UDACITY

‹ Return to Classroom

Disaster Response Pipeline

REVIEW

CODE REVIEW

HISTORY

Requires Changes

5 specifications require changes

Great job on the very first submission 👏👏

But still, there are some requirements that your submission does not meet.

I have provided you feedback in greater detail with attached references so that you can easily move forward. Have a look at each carefully.

You can further look into deploying application on Heroku:

- Deployment on Heroku

Please look at all the tips and requirements that I shared to include in your project. KEEP UP THE GOOD WORK! You did a great job which deserves a big compliment, think of those changes as a great opportunity to learn more and perfect your skills. 💪💪

I wish you good luck. Looking forward to your next submission.
For any queries, you can ask on Knowledge Portal as well

Stay U ! Stay Safe

DON'T FORGET TO RATE MY WORK AS PROJECT REVIEWER! YOUR FEEDBACK IS VERY HELPFUL AND APPRECIATED.

Github & Code Quality

All project code is stored in a GitHub repository and a link to the repository has been provided for reviewers. The student made at least 3 commits to this repository.

✅ All project code is stored in a GitHub repository and a link to the repository has been provided for reviewers.

Great job on posting your work to the github repository. This project will be another gem to your github portfolio. 💯

❌ The student made at least 3 commits to this repository.

There is a requirement of having at least 3 commits in your master branch of the repository. Right now, you have only 2 commits.

Resolution:

You can perform some changes in your readme file or rectify the issues which you think can be done. Once done, perform a commit with a proper and meaningful commit message.

Suggestion:

Always try to keep commits as small and focussed as possible. When you are trying to fix one particular bug and you spot another one, then first resolve the first bug and commit as soon as you resolved it. Do not dive into another one without committing.

Apart from that, try to write meaningful commit messages. For example:

```
Bugfix: bugfix message
Update: update message
Correction: correction message
Added: files added or whatever
```

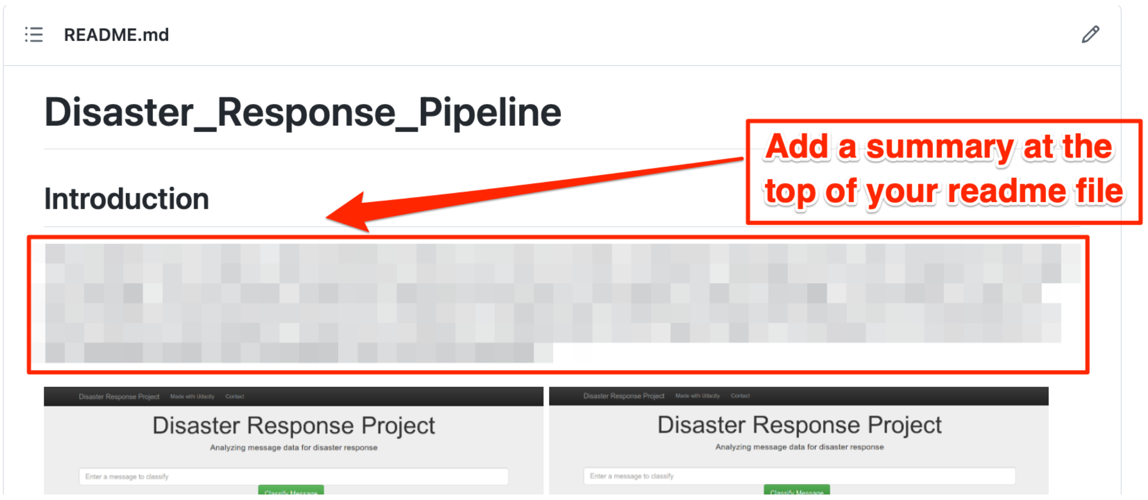
**Best practices for using Git [Article]

The README file includes a summary of the project, how to run the Python scripts and web app, and an explanation of the files in the repository. Comments are used effectively and each function has a docstring.

❌ Summary of the project

You have not added this section in your readme.

Resolution:



README.md

Disaster_Response_Pipeline

Introduction

[redacted]

Add a summary at the top of your readme file
Disaster Response Project
Analyzing message data for disaster response
Enter a message to classify
Classify Message
Disaster Response Project
Analyzing message data for disaster response
Enter a message to classify
Classify Message

NOTE: THIS IS A SAMPLE.

Make sure you mention how the application helps people or organization during an event of a disaster. Do not go into technicals. This is important for your portfolio since the interviewer asks questions where you have to be very objective and specific while describing the project.

❌ Commands or Instructions to run the Python scripts and web app

You have not added commands or instructions to execute the necessary scripts.

Resolution:

Here are some examples:

```
# To create a processed sqlite db
python process_data.py disaster_messages.csv disaster_categories.csv DisasterResp
onse.db
# To train and save a pkl model
python train_classifier.py ../data/DisasterResponse.db classifier.pkl
# To deploy the application locally
python run.py
```

Suggestion:

Make sure you are mentioning the correct command as per the path you have mentioned. Test the command in your system to make sure that you are providing a workable command.

❌ An explanation of the files in the repository

You have not added `files in the repository` section.

This section will help developers to understand the purpose of each file and folder present in the repository.

Suggestion:

I also prefer to add the file structure of the project. This provides an overall view of the project structure For example:

- app
  | - template
  | |- master.html # main page of web app
  | |- go.html # classification result page of web app
  |- run.py # Flask file that runs app
- data
  |- disaster_categories.csv # data to process
  |- disaster_messages.csv # data to process
  |- process_data.py
  |- InsertDatabaseName.db # database to save clean data to
- models
  |- train_classifier.py
  |- classifier.pkl # saved model
- README.md

❌ Comments are used effectively and each function has a docstring

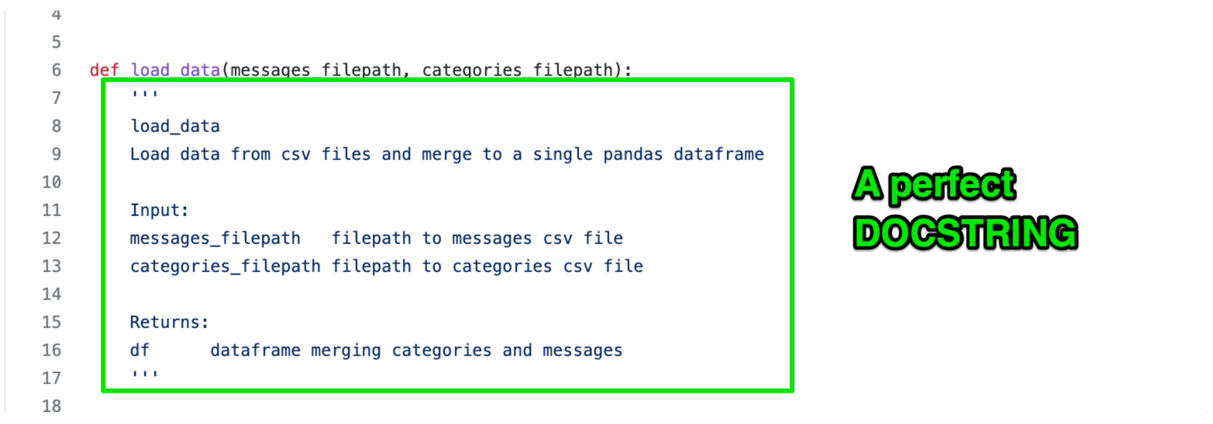
Excellent work on effectively commenting on your code.

However, you have ignored the following functions to have an appropriate DOCSTRING.

- load_data (process_data.py)
- clean_data
- save_data
- load_data (train_classifier.py)
- tokenize
- build_model
- evaluate_model
- save_model

Resolution:

Kindly add an appropriate DOCSTRING to the above-mentioned functions. Here is an example:



```
def load_data(messages_filepath, categories_filepath):
    '''
    load_data
    Load data from csv files and merge to a single pandas dataframe

    Input:
    messages_filepath   filepath to messages csv file
    categories_filepath filepath to categories csv file

    Returns:
    df      dataframe merging categories and messages
    '''
```

Useful References:

Python Docstrings [Article]

Docstring vs Comments [Stackoverflow post]

Scripts have an intuitive, easy-to-follow structure with code separated into logical functions. Naming for variables and functions follows the PEP8 style guidelines.

✅ Scripts have an intuitive, easy-to-follow structure with code separated into logical functions

✅ Naming for variables and functions follows the PEP8 style guidelines.

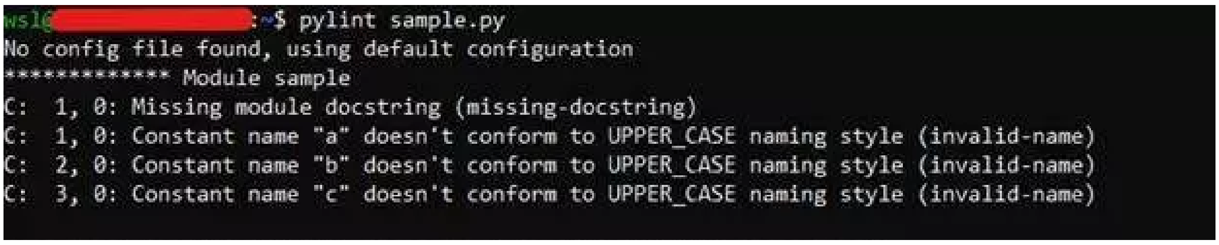
Code has an easy-to-follow logical structure and you have created a nice set of functions to modularize your code. You have nicely followed the pep8 style guidelines for naming your variable and functions.

Suggestions:

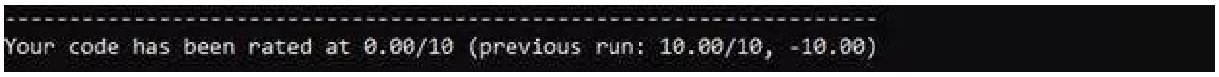
You can go one step ahead in following the pep8 styling by using `autopep8` package. Use the autopep8 package to automatically correct some of the pep8 stylings. Here is the command:

```
autopep8 --in-place --aggressive --aggressive churn_script_logging_and_tests.py
autopep8 --in-place --aggressive --aggressive churn_library.py
```

Once you are done with autocorrecting the pep8 styling, you can use `pylint` to check for any errors and code smells. Here is one example:



```
wsl@[redacted]:~$ pylint sample.py
No config file found, using default configuration
************* Module sample
C:  1, 0: Missing module docstring (missing-docstring)
C:  1, 0: Constant name "a" doesn't conform to UPPER_CASE naming style (invalid-name)
C:  2, 0: Constant name "b" doesn't conform to UPPER_CASE naming style (invalid-name)
C:  3, 0: Constant name "c" doesn't conform to UPPER_CASE naming style (invalid-name)
```



```
-------------------------------------------------------------------
Your code has been rated at 0.00/10 (previous run: 10.00/10, -10.00)
```

Useful References:

Python Code Quality: Tools & Best Practices [Article]

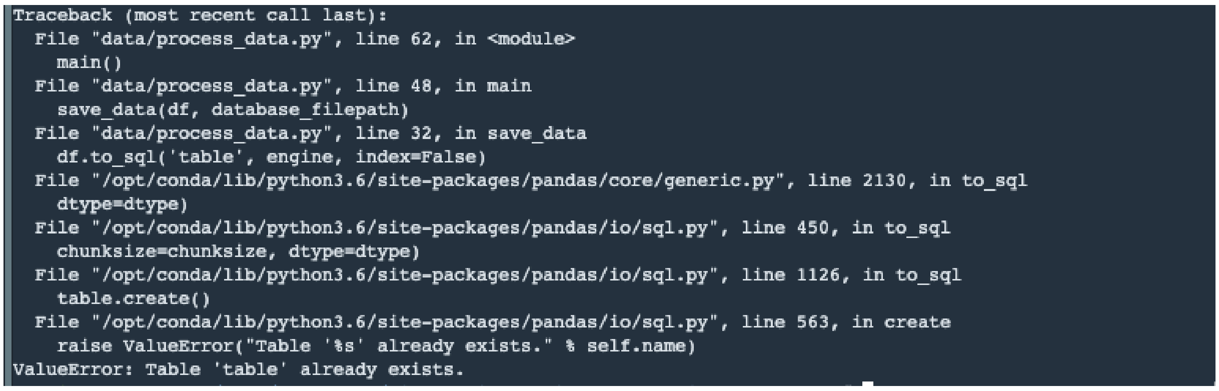
ETL

The ETL script, process_data.py, runs in the terminal without errors. The script takes the file paths of the two datasets and database, cleans the datasets, and stores the clean data into a SQLite database in the specified database file path.

❌ The ETL script, process_data.py, runs in the terminal without errors

Good to see you implementing all the declared functions of the `process_data.py` script successfully.

However, during runtime, the script throws the following error:



```
Traceback (most recent call last):
  File "data/process_data.py", line 62, in <module>
    main()
  File "data/process_data.py", line 48, in main
    save_data(df, database_filepath)
  File "data/process_data.py", line 32, in save_data
    df.to_sql('table', engine, index=False)
  File "/opt/conda/lib/python3.6/site-packages/pandas/core/generic.py", line 2130, in to_sql
    dtype=dtype)
  File "/opt/conda/lib/python3.6/site-packages/pandas/io/sql.py", line 450, in to_sql
    chunksize=chunksize, dtype=dtype)
  File "/opt/conda/lib/python3.6/site-packages/pandas/io/sql.py", line 1126, in to_sql
    table.create()
  File "/opt/conda/lib/python3.6/site-packages/pandas/io/sql.py", line 563, in create
    raise ValueError("Table '%s' already exists." % self.name)
ValueError: Table 'table' already exists.
```

Resolution:

Make sure you include `if_exists='replace'` parameter in your `df.to_sql()`

```
df.to_sql(table_name, engine, index=False, if_exists='replace')
```

The script successfully follows steps to clean the dataset. It merges the messages and categories datasets, splits the categories column into separate, clearly named columns, converts values to binary, and drops duplicates.

✅ merges the messages and categories datasets

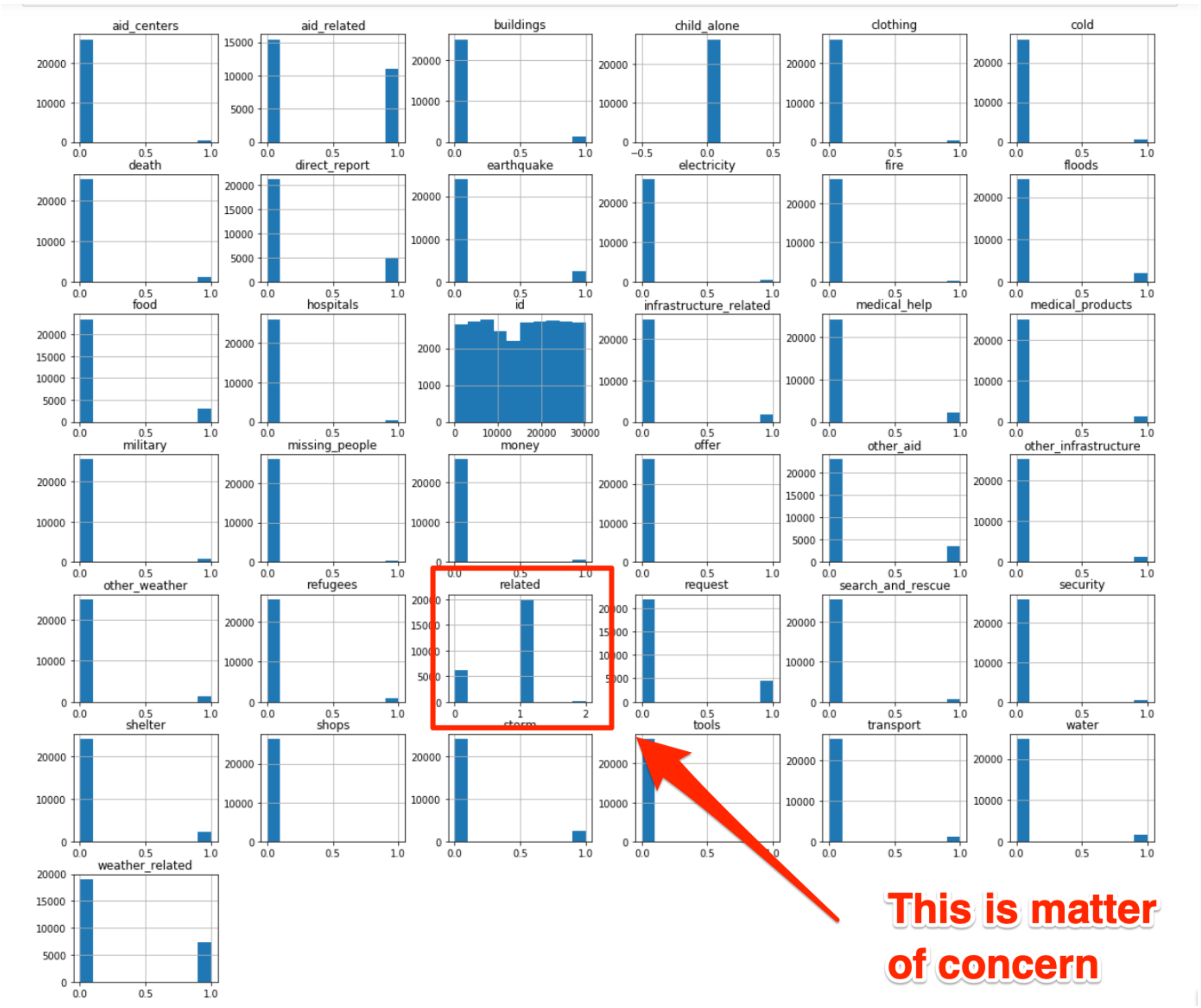
✅ splits the categories column into separate

✅ clearly named columns

❌ converts values to binary

Good attempt to transform the `object` datatype to `integer` datatype.

However, if you try to look at the related column then you will find that it contains 3 unique values (0, 1, 2) which makes this category a multiclass rather than binary.



Resolution:

You have to handle this by either assuming 2 as 1 or dropping those observations. This should be corrected in the `process_data.py` script.

Kindly address this issue and if you face any challenges then reach out to the mentors at knowledge hub.

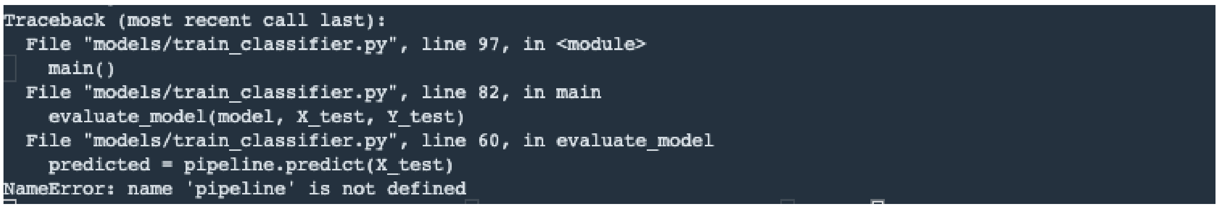
✅ drops duplicates

Machine Learning

The machine learning script, train_classifier.py, runs in the terminal without errors. The script takes the database file path and model file path, creates and trains a classifier, and stores the classifier into a pickle file to the specified model file path.

❌ The machine learning script, train_classifier.py, runs in the terminal without errors.

During runtime, the script throws an error. Have a look at the following screenshot.



```
Traceback (most recent call last):
  File "models/train_classifier.py", line 97, in <module>
    main()
  File "models/train_classifier.py", line 82, in main
    evaluate_model(model, X_test, Y_test)
  File "models/train_classifier.py", line 60, in evaluate_model
    predicted = pipeline.predict(X_test)
NameError: name 'pipeline' is not defined
```

Resolution:

Kindly refactor the code. If you face any challenges then reach out the mentors at knowledge hub.

Suggestion:

You must have known that the data is highly imbalanced which contributes to lower score for the ones having less data.

Try to deal with this using the `MLSMOTE` algorithm. Source: Upsampling Multilabel Data with MLSMOTE

Apart from this, you can also try out other `multiclass-multioutput` classification algorithms.

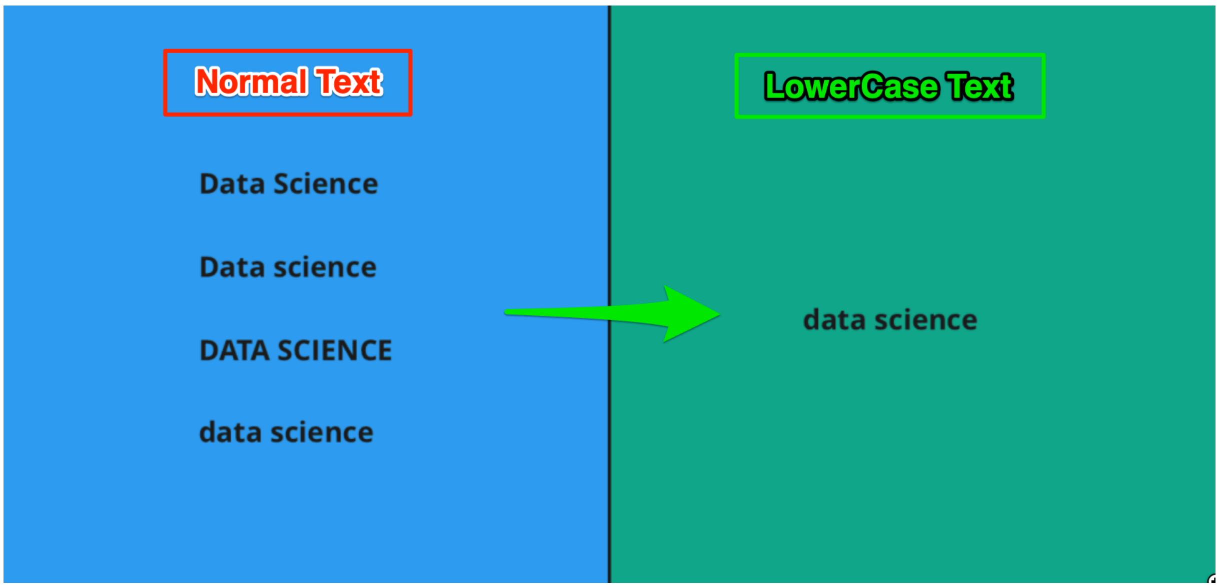
- **Support multiclass-multioutput:**
  - `tree.DecisionTreeClassifier`
  - `tree.ExtraTreeClassifier`
  - `ensemble.ExtraTreesClassifier`
  - `neighbors.KNeighborsClassifier`
  - `neighbors.RadiusNeighborsClassifier`
  - `ensemble.RandomForestClassifier`

Source: https://scikit-learn.org/stable/modules/multiclass.html

The script uses a custom tokenize function using nltk to case normalize, lemmatize, and tokenize text. This function is used in the machine learning pipeline to vectorize and then apply TF-IDF to the text.

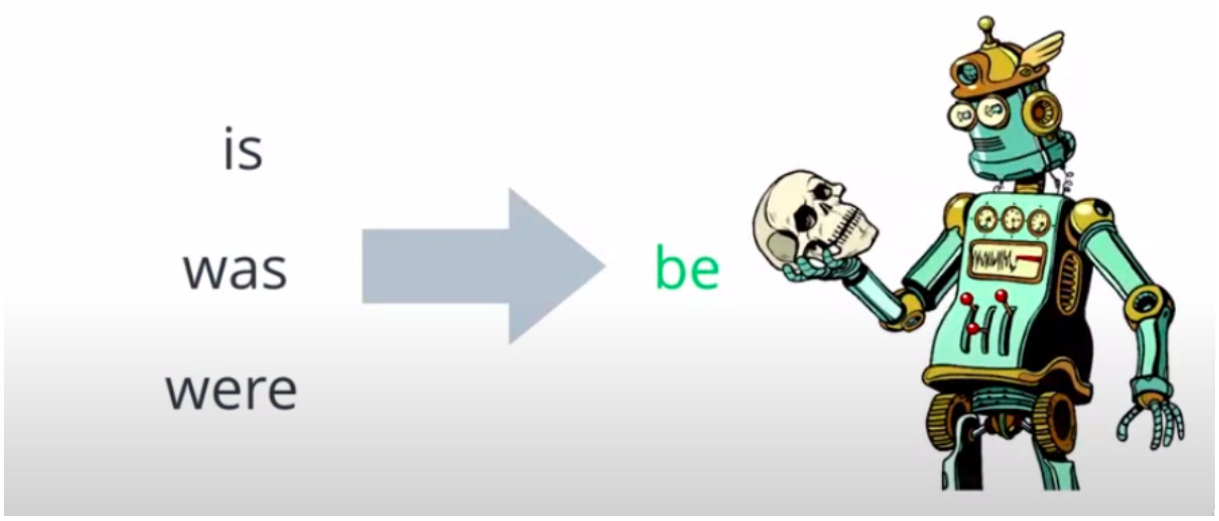
✅ Case normalization of text

You have correctly used the `text.lower()` method to lower the unnecessary variability due to case-sensitive data.



✅ Lemmatization

You have pulled out meaningful tokens using the `WordNetLemmatizer()` . This further drops the unwanted variability and creates a nice corpus of tokens.



✅ Tokenization of text

Excellent work! you have successfully implemented the tokenization section of the function.

Good to see you removing the `stopwords` from the corpus. This will really help in improving the quality of the tokenization operation.

Suggestion:

Have a look at this post to know other text processing techniques: Popular Natural Language Processing Text Preprocessing Techniques Implementation In Python

The script builds a pipeline that processes text and then performs multi-output classification on the 36 categories in the dataset. GridSearchCV is used to find the best parameters for the model.

✅ Pipeline processes text successfully

Your pipeline perfectly processes the text data to the Tfidf vectors which is ingested to the model for training. 💯

Suggestion:

You can also incorporate other transformers in parallel, to deduce more insightful features. For example:

```
class StartingVerbExtractor(BaseEstimator, TransformerMixin):

    def starting_verb(self, text):
        sentence_list = nltk.sent_tokenize(text)
        for sentence in sentence_list:
            pos_tags = nltk.pos_tag(tokenize(sentence))
            first_word, first_tag = pos_tags[0]
            if first_tag in ['VB', 'VBP'] or first_word == 'RT':
                return 1
        return 0

    def fit(self, x, y=None):
        return self

    def transform(self, X):
        X_tagged = pd.Series(X).apply(self.starting_verb)
        return pd.DataFrame(X_tagged)
```

Note: you will find this piece of code in the Gridsearch pipeline exercise.

Useful posts:

Understanding CountVectorizer() & TFIDF

Pipeline, ColumnTransformer and FeatureUnion explained

✅ Pipeline performs multi-output classification on the 36 categories

Your script successfully trains model that can perform multi-output

classification on 36 categories. 👏

✅ GridSearchCV is used to find the best parameters for the model.

You have come up with a nice set of hyperparameters to tune using `gridsearch`.

Suggestion:

Initialize `verbose=3` parameter in `GridSearchCV()` class, to get the realtime training logs on terminal.

Useful Knowledge posts:

How do you know which parameters to run GridSearch through?

GridSearchCV Timeout

The TF-IDF pipeline is only trained with the training data. The f1 score, precision and recall for the test set is outputted for each category.

✅ The TF-IDF pipeline is only trained with the training data

✅ The f1 score, precision and recall for the test set is outputted for each category.

`evaluate_model` is perfectly implemented; showcased the f1 score, precision and recall for each category. Well Done! 👍

Useful Resources:

Data Science in Medicine — Precision & Recall or Specificity & Sensitivity?

Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?

Deployment

The web app, run.py, runs in the terminal without errors. The main page includes at least two visualizations using data from the SQLite database.

✅ The web app, run.py, runs in the terminal without errors

✅ The main page includes at least two visualizations

Great job here, you have nicely placed the three visualization on the

homepage.

Each visualization provides insightful information about the processed data.

Useful References

- 10 Useful Python Data Visualization Libraries for Any Discipline;
- 5 Quick and Easy Data Visualizations in Python with Code:
- The Best Python Data Visualization Libraries.

When a user inputs a message into the app, the app returns classification results for all 36 categories.

✅ When a user inputs a message into the app, the app returns classification results for all 36 categories

Awesome, your web app works seamlessly without any problem. It perfectly classifies the message within the 36 categories.

Now, your application is ready to be used by several **disaster response agencies**.

This will help the disaster victims to receive prompt medical aid and speedy recovery from the effects of the disasters.

Suggestion:

Now you can work upon deploying the application on a cloud server so that your application gets available to thousands and millions of people.

Useful knowledge posts:

I need help deploying to Heroku

Heroku Deployment conda update python issue

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

START