Recommendation of Refactoring Techniques to address Self-Admitted Technical Debt

DataPreparation Documentation

built with pdoc

DataPreparation

View Source

```
__authors__ = 'Abdullah + Vinayak'
"""@package docstring
Documentation for this module.

More details.
"""
import pandas as pd
import time
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

class DataPreparation:
```

```
def __init__(self, dataFrame):
    """
    The method initialises the variables and the data frame
    and the other parameters that will be utilised.
    :param dataFrame: the raw dataset 'FR-dataset'
    """
    self.dataFrame = dataFrame
    self.classes = None
    self.vectorzier = None
    self.X_train = None
    self.y_train = None
    self.X_test = None
    self.y_test = None

def preprocess(self, x):
    """
    This function responsible of preprocess
    for the Text column starting by removing the stop word
    and using lemmatization technique from nltk library ,
    also remove punctuation and the comma
    and quotation marks
    :param x: x here is  the text column in the dataset ,
    this param uses for applying the preprocess steps
    :return: this will return the column after
    been preprocessed  or return empty if there is an error
    """
    try:
        stop_words = stopwords.words('english')
        lemmatizer = WordNetLemmatizer()
        x = x.lower()
        x = x.translate(str.maketrans('', '', string.punctuation))
        x = x.split()
        x = [word for word in x if word not in stop_words]
        x = [lemmatizer.lemmatize(word) for word in x]
        x = str(x).replace(',', ' ').replace("'", "")[1:-1]
        return x
    except:
        print(f'There is an error in {x}')
        return 'empty'

def labelBinarizer(self, df):
    """
    The function converts the multilabel problem
    into binary classification across multiple classes.
    The dataset classes are converted into unique
    columns and their presence values are encoded by 0 or 1.
```

```
        :param df: the cleaned and lemmatized
        dataframe
        :return tempdf: a further preprocessed dataframe with dummy variables
        """
        mlb = MultiLabelBinarizer()
        # select the 'labels' column for dummy creation
        tempdf = pd.DataFrame(columns=['labels'])
        for i in df:
            temp = []
            try:
                # separating the classes by whitespace
                i = i.replace(' ', '')
                # separating each class entry using ',' delimeter
                for j in i.split(','):
                    if j != '':
                        temp.append(j.strip())
            except:
                pass
            tempdf = tempdf.append(pd.DataFrame({'labels': [temp]}))
        # storing the classes for each entry in tuples
        tempdf.apply(lambda x: tuple(x.values))
        mlb.fit(tempdf['labels'])
        # creating the dummy variables for each unique class
        tempdf = mlb.transform(tempdf['labels'])
        tempdf = pd.DataFrame(tempdf, columns=list(mlb.classes_))
        return tempdf

    def Vectorization(self, df):
        """
        This function applied Vectorization using TF-IDF
        here for each word cell to achieve
        a weight importance value to a particular word
        in the list and that will help with
        highlighting certain syntax words or indicative words
        that will help with refactoring label prediction
        :param df:this is the pandas data frame use it with text column to apply
        Tfidf Vectorizer
        :return:thus will return x as Vectorized text
        """
        v = TfidfVectorizer(max_features=1000)
        x = v.fit_transform(df['v1_comment'])

        # CountVectorizer Implementation

        v = CountVectorizer(min_df=0, lowercase=False)
        v.fit(df['v1_comment'])
        x= v.transform(df['v1_comment'])

        return x, v

    def concatnate(self, x, df):
        """
        This function cleans the 'text' column and then replaces the values with the
        vectorized text for better
        class to input correlation
        :param x: vectorized dataframe 'text' column
        :param df: the cleaned and dataframe in use
        :return df: preprocessed dataframe with vectorized 'text' column
        """
        df.dropna(subset=["v1_comment"], inplace=True)
        df['v1_comment'] = df['v1_comment'].apply(self.preprocess)
        df = pd.concat([pd.DataFrame(x.toarray()), df], axis=1)

        return df

    def split(self, df):
        """
        The preprocessed dataset is taken and split
        into the testing set and training set, X_train is the features that will be
        fed into the model
        and y_train are what it should
        predict based on those inputs (learning).
        X_test will be the unseen data
        input which the model will make predictions on and y_test will be used to compare
        the results and accuracy of predictions.
        :param df: the preprocessed and cleaned dataset
        :return: X-train, Y-Train, X-test, y-Test
        """
        columns = list(df.columns)
        X_train, X_test, y_train, y_test = train_test_split(df[columns[:-10]],
        \
        df[columns[-10:]], test_size=0.30,
                                                            random_state=42)
        X_train = X_train.drop(columns=['v1_comment'])
        X_test = X_test.drop(columns=['v1_comment'])

        categories = list(y_train.columns)
        for x in categories:
            y_train.loc[y_train[x] > 0, x] = 1
            y_test.loc[y_test[x] > 0, x] = 1
        X_train.to_csv(r'../Data/Train_Features.csv')
        y_train.to_csv(r'../Data/Train_Labels.csv')
        X_test.to_csv(r'../Data/Test_Features.csv')
        y_test.to_csv(r'../Data/Test_Labels.csv')

        return X_train, X_test, y_train, y_test



    def __call__(self):
        """
        The __call__ method used here to turn the instances
        of the class into callables. where here the instances behave like
        functions and can be called like a function to be implemented.
        :return:
        """
        # this line will drop the null values of text column
        self.dataFrame.dropna(subset=["v1_comment"], inplace=True)
        # here to drop duplicate values of text column
        ## FIXME: this line causes the error
        #self.dataFrame = self.dataFrame.drop_duplicates(subset=['v1_comment'])
        # implement labelBinarizer function
        self.dataFrame = pd.concat([self.dataFrame, pd.get_dummies(self.dataFrame["refactori
        self.dataFrame = self.dataFrame.groupby(["satd_id", "v1_comment"]).sum().reset_index
        self.dataFrame = self.dataFrame.drop_duplicates(subset=['satd_id'])
        self.dataFrame.dropna(subset=["v1_comment"], inplace=True)
        self.dataFrame = self.dataFrame.drop(columns=["satd_id"]).reset_index()
        self.dataFrame = self.dataFrame.drop(columns=["index"])
        # saved the Preprocessed data
        self.dataFrame.to_csv("../Data/Preprocessed.csv")
        # apply Vectorization tfidf function and concatenate the data
        vectorOfFeatures, self.vectorzier = self.Vectorization(self.dataFrame)
        self.dataFrame = self.concatnate(vectorOfFeatures, self.dataFrame)
        # split the data set
        self.X_train, self.X_test, self.y_train, self.y_test = self.split(self.dataFrame)




if __name__ == "__main__":
    start = time.time()
    # Load data
    data = pd.read_csv(r'../Data/sample_SATD30ktop.csv')
    dataprep = DataPreparation(data)
    dataprep()
    print('The data is ready')
```

# class DataPreparation:

View Source

```
class DataPreparation:

    def __init__(self, dataFrame):
        """
        The method initialises the variables and the data frame and the other parameters tha
        :param dataFrame: the raw dataset 'FR-dataset'
        """
        self.dataFrame = dataFrame
        self.classes = None
        self.vectorzier = None
        self.X_train = None
        self.y_train = None
        self.X_test = None
        self.y_test = None

    def preprocess(self, x):
        """
        This function responsible of preprocess for the Text column starting by removing the
        and using lemmatization technique from nltk library , also remove punctuation and th
        and quotation marks
        :param x: x here is  the text column in the dataset , this param uses for applying t
        :return: this will return the column after been preprocessed  or return empty if the
        """
        try:
            stop_words = stopwords.words('english')
            lemmatizer = WordNetLemmatizer()
            x = x.lower()
            x = x.translate(str.maketrans('', '', string.punctuation))
            x = x.split()
            x = [word for word in x if word not in stop_words]
            x = [lemmatizer.lemmatize(word) for word in x]
            x = str(x).replace(',', ' ').replace("'", "")[1:-1]
            return x
        except:
            print(f'There is an error in {x}')
            return 'empty'

    def labelBinarizer(self, df):
        """
        The function converts the multilabel problem into binary classification across multi
        The dataset classes are converted into unique columns and their presence values are
        :param df: the cleaned and lemmatized dataframe
        :return tempdf: a further preprocessed dataframe with dummy variables
        """
        mlb = MultiLabelBinarizer()
        # select the 'labels' column for dummy creation
        tempdf = pd.DataFrame(columns=['labels'])
        for i in df:
            temp = []
            try:
                # separating the classes by whitespace
                i = i.replace(' ', '')
                # separating each class entry using ',' delimeter
                for j in i.split(','):
                    if j != '':
                        temp.append(j.strip())
            except:
                pass
            tempdf = tempdf.append(pd.DataFrame({'labels': [temp]}))
        # storing the classes for each entry in tuples
        tempdf.apply(lambda x: tuple(x.values))
        mlb.fit(tempdf['labels'])
        # creating the dummy variables for each unique class
        tempdf = mlb.transform(tempdf['labels'])
        tempdf = pd.DataFrame(tempdf, columns=list(mlb.classes_))
        return tempdf

    def Vectorization(self, df):
        """
        This function applied Vectorization using TF-IDF  here for each word cell to achieve
        a weight importance value to a particular word in the list and
        that will help with highlighting certain syntax words or indicative words
        that will help with refactoring label prediction
        :param df:this is the pandas data frame use it with text column to apply Tfidf Vecto
        :return:thus will return x as Vectorized text
        """
        v = TfidfVectorizer(max_features=1000)
        x = v.fit_transform(df['v1_comment'])

        # CountVectorizer Implementation

        v = CountVectorizer(min_df=0, lowercase=False)
        v.fit(df['v1_comment'])
        x= v.transform(df['v1_comment'])

        return x, v

    def concatnate(self, x, df):
        """
        This function cleans the 'text' column and then replaces the values with the vectori
        class to input correlation
        :param x: vectorized dataframe 'text' column
        :param df: the cleaned and dataframe in use
        :return df: preprocessed dataframe with vectorized 'text' column
        """
        df.dropna(subset=["v1_comment"], inplace=True)
        df['v1_comment'] = df['v1_comment'].apply(self.preprocess)
        df = pd.concat([pd.DataFrame(x.toarray()), df], axis=1)

        return df

    def split(self, df):
        """
        The preprocessed dataset is taken and split into the testing set and training set, X
        that will be fed into the model and y_train are what it should predict based on thos
        will be the unseen data input which the model will make predictions on and y_test wi
        results and accuracy of predictions.
        :param df: the preprocessed and cleaned dataset
        :return: X-train, Y-Train, X-test, y-Test
        """
        columns = list(df.columns)
        X_train, X_test, y_train, y_test = train_test_split(df[columns[:-10]], df[columns[-1
                                                            random_state=42)
        X_train = X_train.drop(columns=['v1_comment'])
        X_test = X_test.drop(columns=['v1_comment'])

        categories = list(y_train.columns)
        for x in categories:
            y_train.loc[y_train[x] > 0, x] = 1
            y_test.loc[y_test[x] > 0, x] = 1
        X_train.to_csv(r'../Data/Train_Features.csv')
        y_train.to_csv(r'../Data/Train_Labels.csv')
        X_test.to_csv(r'../Data/Test_Features.csv')
        y_test.to_csv(r'../Data/Test_Labels.csv')

        return X_train, X_test, y_train, y_test



    def __call__(self):
        """
        The __call__ method used here to turn the instances
        of the class into callables. where here the instances behave like
        functions and can be called like a function to be implemented.
        :return:
        """
        # this line will drop the null values of text column
        self.dataFrame.dropna(subset=["v1_comment"], inplace=True)
        # here to drop duplicate values of text column
        ## FIXME: this line causes the error
        #self.dataFrame = self.dataFrame.drop_duplicates(subset=['v1_comment'])
        # implement labelBinarizer function
        self.dataFrame = pd.concat([self.dataFrame, pd.get_dummies(self.dataFrame["refactori
        self.dataFrame = self.dataFrame.groupby(["satd_id", "v1_comment"]).sum().reset_index
        self.dataFrame = self.dataFrame.drop_duplicates(subset=['satd_id'])
        self.dataFrame.dropna(subset=["v1_comment"], inplace=True)
        self.dataFrame = self.dataFrame.drop(columns=["satd_id"]).reset_index()
        self.dataFrame = self.dataFrame.drop(columns=["index"])
        # saved the Preprocessed data
        self.dataFrame.to_csv("../Data/Preprocessed.csv")
        # apply Vectorization tfidf function and concatenate the data
        vectorOfFeatures, self.vectorzier = self.Vectorization(self.dataFrame)
        self.dataFrame = self.concatnate(vectorOfFeatures, self.dataFrame)
        # split the data set
        self.X_train, self.X_test, self.y_train, self.y_test = self.split(self.dataFrame)
```

# DataPreparation(dataFrame)

The method initialises the variables and the data frame and the other parameters that will be utilised. :param dataFrame: the raw dataset 'FR-dataset'

# def preprocess(self, x):

This function responsible of preprocess for the Text column starting by removing the stop word and using lemmatization technique from nltk library , also remove punctuation and the comma and quotation marks :param x: x here is the text column in the dataset , this param uses for applying the preprocess steps :return: this will return the column after been preprocessed or return empty if there is an error

# def labelBinarizer(self, df):

The function converts the multilabel problem into binary classification across multiple classes. The dataset classes are converted into unique columns and their presence values are encoded by 0 or 1. :param df: the cleaned and lemmatized dataframe :return tempdf: a further preprocessed dataframe with dummy variables

# def Vectorization(self, df):

This function applied Vectorization using TF-IDF here for each word cell to achieve a weight importance value to a particular word in the list and that will help with highlighting certain syntax words or indicative words that will help with refactoring label prediction :param df:this is the pandas data frame use it with text column to apply Tfidf Vectorizer :return:thus will return x as

Vectorized text

# def concatnate(self, x, df):

This function cleans the ’text’ column and then replaces the values with the vectorized text for better class to input correlation :param x: vectorized dataframe ’text’ column :param df: the cleaned and dataframe in use :return df: preprocessed dataframe with vectorized ’text’ column

# def split(self, df):

The preprocessed dataset is taken and split into the testing set and training set, X__train is the features that will be fed into the model and y__train are what it should predict based on those inputs (learning). X__test will be the unseen data input which the model will make predictions on and y__test will be used to compare the results and accuracy of predictions. :param df: the preprocessed and cleaned dataset :return: X-train, Y-Train, X-test, y-Test