Waskorn. seaborn Python library. Available at https://seaborn.pydata.org/index.html. Accessed on 04/24/2020.

[5] U.S. Department of Transportation. Oil Pipeline Accidents, 2010-Present. Causes, injuries/fatalities, and costs of pipeline leaks and spills. Available at: https://www.kaggle.com/usdot/pipeline-accidents#database.csv, accessed on 04/24/2020.

[6] Van der Walt, S., Colbert, S. C., Varoquaux, G., The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011).

[7] Hunter, J. D., Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007).

[8] Millman, K. J., Aivazis, M., Python for Scientists and Engineers, Computing in Science & Engineering, 13, 9-12 (2011).

About the author:

José Geraldo de L. Júnior is a Software Engineer and Team Leader at VIRTUS, where he's always been challenged and it has been his second home for the past five years. He currently leads a medium software team project for a Taiwanese company client. He earned an MBA from UNIFACISA where his work was based on the learned Mobile Applications Development technologies and practices. See more at https://www.linkedin.com/in/josegeraldojunior/.

APPENDIX

DATASET COLUMNS DESCRIPTION

- C_REPORT_NUM - The record id number
- C_SUPPLEMENTAL_NUM - Supplemental number
- C_ACCIDENT_YEAR - The accident year
- C_ACCIDENT_DATE - The accident date/time
- C_HOUR_OF_DAY - Hour of accident day
- C_DAY_OF_WEEK - Accident day of the week
- C_MONTH - Month of the accident
- C_OPERATOR_ID - Operator ID evolved in the accident
- C_OPERATOR_NAME - Operator name evolved in the accident
- C_PIPELINE_FAC_NAME - Facility name where the accident happened
- C_PIPELINE_LOCATION - Where the pipeline were located in the facility
- C_PIPELINE_TYPE - The pipeline type
- C_LIQUID_TYPE - The liquid type
- C_LIQUID_SUBTYPE - The liquid subtype
- C_LIQUID_NAME - The liquid name
- C_ACCIDENT_CITY - The city where the accident happened
- C_ACCIDENT_COUNTY - The county where the accident happened
- C_ACCIDENT_STATE - The state where the accident happened
- C_ACCIDENT_LAT - The location’s latitude where the accident happened
- C_ACCIDENT_LONG - The location’s longitude where the accident happened
- C_CAUSE_CAT - The accident cause category
- C_CAUSE_SUBCAT - The accident sub cause category
- C_UNIN_RELEASE - Unintentional Release (Barrels)
- C_INTENT_RELEASE - Intentional Release (Barrels)
- C_LIQUID_RECOV - Liquid Recovery (Barrels)
- C_NET_LOSS - Net Loss (Barrels)
- C_LIQUID_IGN - Whether there was a liquid ignition
- C_LIQUID_EXPL - Whether there was a liquid explosion
- C_PIPELINE_SHUT - Whether the pipeline was shutdown
- C_SHUTDOWN_DATE - The pipeline shutdown date/time
- C_RESTART_DATE - The pipeline restart date/time
- C_PUBLIC_EVAC - Public Evacuations
- C_OPERATOR_EMPLOYEE_INJU - Operator Employee Injuries
- C_OPERATOR_CONTR_INJU - Operator Contractor Injuries
- C_EMERGENCY_RESP_INJU - Emergency Responder Injuries
- C_OTHER_INJU - Other Injuries
- C_PUBLIC_INJU - Public Injuries
- C_ALL_INJU - All Injuries
- C_OPERATOR_EMPL_FAT - Numer of operator employee fatalities in the accident
- C_OPERATOR_CONTR_FAT - Numer of operator contractor fatalities in the accident
- C_EMERGENCY_RESP_FAT - Numer of emergency responder fatalities in the accident
- C_OTHER_FAT - Other fatalities caused by the accident
- C_PUBLIC_FAT - Public fatalities caused by the accident
- C_ALL_FAT - All fatalities in the accident
- C_PROPERTY_DMG - Property damage costs
- C_LOST_COMMO - Lost commodity costs
- C_PUBLIC_PRIV - Public/private property damage costs
- C_EMERGENCY_RESP - Emergency response costs
- C_ENVIRONMENTAL_REM - Environmental Remediation Costs
- C_OTHER_COSTS - Other costs
- C_ALL_COSTS - All costs

For more details, see this dataset source from Kaggle at https://www.kaggle.com/usdot/pipeline-accidents.