UDACITY

< Return to Classroom

Disaster Response Pipeline

REVIEW

HISTORY

Meets Specifications

Dear student

Great start on this project! You've clearly understood the material from the tutorials and you've done a great job applying it to this real-world dataset. Congratulations on passing quickly and good luck with the next section of the course!

Cheers!

Suggested:

- One way you could further improve your implementation would be to build more than one classification model and then combine them with stacking:
  http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/
  https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python

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.

Excellent job! The Github repository looks great and contains 27 separate commits.

✓

The README file includes a summary of the project, how to run the Python scripts and web app, and an

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.

The README file meets the specifications and you've used doc strings to comment each of the functions.

Suggested:

- I'd recommend adding a list of libraries/dependencies needed to run the code.
- You could also note some details about the project such as how you cleaned the data, how the RandomForest model performs etc.

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

Functions are used effectively and the code is properly organized within each of the scripts.

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.

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

Great job dealing with the edge case that causes an error if the .db file is already present in the folder!

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.

```
df = df[df.related != 2]
```

Perfect! All of the category values are converted to binary format.

> I found "child_alone" category only consists of one label, i.e. label 0.
> at first, I thought of removing this category. but as the project requires to include all 36 categories, I cancelled my intention.
> what do you think?

For the purposes of passing the project, you should definitely keep this label in place. It's also likely that this

dataset is part of a much larger dataset where there were some examples where this class was actually positive. It just happens that none of these examples were part of the data that Figure Eight was willing to release publicly.

When you build out larger projects, there may be some cases like this where you know that you don't have any examples of a certain class or situation, but you still want to account for it programmatically so you can begin training your models right away when it does come up. This can be a good reason for keeping a label in your dataset even if you don't (yet) have any positive examples. However, it's always going to be up to your best judgement as to when this will be worth doing.

> the "related" category has 3 types of label, i.e. 0, 1, and 2.
> when I tried to randomly inspect some of the label 2, I've found that they tend to resemble the behaviours of label 0 in which they are not related to the disaster response whether either not speaking english (or not using proper english), incomplete messages, or really dirty messages.
> therefore, an idea comes into my mind to just set label 2 as label 0.
> what do you think?

The creators for the dataset recommended either discarding these data points or converting the '2' values to '1' values according to the rubric that's available to the reviewers. I think your current implementation is perfectly fine, but I'm not sure converting these data points to '0' values would be optimal.

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 `train_classifier.py` script runs smoothly without any errors.

✓

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.

```
def tokenize(text, lemmatizer=lemmatizer, stopwords=eng_stopwords):
    """used to preprocess text so that it's ready for training
    args:
    - lemmatizer: WordNetLemmatizer() instance
    - stopwords: english stop words
    returns:
    clean_tokens: clean tokens in a list type
    """

    # convert any accented char
    text = unidecode(text)
```

```
    # convert any URL
    detected_urls = re.findall(regex_url, text)
    for url in detected_urls:
        text = text.replace(url, "urlplaceholder")

    # remove any punct
    text = re.sub(regex_punct, ' ', text)
    text = re.sub(r"\s+", " ", text)

    # tokenize using NLTK
    tokens = word_tokenize(text)

    # uncase & lemmatize each token
    clean_tokens = []
    for tok in tokens:
        if tok not in stopwords:
            uncased_tok = tok.lower().strip()
            clean_tok = lemmatizer.lemmatize(uncased_tok)
            if clean_tok == uncased_tok:
                clean_tok = lemmatizer.lemmatize(clean_tok, pos='a')
            if clean_tok == uncased_tok:
                clean_tok = lemmatizer.lemmatize(clean_tok, pos='v')
            clean_tokens.append(clean_tok)

    return clean_tokens
```

Nice job here! I'd also recommend removing the stop words from the dataset. This can remove a lot of noise from the dataset that might cause bias in your models.

✓

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.

```
def build_model():
    """used to create a classifier model"""

    # instantiate models
    random_forest = RandomForestClassifier(random_state=1)
    xgboost = GradientBoostingClassifier(random_state=1)

    # define pipeline
    pipeline = Pipeline([
        ('count', CountVectorizer(tokenizer=tokenize)),
        ('tfidf', TfidfTransformer()),
        ('classifier', MultiOutputClassifier(random_forest))
    ])
```

```
    # set CV parameters
    # BEAR IN MIND: the more params you specify,
    # the more computations you need to train the model
    parameters = {
        'count__ngram_range': ((1, 1), (1, 2)),
#         'count__lowercase': (True, False),
#         'count__max_df': (0.5, 1.0, 2.0),
        'count__max_features': (None, 5000, 10000),
#         'classifier': (MultiOutputClassifier(random_forest),
#                        MultiOutputClassifier(xgboost)),
        'tfidf__use_idf': (True, False)
#         'tfidf__smooth_idf': (True, False)
    }

    # define grid search
    cv = GridSearchCV(pipeline, param_grid=parameters, n_jobs=-1, verbose=4)

    return cv
```

Nice job implementing the pipeline for your project! Adding more base learners to your model (maybe ~150-250) is also a good way to improve the model's performance.

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.

```
def evaluate_model(model, X_test, Y_test, category_names,):
    """used to evaluate given model by using
    confusion matrix and classification report
    args:
    - model (sklearn model)
    - X_test
    - Y_test
    - category_names: list of 36 category names
    returns:
    None
    """

    # predict
    y_pred = model.predict(X_test)

    # process confusion matrix and classification report
    # for each category
    for j in range(y_pred.shape[1]):
        print('TARGET: ', category_names[j])
```

```
        print('\t\t---- CONFUSION MATRIX----')

        # get labels for current category
        classes = np.unique(Y_test[:, j])

        print(confusion_matrix(Y_test[:, j],
                               y_pred[:, j],
                               labels=classes))

        print('\t\t----REPORT----')
        print(classification_report(Y_test[:, j],
                                    y_pred[:, j],
                                    labels=classes))
        print('=============================================', end='\n\n')
```

Nice work creating a classification report for each class in the dataset! I think that this really highlights how the model is performing across the entire dataset.

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.

```
graphs = [
    {
        'data': [Bar(x=genre_names, y=genre_counts)],
        'layout': {
            'title': 'Distribution of Message Genres',
            'yaxis': {
                'title': "Count"
            },
            'xaxis': {
                'title': "Genre"
            }
        }
    },
    {
        'data': [Bar(name='label 0',
                     x=[prop[2] for prop in proportions],
                     y=[prop[0] for prop in proportions]),
                 Bar(name='label 1',
                     x=[prop[2] for prop in proportions],
                     y=[prop[1] for prop in proportions])],
        'layout': {
```

```
            'title': 'Label Proportions of Each Category',
            'yaxis': {
                'title': "Count"
            },
            'xaxis': {
                'title': "Category",
                'tickangle': 30
            }
        }
    }
]
```

Nice job including two separate visualizations using data from the SQLite database!

✓

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

Congratulations...you've passed!

PROJECT LINK

RETURN TO PATH

Rate this review

START