# Disaster Response Pipeline

By

Krishnan Raghavan

Towards Partial Fulfillment of Udacity's Data Scientist Nanodegree

2021

# Table of Contents

# Abstract

This project involves classifying messages from disaster affected areas into various categories for efficient allocation of resources.

First, a set of classified messages are obtained using a data pipeline that retrieves, cleans, formats and saves the classified data for further analysis. This saved data is then used in a Machine Learning pipeline that is used to tune a classifier model. This model can then be used to classify other messages.

# Chapter 1. Introduction

Providing quick and efficient help to disaster affected areas can mean the difference between tens, hundreds or thousands of lives saved or lost. The better one is able to classify messages from a disaster affected population, the easier it is to allocate the correct resources to the required areas in a timely manner. The following project attempts to use Machine Learning on disaster data, to classify the messages according to various disaster related categories

## 1.1 Business Case

**Problem**: How do I classify a set of disaster related messages?

## 1.2 Such a Project is useful for...

The model developed in this project can be applied to facilitate classifying messages to a variety of disaster setups, such as floods, tornadoes, shortage of water, etc.

## 1.3 Project Overview

In this project, we implement a model development strategy to classify messages. The salient features of this project are:

- A data Pipeline to obtain, clean and format data
- Splitting the data into messages (X) and categorical (Y) data
- Split the X and Y data into training and testing sets, to train the model on the train data and evaluate it on the test data, using appropriate classifier models
- Identify problems with classification, if any, and improve on them
- Output a model that can be used on future messages

## 1.4 Overview of this Report

The following is an overview of the contents of this report:

- Chapter 2: Data – Obtaining data and formatting it for analysis
- Chapter 3: Machine Learning Pipeline – Development of the Machine Learning model
- Chapter 4: Conclusions

# Chapter 2. Data

In this chapter, we talk about what kind of data was used in the project, where it was obtained from and an overview of how it was transformed to create a classifier.

## 2.1 Extract, Transform and Load (ETL) Pipeline:

The messages data consists of two datasets:

1. "**messages.csv**" – the dataset of messages from affected people (Table 2.1.1)

Table 2.1.1: Initial "messages" data dataframe

| | id | message | original | genre |
|---|---|---|---|---|
| 0 | 2 | Weather update - a cold front from Cuba that c... | Un front froid se retrouve sur Cuba ce matin. ... | direct |
| 1 | 7 | Is the Hurricane over or is it not over | Cyclone nan fini osinon li pa fini | direct |
| 2 | 8 | Looking for someone but no name | Patnm, di Maryani relem pou li banm nouvel li ... | direct |
| 3 | 9 | UN reports Leogane 80-90 destroyed. Only Hospi... | UN reports Leogane 80-90 destroyed. Only Hospi... | direct |
| 4 | 12 | says: west side of Haiti, rest of the country ... | facade ouest d Haiti et le reste du pays aujou... | direct |

2. "**categories.csv**" - dataset of categories corresponding to each message in "messages.csv"

Table 2.1.2: Initial "categories" data dataframe

| | id | message | original | genre | categories |
|---|---|---|---|---|---|
| 0 | 2 | Weather update - a cold front from Cuba that c... | Un front froid se retrouve sur Cuba ce matin. ... | direct | related-1;request-0;offer-0;aid_related-0;medi... |
| 1 | 7 | Is the Hurricane over or is it not over | Cyclone nan fini osinon li pa fini | direct | related-1;request-0;offer-0;aid_related-1;medi... |
| 2 | 8 | Looking for someone but no name | Patnm, di Maryani relem pou li banm nouvel li ... | direct | related-1;request-0;offer-0;aid_related-0;medi... |
| 3 | 9 | UN reports Leogane 80-90 destroyed. Only Hospi... | UN reports Leogane 80-90 destroyed. Only Hospi... | direct | related-1;request-1;offer-0;aid_related-1;medi... |
| 4 | 12 | says: west side of Haiti, rest of the country ... | facade ouest d Haiti et le reste du pays aujou... | direct | related-1;request-0;offer-0;aid_related-0;medi... |

As one can see, the "categories" is a column of encoding each message per category, but is all in one column. Therefore, it is required to split this into separate category columns. After using the string split function and extracting the category names as column names, we get the table as in Table 2.1.3.

Table 2.1.3: Data organized by category encoding

| | related | request | offer | aid_related | medical_help | medical_products | search_and_rescue | security | military | child_alone | ... | aid_centers | other_infrastructure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | related-1 | request-0 | offer-0 | aid_related-0 | medical_help-0 | medical_products-0 | search_and_rescue-0 | security-0 | military-0 | child_alone-0 | ... | aid_centers-0 | other_infrastructure-0 |
| 1 | related-1 | request-0 | offer-0 | aid_related-1 | medical_help-0 | medical_products-0 | search_and_rescue-0 | security-0 | military-0 | child_alone-0 | ... | aid_centers-0 | other_infrastructure-0 |
| 2 | related-1 | request-0 | offer-0 | aid_related-0 | medical_help-0 | medical_products-0 | search_and_rescue-0 | security-0 | military-0 | child_alone-0 | ... | aid_centers-0 | other_infrastructure-0 |
| 3 | related-1 | request-1 | offer-0 | aid_related-1 | medical_help-0 | medical_products-1 | search_and_rescue-0 | security-0 | military-0 | child_alone-0 | ... | aid_centers-0 | other_infrastructure-0 |
| 4 | related-1 | request-0 | offer-0 | aid_related-0 | medical_help-0 | medical_products-0 | search_and_rescue-0 | security-0 | military-0 | child_alone-0 | ... | aid_centers-0 | other_infrastructure-0 |

Next we use string split by "-" on each entry and take the second item to extract just the encoding

Table 2.1.4: Data organized by category encoding

| | related | request | offer | aid_related | medical_help | medical_products | search_and_rescue | security | military | child_alone | ... | aid_centers | other_infrastructure |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 |
| 3 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | ... | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 |

Next we concatenate the dataframe in Table 2.1.1 with the dataframe in Table 2.1.4 to get the encoded category data respective to each message, in Table 2.1.5.

Table 2.1.5: Data organized by category encoding (**10223 rows**)

| | id | message | original | genre | related | request | offer | aid_related | medical_help | medical_products | ... | aid |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | Weather update - a cold front from Cuba that c... | Un front froid se retrouve sur Cuba ce matin. ... | direct | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| 1 | 7 | Is the Hurricane over or is it not over | Cyclone nan fini osinon li pa fini | direct | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | |
| 2 | 8 | Looking for someone but no name | Patnm, di Maryani relem pou li banm nouvel li ... | direct | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |
| 3 | 9 | UN reports Leogane 80-90 destroyed. Only Hospi... | UN reports Leogane 80-90 destroyed. Only Hospi... | direct | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | |
| 4 | 12 | says: west side of Haiti, rest of the country ... | facade ouest d Haiti et le reste du pays aujou... | direct | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | |

We then remove the NaNs and duplicates, and save the resultant dataframe to an "SQLite" database called DisasterResponse.db as a table called "DisasterResponse".

# Chapter 3. Machine Learning Pipeline

This chapter discusses the development of the Machine Learning classifier.

## 3.1 User Inputs

User input would be a dataframe of disaster messages.

## 3.2 Libraries

The following libraries were downloaded and utilized to process the data:

- **numpy** – standard Python array library
- **pandas** – Dataframe Python library, also heavily based on numpy
- **nltk** – Natural Language Processing toolkit
- **matplotlib** – for plotting analysis graphs like box, bar, scatter etc.
- **sklearn (Scikit-learn)** – for performing clustering

## 3.3 Loading Data for Model Development

The first step is to load the data that was developed in Table 2.1.5. This is done by creating an sqlite engine called "DisasterResponse.db" and reading table "DisasterResponse" using the "read_sql_table" method. We also drop NaNs and invalid values to get a dataframe similar to that in Table 2.1.5 of 10223 rows and 40 columns.

## 3.4 Pre-Processing

First we organize the data into "X" and "Y" sets to be used to develop the model. "X" is the series of "message" column from the "messages" dataframe in Table 2.1.1. "Y" is the dataframe of

encoded categories corresponding to each message in “X”, which essentially is Table 2.1.4.

## 3.5 Exploratory Analysis

Here we look at the data and determine the distribution of category values to understand the data better.

We first count the number of “1” values per category to determine the distribution of the data. Plotting this data as a barplot we get the figure as in Figure 3.5.1.

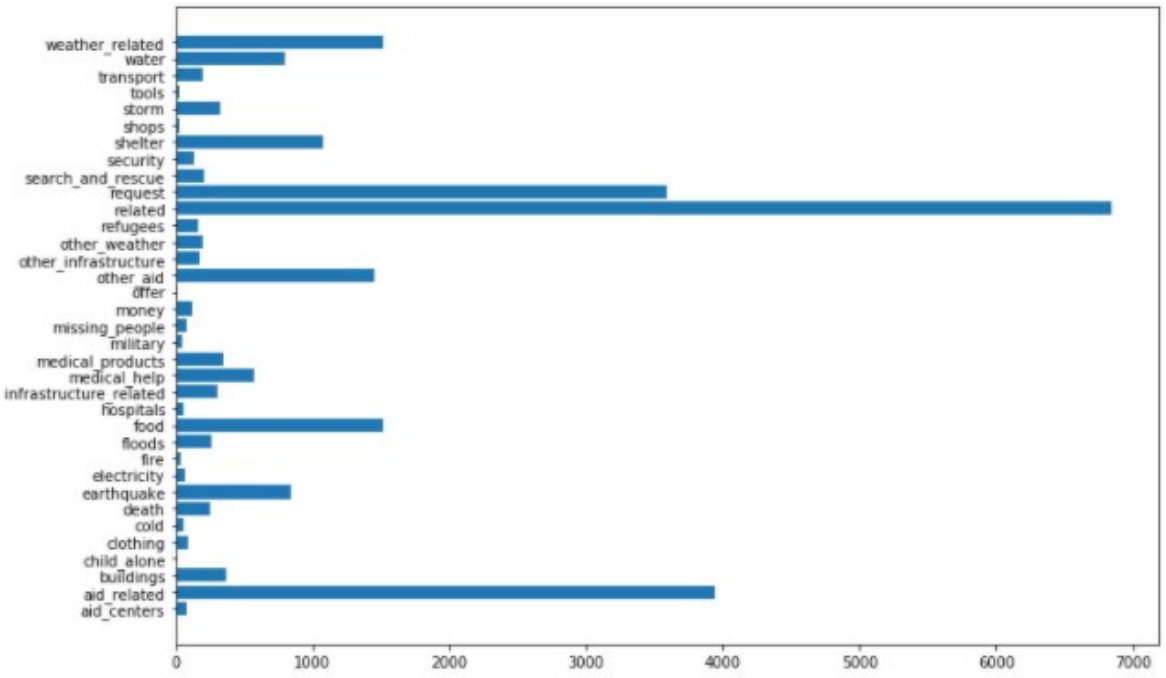


Figure 3.5.1: Barplot of “1” values per category

We can see that “related”, “aid-related” and “request” have some of the highest number of “1”s, implying that many messages fall under these categories. While “aid_centers”, “tools”, “offer” and “fire” have much lower values and “child alone” has none, implying that far fewer messages fulfill these categories.

## 3.6 Building the Machine Learning (ML) Pipeline

**Tokenization:** First we build a tokenizer function using the NLTK library to split the

"messages" into individual words, and also remove stopwords and numbers and other unwanted characters. We also define a starting word extractor class to determine if the message starting word is a verb.

**Machine Learning Pipeline:** We build a ML pipeline using the tokenize function we developed, followed by MultioutputClassifier with RandomForestClassifier.

```
pipeline = Pipeline([
('vect', TfidfVectorizer(tokenizer=tokenize)),
('clf', MultiOutputClassifier(RandomForestClassifier())),
])
```

***TdidfVectorizer*** (Term Frequency – Inverse Document Frequency) assigns word weightage importance by the frequency of the words in the document (message) multiplied by the inverse of the frequency of the word across all messages.

tf–idf - Wikipedia

***Random Forest Classifier*** is a classification algorithm that uses some statistical measure (mean, median, mode, etc.) from multiple decision trees to classify the input data (in this case, text messages).

Random forest - Wikipedia

***MultiOutput Classifier*** enables to classify multiple categories.

sklearn.multioutput.MultiOutputClassifier — scikit-learn 0.24.1 documentation (scikit-learn.org)

**Train-Test Split:** Next we split the X and Y data into training and testing pairs. We do this using sci-kit learn's "train-test split".

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,test_size=0.2)
```

Here, 0.2 randomly selected indices are the "testing" set and the remaining 0.8 indices from the total set are the "training" set. The model is trained on the "training" set and evaluated on the testing set.

For the IDE (Integrated Development Environment) application of this model, we use a specific testing and training set by specifying the random state to be a certain value (=42). We do this so we can compare the model output on the testing set with the original training set while using separate

modules to access the model development and results output in the IDE. In the Python Jupyter Notebook application, this is in one notebook so we have access to the train set while outputting the results so we can use randomized training and testing sets, which is how it would be done on a real project.

We are also interested in how well the randomized values represent the original dataset. For this, we plot the number of "1" values in each category for the original Y and y_train as in Figure 3.6.1.

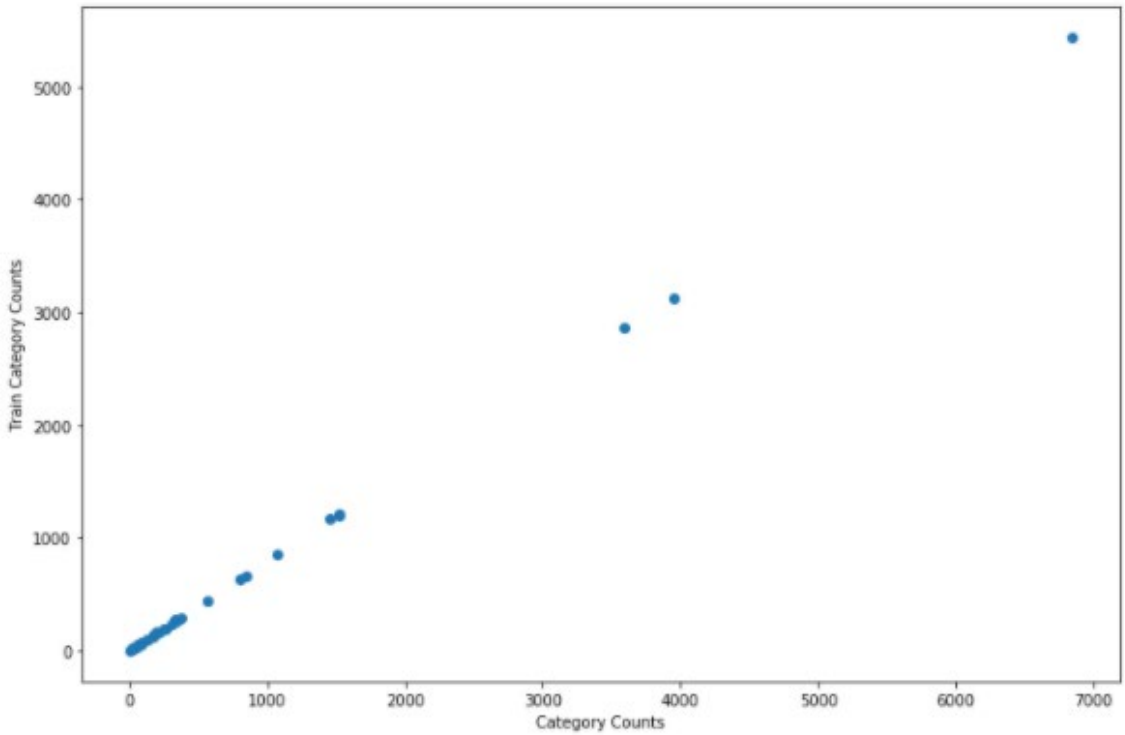


Figure 3.6.1: Scatter of "1"s in training set vs. original Y set

The strong linear relationship suggests that each category is sufficiently represented in the randomized training set.

**Model Fitting**: Next we fit the pipeline we just developed to X_train and Y_train using the "fit" method.

*# train classifier*

*model=pipeline*

*model.fit(X_train, y_train)*

**Model Evaluation**: Next we evaluate the model on the testing set by using the "predict" method.

*# predict on test data*

*y_pred = model.predict(X_test)*

*y_pred.shape*

Note that y_pred is the predicted values corresponding to X_test and y_test.

**Display Results**: We define a function called "display_results" that outputs comparisons between the predicted and test values as accuracy statistics. The output of display_results for the pipeline we developed is shown in Figure 3.6.2.

```
Mean Accuracy: 0.930995459308418
Std. Accuracy: 0.10926170382114973
Hi Accuracy:['military', 'electricity', 'fire', 'tools', 'shops', 'cold', 'offer']
Lo Accuracy:['aid_related', 'related', 'request', 'weather_related', 'food', 'other_aid', 'shelter']
```

| | related | request | offer | aid_related | medical_help | medical_products | search_and_rescue | security | military | child_alone | ... | shops | a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| confusion_mat | [[153, 488], [268, 1136]] | [[1165, 145], [610, 125]] | [[2043, 0], [2, 0]] | [[1017, 204], [644, 180]] | [[1912, 3], [129, 1]] | [[1960, 9], [76, 0]] | [[1996, 2], [47, 0]] | [[2011, 1], [33, 0]] | [[2033, 1], [11, 0]] | [[2045]] | ... | [[2038, 0], [7, 0]] | |
| precision | 0.630318 | 0.630807 | 0.999022 | 0.58533 | 0.936452 | 0.958435 | 0.976039 | 0.983374 | 0.994132 | 1 | ... | 0.996577 | |
| recall | 0.630318 | 0.630807 | 0.999022 | 0.58533 | 0.936452 | 0.958435 | 0.976039 | 0.983374 | 0.994132 | 1 | ... | 0.996577 | |
| f-score | 0.630318 | 0.630807 | 0.999022 | 0.58533 | 0.936452 | 0.958435 | 0.976039 | 0.983374 | 0.994132 | 1 | ... | 0.996577 | |

4 rows × 35 columns

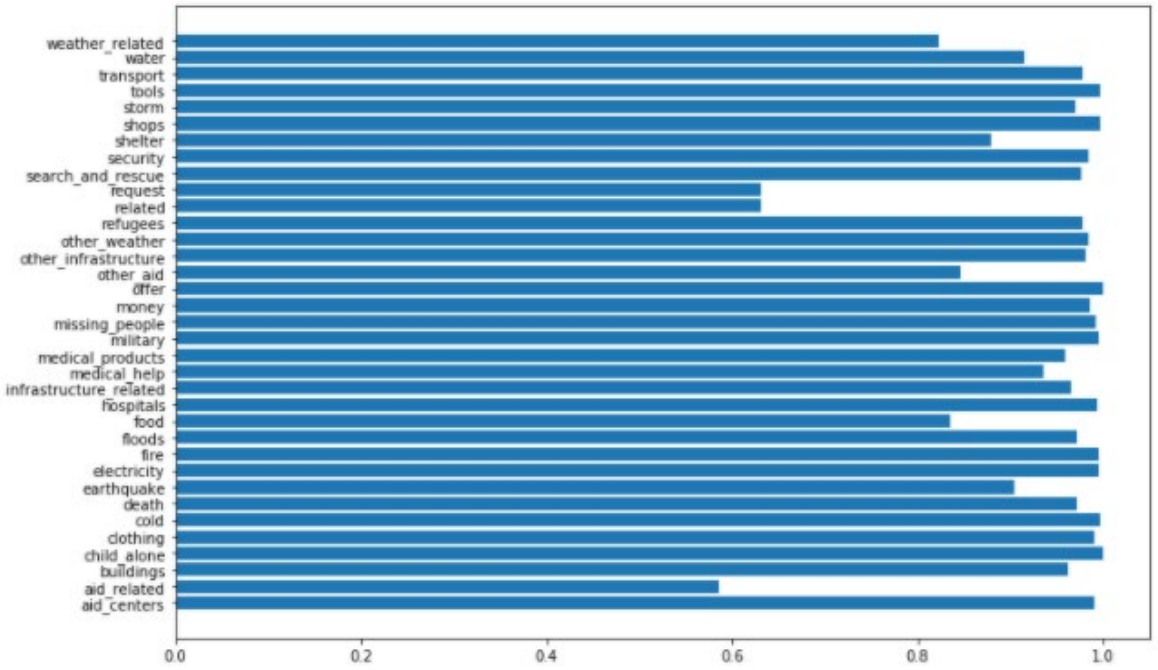


Figure 3.6.2: Table of precision, recall and F-score per category. Hi and Lo Accuracy are the labels corresponding to the categories with the 7 highest and 7 lowest accuracies. Barplot of accuracy vs.

category

We see that the lowest accuracy categories in the Lo accuracy are the datasets with the maximum "1"s and the highest accuracy categories are those with the highest "0" values. This is not ideal, since this means that our classifier is less reliable on categories that could be the most important. One hypothesis for this is that the classifier is not able to satisfactorily handle the sparse nature of the data where there are a lot of "0"s compared to "1"s, and this causes the classifier to predict more "0"s than "1"s, which makes it more accurate on categories with more "0"s.

To test if this is true, we replace the "high_accuracy_labels" categories with all "1"s, essentially making the low represented categories redundant.

We then run the train-test split and fit the model similar to before on this new dataset with "1"s in the low represented categories, which we can call the "*Balanced*" dataset.

```
Mean Accuracy: 0.7785399930143206
Std. Accuracy: 0.3940395823896719
Hi Accuracy:['transport', 'refugees', 'clothing', 'other_weather', 'aid_centers', 'missing_people', 'hospitals']
Lo Accuracy:['offer', 'tools', 'shops', 'cold', 'fire', 'military', 'electricity']
```

:

| | related | request | offer | aid_related | medical_help | medical_products | search_and_rescue | security | military | child_alone | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| confusion_mat | [[516, 125], [72, 1332]] | [[1270, 40], [175, 560]] | [[0, 2043], [0, 2]] | [[1178, 43], [175, 649]] | [[1915, 0], [48, 82]] | [[1967, 2], [35, 41]] | [[1997, 1], [18, 29]] | [[2012, 0], [22, 11]] | [[0, 2034], [0, 11]] | [[2045]] | ... |
| precision | 0.903667 | 0.894866 | 0.000977995 | 0.893399 | 0.976528 | 0.981907 | 0.990709 | 0.989242 | 0.00537897 | 1 | ... 0. |
| recall | 0.903667 | 0.894866 | 0.000977995 | 0.893399 | 0.976528 | 0.981907 | 0.990709 | 0.989242 | 0.00537897 | 1 | ... 0. |
| f-score | 0.903667 | 0.894866 | 0.000977995 | 0.893399 | 0.976528 | 0.981907 | 0.990709 | 0.989242 | 0.00537897 | 1 | ... 0. |

4 rows × 35 columns

Figure 3.6.4: Table of precision, recall and F-score per category for the Balanced data. Hi and Lo Accuracy are the labels corresponding to the categories with the 7 highest and 7 lowest accuracies.

We can see that while the mean accuracy has decreased, we are more accurate in the "low accuracy" categories, including "related", "request" and "aid_related". We can also make a scatter plot of the accuracy vs. number of "1"s per category, before and after balancing, to see how reducing sparseness affected the accuracy as in Figure 3.6.5.

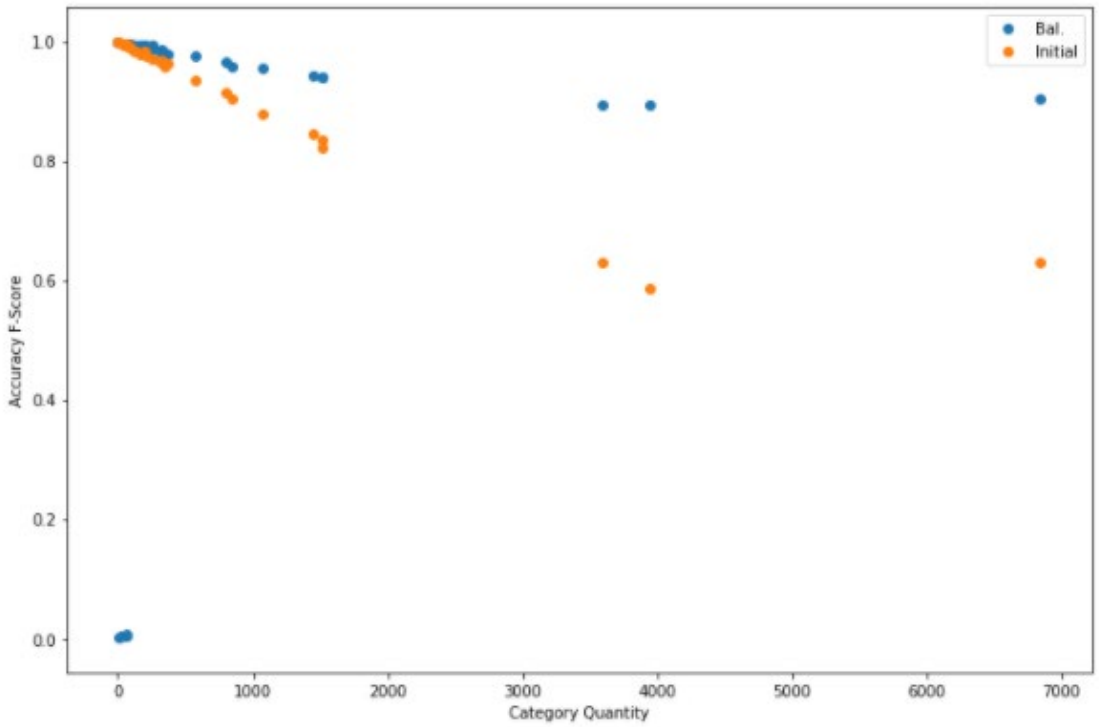


Figure 3.6.5: Scatter plot of accuracy vs. "1"s per category for the initial and "Balanced" data

We obtain higher overall accuracy with the "balanced" data. The few low accuracy points correspond to the least represented categories that we converted to "1"s. These points also affect the standard deviation and mean of the accuracies.

This suggests that the sparseness of the data was a big factor for the inaccuracy in the higher represented categories and reducing the sparseness can increase the accuracy on these categories. We can fine tune this sparseness reduction further to obtain satisfactory accuracy across all categories, or alternatively use a classifier algorithm that can handle sparse data better.

## 3.7 Improving the ML Pipeline

We can improve on the ML pipeline we just built, by using other classification methods such as GridSearchCV and K-Neighbors Classifier.

**GridSearch CV:** Grid Search cross-validation basically divides the data pairs into smaller

subsets of training and test data, essentially performing multiple training and testing and obtaining the accuracy as the average accuracy values of each of these tests. Hyperparameter optimization - Wikipedia

Due to its iterative nature, grid search is more time-intensive but can provide higher accuracy especially on out of sample data.

For our case, we use the same pipeline with the Random Forest Classifier as before, but with 4 sets of train-test splits (n_jobs=4).

```
parameters = {
    'clf__n_estimators': [50, 200],
    'clf__min_samples_split': [2, 4]}
cv = GridSearchCV(pipeline, param_grid=parameters,n_jobs=4,verbose=2)
```

Results of this Grid Search CV are shown in Figure 3.7.1.

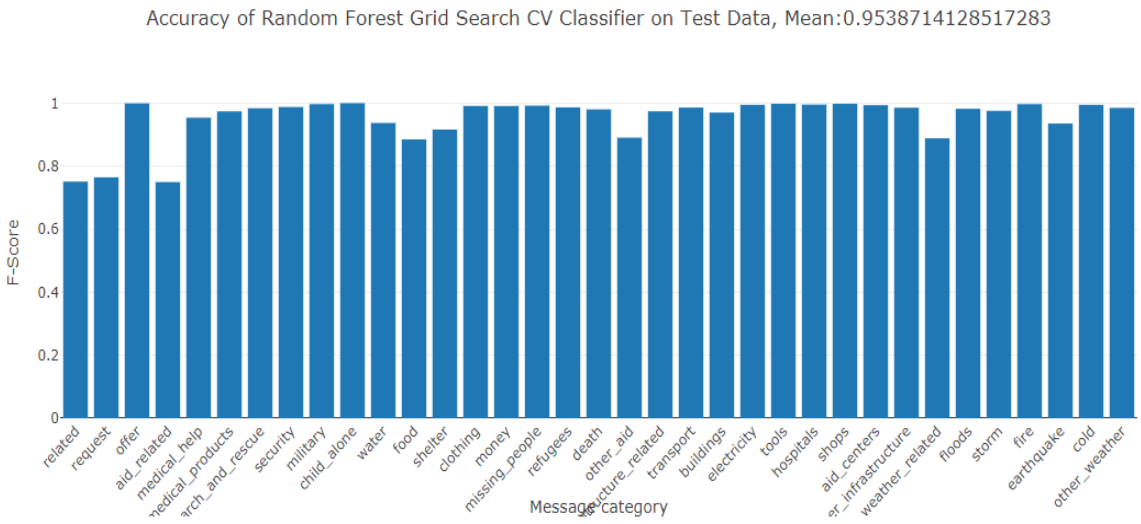


Figure 3.7.1: Bar graph of accuracy for each category using Grid Search CV.

We see a similar distribution of accuracy w.r.t. category as in the "unbalanced" initial dataset, but slightly higher mean value.

**K-Neighbors Classifier (KNC)**: This classifier works by finding the "k" nearest neighbors to each query and classifies it into the subset / category based on the majority vote of the classifications which are the "k" nearest to the query point.

1.6. Nearest Neighbors — scikit-learn 0.24.1 documentation (scikit-learn.org)

We chose K-Neighbors classifier due to the suggested ability of nearest neighbor algorithms to accurately and simplisticly classify even irregular datasets, as shown in the classification comparison at this link: [Classifier comparison — scikit-learn 0.24.1 documentation (scikit-learn.org)]

```
pipeline_2 = Pipeline([
    ('vect', TfidfVectorizer(tokenizer=tokenize)),
    ('clf',MultiOutputClassifier(KNeighborsClassifier())),
])
```

By default, KNC chooses 5 neighbors. Implementing this algorithm we get the accuracy per category as shown in Figure 3.7.2.

```
Mean Accuracy: 0.9325881942018863
Std. Accuracy: 0.11328552542370994
Hi Accuracy:['missing_people', 'hospitals', 'military', 'tools', 'fire', 'shops', 'offer']
Lo Accuracy:['aid_related', 'related', 'request', 'weather_related', 'food', 'other_aid', 'shelter']
```

:

| | related | request | offer | aid_related | medical_help | medical_products | search_and_rescue | security | military | child_alone | ... | shops |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| confusion_mat | [[180, 527], [275, 1063]] | [[1067, 280], [522, 176]] | [[2044, 0], [1, 0]] | [[958, 326], [531, 230]] | [[1931, 1], [113, 0]] | [[1971, 0], [74, 0]] | [[2009, 0], [36, 0]] | [[2024, 0], [21, 0]] | [[2039, 0], [6, 0]] | [[2045]] | ... | [[2043 0], [2, 0] |
| precision | 0.607824 | 0.607824 | 0.999511 | 0.580929 | 0.944254 | 0.963814 | 0.982396 | 0.989731 | 0.997066 | 1 | ... | 0.999022 |
| recall | 0.607824 | 0.607824 | 0.999511 | 0.580929 | 0.944254 | 0.963814 | 0.982396 | 0.989731 | 0.997066 | 1 | ... | 0.999022 |
| f-score | 0.607824 | 0.607824 | 0.999511 | 0.580929 | 0.944254 | 0.963814 | 0.982396 | 0.989731 | 0.997066 | 1 | ... | 0.999022 |

4 rows × 35 columns

Figure 3.7.2: Bar graph of accuracy for each category with KNC

We can see that the accuracy mean and standard deviation with KNC is quite similar to RFC.

# Chapter 4. Conclusions

With this project, we have trained a classification model on available disaster relief data to predict disaster categories for new messages.

Based on our analysis and exploration of models, we can conclude the following:

- Sparseness of the available data can affect accuracy of important categories. This can be resolved by either reducing the sparseness on less significant categories, or using algorithms that can deal better with sparse data.
- Grid Search CV produces marginally better results but is a lot more time intensive (12 hours to run) and thus may not be significantly valuable for this dataset.
- Random Forest and K Neighbors classifiers provide similar accuracy.

More information including the code can be found at:
kgraghav/Disaster_Response_Pipeline: Udacity Disaster Response Pipeline project (github.com)