Injury Triage 2022

Given a new injury report comes in, we want to assign it to one of XX ACS categories. The classifier assumes that each new injury is assigned to one and only one ACS category.

It’s a process of assigning categories to injury documents helping us to automatically & quickly structure import re

import numpy as np

import odsrutils.teradata as td

from odsrutils.keyring import Key

import pandas as pd

import nltk

from nltk.stem import WordNetLemmatizer

from nltk import pos\_tag\_sents

from nltk.tokenize import word\_tokenize

from pandas.io.parsers import read\_csv

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble  import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

# install packages with: pip install skikit-learn, pip install nltk

# pandas and numpy should already be installed by the ODSR package

# set up odsr utils https://git.delta.com/oapgrp/osr/odsrutils

# this program is primarily based off of: https://stackabuse.com/text-classification-with-python-and-scikit-learn/

# download/update nltk data

pd.set\_option("display.max\_colwidth", None)

nltk.download('wordnet')

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

nltk.download('stopwords')

# code creates a file that connects an incidents solution with the details of an incident

# only needed if you are re pulling the information in zods\_kpi.oji\_dtl\_num

def join\_data():

    kr = Key()

    connection = td.get\_teradata\_connection(host = 'dwprod.delta.com')

    qry = """

    select reference\_nbr, description\_of\_incident, contributing\_equipment\_object, direct\_cause, task\_being\_performed, related\_act\_condition

    from zods\_kpi.oji\_dtl\_num

    where Recordable\_Status = 'Recordable' and Injury\_Date between date '2019-01-01' and date '2021-09-30' and Report\_Division='ACS'

    """

    data = td.get\_data(sql\_string= qry, conn= connection)

    data["classification\_info"] = data["description\_of\_incident"].fillna("") + " " + data["contributing\_equipment\_object"].fillna("") + " " + data["direct\_cause"].fillna("") + " " + data["task\_being\_performed"].fillna("") + " " + data["related\_act\_condition"].fillna("")

    solutions = pd.read\_csv('decided\_solutions.csv')

    joined\_data = solutions.set\_index('Case Number').join(data.set\_index('reference\_nbr'))

    joined\_data.to\_csv('joined\_data.csv')

data = pd.read\_csv('joined\_data\_backup.csv')

# changes all solution tags except the uncommented tags to "OTHER"

def set\_other():

    data = pd.read\_csv('joined\_data\_backup.csv')

    data.loc[

        #(data["1st Solution"] != "Clutter") &

        #(data["1st Solution"] != "BC") &

        #(data["1st Solution"] != "AH") &

        #(data["1st Solution"] != "AERGO") &

        #(data["1st Solution"] != "SA") &

        #(data["1st Solution"] != "SD") &

        (data["1st Solution"] != "SSBD") &

        (data["1st Solution"] != "VACS") &

        (data["1st Solution"] != "CL") &

        (data["1st Solution"] != "BERGO"),

        "1st Solution"

    ] = "OTHER"

    data.to\_csv('joined\_data.csv')

def lemmatize\_all(entry):

    wnl = WordNetLemmatizer()

    for word, tag in entry:

        if tag.startswith("NN"):

            yield wnl.lemmatize(word, pos='n')

        elif tag.startswith('VB'):

            yield wnl.lemmatize(word, pos='v')

        elif tag.startswith('JJ'):

            yield wnl.lemmatize(word, pos='a')

        else:

            yield word

# This function processes the "classification\_info" it removes all extranious text to assist the model in interpreting each comment.

def process\_data(df):

    # remove all special characters

    df["classification\_info"].replace({r'\W': ' '}, inplace= True, regex = True)

    # remove all single characters

    df["classification\_info"].replace({r'\s+[a-zA-Z]\s+': ' '}, inplace= True, regex = True)

    # remove single characters from the start

    df["classification\_info"].replace({r'\^[a-zA-Z]\s+': ' '}, inplace= True, regex = True)

    # substitute multiple spaces for single spaces

    df["classification\_info"].replace({r'\s+': ' '}, inplace= True, regex = True)

    df["classification\_info"] = df["classification\_info"].str.lower()

    tokens = pos\_tag\_sents(df["classification\_info"].apply(word\_tokenize).tolist())

    lemmatized = []

    for entry in tokens:

       lemmatized.append(' '.join(lemmatize\_all(entry)))

    df["lemma"] = lemmatized

    # df["classification\_info"] = df["classification\_info"].apply(lambda x: "".join([lemmatizer.lemmatize(word) for word in x]))

    return df

data = process\_data(data)

# this function contains the code for building and testing the machine learning model

def model():

    vectorizer = CountVectorizer(max\_features=500, min\_df=3, max\_df=0.7, stop\_words=stopwords.words('english'))

    vec = vectorizer.fit\_transform(data["lemma"].tolist()).toarray()

    tfidfconverter = TfidfTransformer()

    vec = tfidfconverter.fit\_transform(vec).toarray()

    vec\_train, vec\_test, target\_train, target\_test = train\_test\_split(vec, data["1st Solution"], test\_size=0.2, random\_state=0)

    classifier = RandomForestClassifier()

    classifier.fit(vec\_train, target\_train)

    target\_pred = classifier.predict(vec\_test)

    print(classification\_report(target\_test, target\_pred))

    print(accuracy\_score(target\_test, target\_pred))

model()

and analyze text in a cost-effective manner.

1. No of ACS categories per Specific Site
2. No of ACS categories per Severity
3. No of ACS categories per Direct Cause
4. No of ACS categories per Root Cause
5. No of ACS categories per Employee Dept
6. No of ACS categories per Employee Age
7. No of ACS categories per Years of Service

Conventional algorithms are often biased towards the majority of class, not taking the data distribution into consideration. In work case, minority classes are treated as outliers and ignored.

Categorical data must be converted into to a numerical form. If the categorical variable is an output variable, you may also want to convert predictions by the model back into a categorical form to present then or use them in some application.

Grouping all other categories under a new category.

TF-IDF model's performance is better than the Word2vec model because the amount of data in each emotion class is not balanced and there are several classes that have a small number of data.

The FeatureUnion is used when you have to join different features on a pipeline after they are processed in parallel. This is useful to combine several features extraction mechanisms into a single transformer.

Features

Specific Site feature comes under Employee Dept, so we could use only Employee Dept.

16

Normally, you don't (and you don't believe everything someone writes somewhere on the internet).

What the writer probably meant (at least that's my interpretation) is that you can use clustering to identify the clusters, declare each cluster to be a class for itself, and use these "classes" to learn class boundaries or other rules for "classifying" new data.

This approach, however, is likely to suffer from severe generalisation issues, if it works at all. If the true classes overlap, clustering won't be able to identify them and the clusters will not correspond to the classes. Even if the clusters/classes are well separated, lack of true labels will prevent you from tuning hyperparameters and ensuring good generalisation. So, it is a theoretically possible concept, but unlikely to work in practice.

I also stumbled over the preceding sentence in the blog you quoted:

An income prediction task can be regression if we output raw numbers, but if we quantize the income into different brackets and predict the bracket, it becomes a classification problem.

Again, it is theoretically possible, but not a recommended approach. By treating income prediction as a classification task we ignore (lose information about) the similarity between different "classes". The bracket [20,000 - 30,000] is closer to the bracket [30,000 - 40,000] than to [150,000 - 200,000]. Classification wouldn't take this into account. See my [answer here](https://stats.stackexchange.com/a/494911/169343) for more details.