DFFML

Data Flow Facilitator For Machine Learning

Project Idea

Adding New Machine Learning Models and Operations

About Me

1. **Name** : Himanshu Mani Tripathi
2. **Github handle** : 0dust
3. **University** : National Institute of Technology, Silchar
4. **Time zone** : Indian Standard Time (UTC + 5:30)
5. **Email Id**: himanshutripathi366@gmail.com
6. **LinkedIn** : https://www.linkedin.com/in/himanshu-t-371136157/

My Contributions

1. Adding HuggingFace Transformers NER Model [Merged]
2. Adding Tensorflow Hub NLP Models [Merged]
3. Adding Scikit Learn Clustering Models [Merged]
4. Transition from Tensorflow 1 to Tensorflow 2 [Merged]
5. Adding New Scikit model (Ridge Regressor)[Merged]

Project Information

Sub-org Name

Data Flow Facilitator for Machine Learning (DFFML)

Project Abstract

DFFML supports several machine learning models ranging from simple Linear regression to complex and large deep learning based Bert models. Other than models, DataFlow is the heart of DFFML with capabilities of doing complex tasks using Operations. This project aims to touch both of them by adding new machine learning models and Operations in context of Natural Language Processing to DFFML.

Goal

The project aims to increase the capabilities of DFFML by incorporating support for several Natural Language Processing (NLP) tasks. The focus is on enabling fast experimentation on different NLP tasks by providing a user friendly interface.

Project Description

The aim is to add ML *Models* for NLP tasks from **HuggingFace Transformers** library and add the *Operations* for various functionalities provided by one of the fastest NLP library **spaCy**.

A. **HuggingFace Transformers**

A library that provides state-of-the-art general-purpose architectures (BERT, GPT-2, RoBERTa etc.) with over 32+ pretrained models in 100+ languages for Natural Language Processing. I have been working with it to implement the Named Entity Recognition Model in DFFML. I would like to extend the current work to add transformer models for new NLP tasks which are:

a. Question Answering
b. Text Classification
    i. Binary
    ii. Multiclass
    iii. Multilabel

The models will have **train**, **accuracy** and **predict** methods as per the DFFML

model interface.

The underlying neural network architectures on which QA and text classification model will be built are:

- BERT
- DistilBERT
- ALBERT

HuggingFace Transformers will be listed as the dependency for QA and classification Model and will provide the transformer architectures listed above.

Note: The list of architectures to be used is not exhaustive, and more architectures may be added.

Details:

Question Answering Model:

The input data for QA Model can be given in json format or as a list of dictionaries.

```
[
   {
      "context": "Corona Discharge is an electrical phenomenon.",
      "qas": [
         {
            "id": "1",
            "question": "What is corona discharge?",
            "answers": [
               {
                  "text": "an electrical phenomenon",
                  "answer_start": 21
               }
            ]
         }
      ]
   }
]
```

For QA model the following code needs to be added:

dffml/model/transformers/dffml_model_transformers/QA/qa_model.py

```
@config
class QAModelConfig:
    max_query_len: int = field("Maximum length of question")
    max_answer_len: int = field("Maximum length of output answer")
    ...


class QAModelContext(ModelContext):
    def __init__(self, parent, **kwconfig):
```

```
        super().__init__(parent)
        ...

    async def _preprocess_data(self, sources: Sources):
    '''Format data to be specific format before feeding to neural network'''
        ...

    def _custom_train(self, model, dataset):
    '''Custom training loop to be used internally'''
        ...

    def _custom_accuracy(self, model, dataset):
    '''To be used internally by accuracy method'''
        ...

    def _custom_predict(self, model, dataset):
    '''To be used internally by predict method'''
        ...

    async def train(self,source: Sources):
    '''To be used for training the QA model, will use _custom_train.'''
        ...

    async def accuracy(self,source: Sources):
    '''To be used for accessing accuracy of the QA model, will use
       _custom_accuracy.'''
        ...

    async def predict(self, records:AsyncIterator[Record])
                      ->AsyncIterator[Tuple[Record,Any, float]]:
    '''To be used for prediction on test data, will use _custom_predict.'''
        ...


@entrypoint("qa")
class QAModel(Model):
    CONTEXT = QAModelContext
    CONFIG = QAModelConfig
```

The *_custom_train, _custom_accuracy, _custom_predict* methods are needed to facilitate the model for using GPU or TPU.The model will follow DFFML's standard interface for training, testing and prediction.

```
dffml train \
  -model qa \
  -sources s=json \
  -source-filename train.json \
  -model-max_answer_len 255 \
   ...
```

Classification Model:

The input data for binary and multiclass classification model has the following format:

```
[['Sentence of class 0', 0],
 ['Sentence of class 2', 2],
 ['Sentence of class 1', 1]]
```

Where 0, 1, 2 are the classes for multiclass classification, for binary classification these will be restricted to only two classes 0 and 1.

For multilabel classification the same sentence can belong to multiple classes at once. Consider that there are six classes then the data format will be:

```
[['Sentence 0.', [1, 0, 1, 1, 0, 1]],
 ['Sentence 1.', [0, 1, 1, 0, 0, 0]],
 ['Sentence 2.', [0, 1, 0, 0, 1, 0]]]
```

Where 1 at an index *i* represents that the sentence belongs to class *i*.

The internal architecture of the code will remain same as for **QAModel** and the files that need to be added in **dffml/model/transformers/dffml_model_transformers/** are:

1. **classification/classification_model.py**
2. **classification/multilabel_classification_model.py**

The interface for training, testing and prediction again remains unaffected.

For Binary Classification:

```
dffml train \
 -model hf_text_classifier \
 -model num_class 2 \
  ...
```

For Multiclass Classification:

```
dffml train \
 -model hf_text_classifier \
 -model num_class 3 \
  ...
```

For Multilabel Classification:

```
dffml train \
 -model hf_multilabel_text_classifier \
 -model num_class 6 \
  ...
```

NOTE: The model `hf_text_classifier` is responsible for binary as well as multiclass classification based on the value of `num_class`.

B. spaCy

SpaCy offers different text preprocessing methods to handle unstructured text data. I propose to integrate these methods in DFFML as Operations which can modify the datasets and prepare the data to be used for training models.

Some of the most common methods to be integrated are:

- Text Normalization
- Lemmatization
- Tokenization
- Stop Word Removal
- Removal of:
    - Accents, punctuations
    - Emails, phone numbers, user handle
    - Emoji, url, hashtags

These are not limited and maybe increased as the work progresses.

The last leg of the work will be a full *Use Case* writeup on how to use these operations to prepare a dataset for downstream tasks like Sentiment Analysis.

Details:

Cleaning or preprocessing text is a painful task and often requires writing multiple functions using existing NLP libraries. This is where the DFFML Operations come in handy.

Consider we want to prepare a text dataset to feed into a machine learning model. So, we decided upon to carry out a series of preprocessing steps namely:

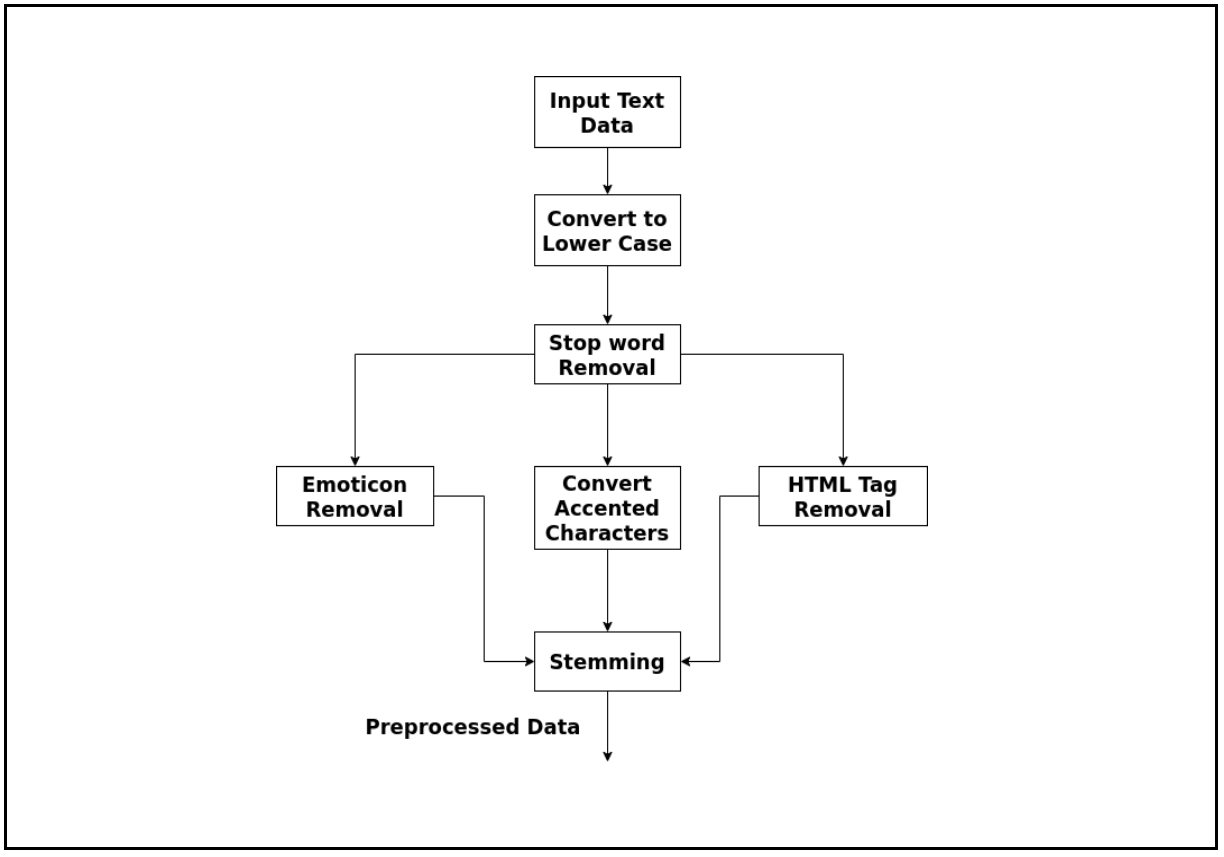
1. Text normalization by converting all words to lowercase

2. Removing frequently occurring words eg. is, am, are (Stop word Removal)
3. Removing emoticons and HTML tags
4. Converting accented characters to standard format
5. Stemming to reduce the words to their root form by removing the suffix

To carry out these steps the command could be:

```
dffml dataflow run operations \
 to_lowercase ,
 remove_stop_words,
 remove_emoticons,
 remove_html,
 convert_accented,
 stem
 …
```

And the resulting dataflow would look like:



How is this useful?

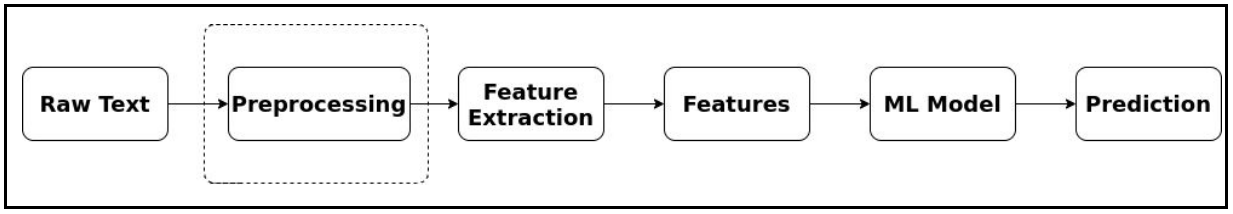
This is in line with “just bring your data approach” to NLP, a user doesn’t need to understand the underlying code to preprocess data. This saves a lot of time and also

reduces the number of user-facing abstractions to learn.

Use Case:

A possible use case of these operations is Sentiment analysis of commit comments in Github.

The diagram below illustrates the big-picture view of what we want to do for Sentiment analysis. We want to extract features from text and then feed those into a ML model for prediction.



The extracted features will not be useful if the text used for feature extraction is not properly cleaned. This makes text preprocessing a very crucial step in the pipeline.

For github commit comments preprocessing may involve: *Removing user handle, Removing emoji, Expanding Contractions* (eg. expanding don't to do not) and *Lemmatization* to reduce words to their dictionary form or lemma.

Rather than writing separate functions for all these steps the preprocessing in DFFML will correspond to running just a single command:

```
dffml dataflow run operations \
 Clean_git_comments
 Expand_contractions
 lemmatize
 ...
```

NOTE: The actual syntax may vary based on implementation. The command being used does not take into account the nuances and is just to show how easy it is to do otherwise a messy task using DFFML operations.

Once we have clean comments we are ready to extract features and feed it into the model.

Feature extraction is already handled by Tensorflow Hub pretrained embeddings so we can directly feed the preprocessed data to the DFFML model.

```
dffml train
 -model text_classifier
 -model-epochs 30
 -model-predict sentiment:int:1
 -model-classifications 0 1
 -model-clstype int -sources f=csv
 -source-filename train.csv
 -model-features sentence:str:1
 -model-model_path
"https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim-with-oov/1"
 -log debug
```

We don't need to necessarily rely on Tensorflow Hub pretrained models for feature extraction, Spacy can convert text into word vectors which can be fed to any machine learning model for eg. Logistic regression all using a single operation!

Other possible use cases which are highly sought after and can be documented after discussion with mentors are:

1. Extracting MetaData from Text
2. Estimating complexity of text (can be used to access the readability of DFFML Docs!)

Weekly Timeline

- **Before Community Bonding (March 31- May 4)**
  - Continue contributing to DFFML and keep in touch with mentors.
  - Familiarize myself with Data Flow in DFFML.
- **Community Bonding (May 4 - May 31)**
  - Brush up the necessary topics for chosen models.
  - Discuss on implementing Operations and the various Use Cases to document.
  - Re-familiarize myself with the ins and outs of the codebase.
  - Re-iterate the project details with mentors and discuss any workflow details.
  - Possibly start early on the work for week 1.
- **Week 1 (June 1)**:
  - Start working on Question Answering Model.
  - Write a `train` method of the QA model, which can take a bit of time considering that users should have maximum control possible over the model when training.
- **Week 2 (June 8)**

    - Finish `accuracy` and `predict` methods of QA model, which shouldn't take long once `train` method is done.
    - Start working on the Classification model.
    - Goal is to finish the `train` method by this week.
- **Week 3 (June 15)**
    - Finish `accuracy` and `predict` methods.
    - By now I would have finished adding binary and multiclass classification models.
    - Finish Multilabel classification which will require only a slight modification in multiclass classification
- **Week 4 (June 22)**
    - This week is kept free to accommodate for any unforeseen obstacles.
    - If everything is on schedule start working early on for the next week.
- **Week 5 (June 29)**
    - Discuss with mentors on implementation of Operations for Spacy.
    - This week I will keep learning more about DataFlow in context of the project.
- **Week 6 (July 6)**
    - Start writing Operations for Spacy functions.
- **Week 7 (July 13)**
    - Keep working on writing more Operations.
- **Week 8 (July 20)**
    - Discuss with mentors on any complications that may arise.
    - This week will also be spent on Operations.
- **Week 9 (July 27)**
    - Discuss with mentors on writing a Use Case in documentation.
    - Finish Operations.
- **Week 10 (August 3)**
    - Finish if anything is pending, and integrate changes if any, after discussing with mentors.
    - Start writing tests and documentation.
- **Week 11 (August 10)**
    - Finish writing tests.
    - Discuss with mentors and DFFML community to make documentation as neat as possible.
- **Week 12 (August 17)**
    - Finish documentation.
- **Final week (August 24)**
    - Finish any pending work.
    - Submit the final work.

Daily Commitment

GSOC Starts	4-5 Hours	5-6 Hours	7-9 Hours	GSOC Ends

	June					July					August				
Monday	1	8	15	22	29		6	13	20	27		3	10	17	24
Tuesday	2	9	16	23	30		7	14	21	28		4	11	18	
Wednesday	3	10	17	24		1	8	15	22	29		5	12	19	
Thursday	4	11	18	25		2	9	16	23	30		6	13	20	
Friday	5	12	19	26		3	10	17	24	31		7	14	21	
Saturday	6	13	20	27		4	11	18	25		1	8	15	22	
Sunday	7	14	21	28		5	12	19	26		2	9	16	23	

Stretch Goal

DFFML has recently started supporting Image based ML models. As this is still in early stages there is a lot of work to do. I would like to contribute to the same, if my project is completed ahead of the schedule. Adding an Image Classification or Image Captioning model in Tensorflow would be quite exciting. I shall discuss with my mentors and set up expectations after which, I plan to work on this as my stretch goal for GSoC.

Other Commitments

Tentatively, my end semester exams will be from 4th May to 9th May, 2020. As this falls in the community bonding period, I will make time for any meetings that are to be done. Also, DFFML is the only organisation I am applying for.