Predicting Root Causes of Safety Observations

Classifying Safety Observations using Natural Language Learning Process to predict root causes.

An assignment submitted in fulfillment of the requirements of IBM Introduction to Machine Learning Specialization

By

Minhaj Ahmed Ansari

1. Introduction

1.1. The target of this machine learning exercise is to identify the best model to predict the root cause of the registered Safety Observation.

1.2. A root cause analysis refers to the process of identifying the underlying cause of an issue or incident. Whether you're investigating an employee who crushed their foot when they dropped a heavy box or one who tripped over a cord in a poorly lit office, an RCA goes beyond simply removing the immediate cause of the accident

1.3. The brief objective of this Machine learning exercise is to preempt the root causes of the safety observations raised by learning from the historical labelled data.

2. Brief description of the data set and a summary of its attributes

2.1. The data which I am using here is a dump from the Safety Observation System of a Major Company. Each row of data consists of the observation description along with pre-selected category and root cause (Classes).

2.2. The objective of this program would be to develop a Machine Learning Model which would be able to predict the root cause based on this historical data for any new observation. By implication this exercise is a Natural Language Processing Problem. As we will be extracting features from text (Observation Description) and using those for training our classification model.

2.3. The number of columns in the dataset is 3 and the no of rows is 1028493. The target variable here will be 'Sub-Category' and the input variable will be 'Observation'. Another column is 'UA/UC' which broadly categorizes observations into different types of unsafe acts and conditions.

3. Data Cleaning and Wrangling

3.1. Plan

- Bring all words on same case (lower case).
- Detect the missing values and treat them accordingly.
- Drop various Stop words which don't add any information to the model.

3.2. Actions

- Importing the necessary module for text preprocessing and feature extraction.

```
[4]: #Let us first bring in all the required modules

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from collections import Counter
```

- During this process it was identified that there are lot of null value columns in the entire datasets.

```
[5]: #Lets first drop the 'UA/UC' Column which is not required for this exercise.

df_obs.drop('UA/UC', axis = 1, inplace = True)
```

```
[6]: df_obs.head()
```

[6]:

	Observation	Sub-Category
0	worker used drilling machine without green tag.	NaN
1	Worker used a plug top for multiple electrica...	No Plug top
2	worker use drilling machine at site without gr...	NaN
3	Site supervisor is not available at site whi...	NaN
4	worker use electric grinder without green tag.	No Inspection Done

```
[7]: # Lets check the missing values

df_obs.isnull().sum()
```

```
[7]: Observation     961881
     Sub-Category    996100
     dtype: int64
```

- Also, there were lot of missing labels for almost half of the data. Now this leaves a lot of room for improvement in data collection and labelling process. Unfortunately, we must leave out all these rows from our modelling.

```
[9]: # Lets check how many values are missing in target variable after dropping missing feature rows
df_obs.shape[0], df_obs.isnull().sum()
```

```
[9]: (66612,
      Observation         0
      Sub-Category    34219
      dtype: int64)
```

We can see that we missing lables for almost half of the data. Now this leaves a lot of room for improvement in data collection and labelling process. Unfortunately, we have to leave out all these rows from our modelling.

```
[10]: #droping values
df_obs.dropna(subset = ['Sub-Category'], axis = 0, inplace = True)
df_obs.shape[0], df_obs.isnull().sum()
```

```
[10]: (32393,
       Observation     0
       Sub-Category    0
       dtype: int64)
```

So, in the end we are now left with 32,393 Nos of individual observations

```
[11]: # Lets check for the duplicates in the observation column

df_obs.duplicated(subset = ['Observation'], keep = 'first').sum()
```

```
[11]: 6806
```

- There were also lot of duplicate observations which may skew the results of the modelling.

```
[12]: df_obs.drop_duplicates(subset = 'Observation', keep = 'first', ignore_index = True, inplace = True)
```

- After all, above exercise, the number of columns in the dataset is 2 and the no of rows is 25587

3.3. Summary

- We dropped the null values and duplicated observations to reduce the dimensions of the data
- This also ensures that we avoid assigning higher weightage to certain duplicated observations at the time of feature extraction.
- We also dropped UA/UC columns as we want to explore this dataset purely from NLP perspective.

4. Exploratory Data Analysis

4.1. Plan

- Remove punctuation marks and any other extra space.
- Lemmatize all the words so that different words stemming from same root word are not counted as separate features.
- Label Encode the target variables
- Check the weight of each class and discuss on the possibility of the oversampling, Under sampling or resampling.
- Extract features from our observation text using tf-idf method

4.2. Actions

- Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meaning to one word.
- For effective lemmatization, punctuations and other similar spaces are also removed.
- For lemmatization, the module which has been used is Word Net Lemmatizer under NLTK library.

```
[17]: def lemmatize_text(word_list):
          lem_word_list = [wordlemmatize.lemmatize(word, pos = 'v') for word in word_list.split()]
          new_lem_word_list = [wordlemmatize.lemmatize(word) for word in lem_word_list]
          return " ".join(new_lem_word_list)

[18]: df_obs.Observation = df_obs.Observation.apply(lemmatize_text)

[19]: #Lets check our cleaned dataframe

      df_obs.sample(10)
```

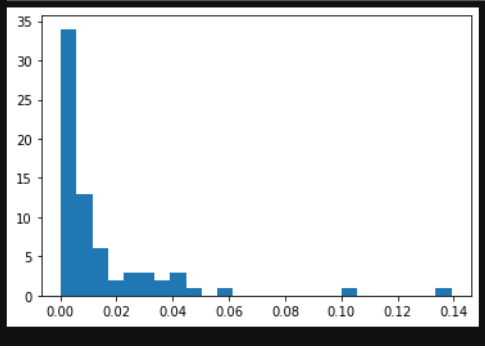
[19]:

	Observation	Sub-Category
3847	Poor electrical joint in cable lay on grind	Improper Cable joints
3741	plug top not implement naked wire use to opera...	No Plug top
22650	improper housekeep be find in work area	Poor Housekeeping
22764	Wooden ladder be find to be use for access egr...	Improper ladder arrangement
20453	material keep on road without barricade	Hard Barricades not provided
19321	Electrical cable tray be use for access egress...	Unsafe/ Blocked Access & Egress
16366	contractor supervisor engage to it work withou...	Workmen engaged without screening/ Induction
15664	Enercom Workmen bhupendra kumar deploy directl...	Workmen engaged without screening/ Induction
23298	Trench pit we re find open condition and witho...	Openings not covered/ barricaded
6842	weld work be carry out without remove water fr...	Unsafe Electrical connections/Practices ...

- If you watch closely lemmatization has taken effect on many words for ex- against index 7778 the word 'been' has been stemmed to 'be', the word 'kept' has been changed back to 'keep' at 7656. These words would have been extracted as separate features thereby increasing the overall feature dimension
- The target variable is label encoded.

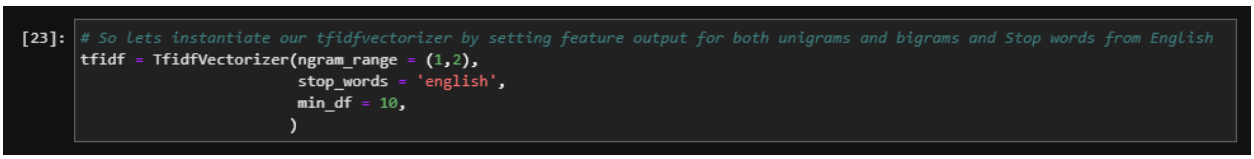
```
[20]: labelencoder = LabelEncoder()
      df_obs['Sub-Category'] = labelencoder.fit_transform(df_obs['Sub-Category'])

[21]: plt.hist(df_obs['Sub-Category'].value_counts(normalize = True), bins = 25,);
```



- This shows that our classes have quite severe imbalance. And this can impact our final model's performance. Instead of opting for any sampling approach let's try to model our data using imbalanced classes by maintaining class imbalance in the train and test split. We will use class_weight feature wherever available to penalize the imbalance.

- Next and probably most important step for modelling is to extract features from text data. Now there are many methods to achieve this objective but the most important from all is Term Frequency and Inverse Document Frequency Method.
- In information retrieval, tf–idf, TF*IDF, or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today.
- Another important term that you will come across would be N-grams. In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words, or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles.
- Using Latin numerical prefixes, an n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". For example, a feature - 'Safety' is unigram whereas a feature 'Safety Helmet' would be bigram
- The words which are generally filtered out before processing a natural language are called stop words. These are the most common words in any language (like articles, prepositions, pronouns, conjunctions, etc.) and does not add much information to the text. Also, I have set min_df = 10, which means words that occurred in too few documents, in this case less than 10, should be filtered out.



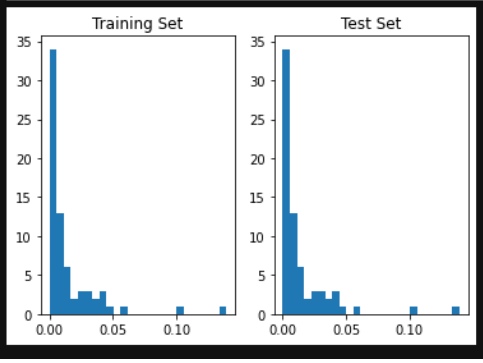
- I would highly recommend the reader to go through sklearn's working with text data resources. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- Before vectorizing, the data is split into training and test sets.

```
[25]: X_train, X_test, y_train, y_test = train_test_split(df_obs.Observation, df_obs['Sub-Category'],
                                                          test_size = 0.3, random_state = 1000,
                                                          stratify = df_obs['Sub-Category']
                                                         )

[26]: #Lets check the class imbalance in both of train and test splits

      fig, (ax1, ax2) = plt.subplots(1, 2)

      ax1.hist(y_train.value_counts(normalize = True), bins = 25,);
      ax1.set(title = 'Training Set');
      ax2.hist(y_test.value_counts(normalize = True), bins = 25,);
      ax2.set(title = 'Test Set');
```



- I have used stratify attribute in train_test_split. So, the class distribution is maintained in train and test splits.
- Next, I have extracted features for further modelling using tfidfvectorizer. A Sample feature matrix is hown below.

```
[27]: features = tfidf.fit_transform(X_train, y_train).toarray()

      print(f'The feature matrix contains {features.shape[0]} observations with {features.shape[1]} features')

      The feature matrix contains 17910 observations with 2582 features

[28]: # Let us create the feature matrix for our future reference

      feature_matrix = pd.DataFrame(data = features, columns = tfidf.get_feature_names())
      feature_matrix.head()
```

[28]:

	aaa	absence	access	access approach	access area	access arrangement	access available	access block	access egress	access filter	...	workplace screen	wtp	xi	xi im	yadav	yadav engage	yard	zone
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 2582 columns

- The feature selection is done using the chi-square test. The Null hypothesis that there is no significance association between the input variable and target variable. While our alternate hypothesis is that there is significance association between the input variable and target variable. Consequently, we will select only those features which have p_value less that 0.05 (Significance level), so that we can reject our null hypothesis.
- Finally, there are 1303 statistically significant features.

4.3. Summary

- We realized that our data consists of highly imbalanced classes. We tried to take that into consideration by splitting our data using stratified option under train_test_split function.
- This ensured that we maintained the equal amount of imbalance in our test as well as train split.
- Next step was to prepare the text data for further feature extraction. We applied various transformations such as lemmatization and achieving homogenization by converting all words into lower case.
- We encoded our target variables.
- Finally, we used tfidfvectorizer to extract features from our text data. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.
- As usual we fitted and transformed tfidfvectorizer over train data and used that fitted model to transform our test data.
- Finally, we performed chi2 test to extract the best associated features with p-value less than 0.05.

5. Model Selection and Evaluation

5.1. Plan

- Drop columns which we rejected based on hypothesis testing.
- Use GridSearchCV to iterate through the models and parameters.
- Select the best model and subsequently the best values for hyperparameters.
- Also, we will try to reach best compromise over the precision and recall values.

- In this part of assignment, I tried modelling using 3 types of Regression models
  - Logistic Regression
  - Support Vector Classification with kernel approximation
  - Random Forrest Classification

5.2. Action

- Dropping the unnecessary features which were rejected using the chi-square test.
- I am using GridSearchCV to find out the best hyperparameters for my model. As explained earlier, I am using stratified kfold for maintaining class imbalance in the cross validated sets.
- Additionally, to optimize the model I am going to set class_weight parameter to 'balanced' to account for the imbalance in class and set warm_start to True to ensure we don't lose our previous coefficients. More importantly this will ensure that our model is efficient and training time is less.
- Another key decision which I make here is that I am choosing weighted f1-score as the scoring metric for my GridSearchCV Object. As we know there is huge imbalance in dataset and trying to classify each minority class correctly may lead to overfitting. Additionally, my target is to get more relevant results rather than just more results, thus I would prefer precision over recall, which means I may get fewer results, but I want to be right. In short, I don't want false alarms.

```
[64]: # random_forrest_pipeline = Pipeline([('standardscaler', StandardScaler()),
      #                                     ('randomforrest', RandomForestClassifier(random_state = 42,
      #                                                                              warm_start = True,
      #                                                                              max_features = 'log2',
      #                                                                              oob_score = True,
      #                                                                              n_jobs = -1))
      #                                    ])

      # params = {'randomforrest__n_estimators': [50, 100, 200, 400],
      #           'randomforrest__max_depth': np.geomspace(1, max_depth_tree+1, 5).astype(int)
      #          }

      # grid_randomforrest = GridSearchCV(random_forrest_pipeline, params, cv = kf, scoring = 'f1_weighted')

[65]: # grid_randomforrest.fit(final_feature_matrix, y_train)

[66]: #pickle_model(grid_randomforrest, 'finalrfcmodel.sav')

[67]: finalrfcmodel = de_pickle('finalrfcmodel.sav')

[68]: finalrfcmodel.best_estimator_
```

- For each model, the best parameters finally achieved are listed below.

[76]:

	logistic regression				SVC Kernel				Random Forrest			
	precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support
accuracy	0.829621	0.829621	0.829621	0.829621	0.674612	0.674612	0.674612	0.674612	0.856454	0.856454	0.856454	0.856454
macro avg	0.784598	0.839341	0.798757	7677.000000	0.510299	0.576438	0.507677	7677.000000	0.847672	0.724757	0.764129	7677.000000
weighted avg	0.844730	0.829621	0.834284	7677.000000	0.733455	0.674612	0.693850	7677.000000	0.854782	0.856454	0.849200	7677.000000

- Here we can see that Random Forrest provides superior results when it comes to weighted scores. Also, Random forrest has better Precision than recall (macro average), which means we get way less false positives. However, when weighted against the support for each label, it easily outperforms logistic Regression model. This meets both of our requirement that we had set forth at the beginning of modelling.
- However, by choosing Random Forrest classifier we risk losing some meaningful predictions for minority classes (Random forrest poorly classifies 18 labels against only 14 labels poorly classified by logistic regression.
- But, in our use case, the cost of missing such True positives is not high, at the same time we need better classification performance for our majority classes.

5.3. Summary

- We tried to train 3 models and chose hyperparameters using GridSearchCV Method
  1. Logistic Regression
  2. Support Vector Classification with kernel approximation
  3. Random Forrest Classification
- We set forth our expectations clearly,
  1. High Precision.
  2. Higher weighted average f1-score.
- Both of which were met by Random Forrest Classifier. We end up selecting a model which had weighted precision, recall and f1 score of 0.854, 0.856 and 0.849 respectively.
- As explained earlier, there is huge imbalance in dataset and trying to classify each minority class correctly may lead to overfitting. Thus, it would be better to avoid false alarms by opting for better precision. Additionally, correct prediction of majority class was must; let me explain in detail,
  - Almost all safety systems are based on basic theory of identifying the root causes and correcting them. This means that if you have truly identified a root

cause and treated it appropriately, there would be no repetition of similar observation. This is of course an ideal situation. However, in real life these incidents may still recur, and it may not be possible to eliminate them due to various factors. Thus, the next best measure which is adopted by most organizations is counting repetition of root causes. Higher the count more serious the issue and thus more immediate action required. This frequency is then multiplied with probable consequence to get the actual risk of incident. However, this model is concerned with only getting the frequency right.

- Thus, it makes sense to identify majority classes with higher precision and recall, hence the choice of measure - Weighted F1 Score for selecting best estimator.

6. Conclusion and Future Directions

6.1. In conclusion, we chose Random Forrest Model for purpose of deployment for future predictions. The model not only shows great prediction power, but it also gives lower false positives at the same time has higher weighted f1-score, precision and recall.

6.2. Going further, in future we may improve our modelling performance by,

- Oversampling or Under sampling or combining both to check whether we can achieve better classification.
- At the time of feature extraction, we only used unigrams and bigrams, we can also use further higher degree of feature such as trigram (combination of three words)
- While training logistic regression model I got lot of failure to convergence warning, we may try to increase the number of iterations to achieve better results.
- In case of SVM, we use kernel approximation to reduce training time and however, in future if we have enough computing resources, we can ditch the approximation go ahead with full scale kernel sampling.
- Again, in Random Forrest Classifier we can again check for intermediate max_depth parameters to see if the results could be achieved using a less complex (less depth) model.
- Another useful thing to do would be to check prediction probabilities and probably get some insights out of it.
- Finally, we can re-work on our input data to maybe eliminate some of the labels by combining with other similar labels, thus reducing some granularity in data when not required.