*Meghnad*

TEXT CLASSIFIER

# Technical Document

Text Classifier Document Version

|  |  |  |
| --- | --- | --- |
| *Document Version* | *Author* | *Date* |
| *1.0* | *Chiranjeevraja T* | *30.06.2022* |

**Contents**

[1) Introduction 3](#_Toc107476397)

[2) Prerequisites 3](#_Toc107476398)

[3) Config 3](#_Toc107476399)

[Models 3](#_Toc107476400)

[User Input 4](#_Toc107476401)

[4) Data Connectors 5](#_Toc107476402)

[5) Basic Structure of Files 6](#_Toc107476403)

[6) Code Module 6](#_Toc107476404)

[TextClfConfig 6](#_Toc107476405)

[TextClfArch 7](#_Toc107476406)

[TextClfPred 8](#_Toc107476407)

[TextClfPrep 8](#_Toc107476408)

[TextClfSelect\_Model 9](#_Toc107476409)

[TextClfTrn 9](#_Toc107476410)

[7) Conclusion 10](#_Toc107476411)

# Introduction

Text classification is the process of categorizing the text into a group of words. By using NLP, text classification can automatically analyze text and then assign a set of predefined tags or categories based on its context. The text depending upon their characteristics can be labelled for one class or more than one class. If a text is assigned to only one class, it is called “single-label” and if the text is assigned to more than one class, it is called “multi-label”. This module supports user to choose the hyperparameters and the configuration for selecting appropriate model to train. The trained models are saved in the system. The best performing model on the train data is used to predict the target in test data.

# Prerequisites

The data which is to be trained and predicted in this module should follow some necessary formats. This module at present can only read csv and txt. The label can be of the below types:

* Single-label (BillingIssues/PaymentIssues/AppointmentIssues/EmailIssues, etc)
* Multi-label (['comedy', 'european'], ['drama', 'crime', 'comedy'], etc)

# Config

## *Models*

Models that are used in text classifier :

|  |  |  |  |
| --- | --- | --- | --- |
| Models | Model Size | Languages | Content Type |
| USE\_en | 916 | English | uncased |
| USE\_multi | 245 | Multiple Languages | uncased |
| USE\_multi\_large | 303 | Multiple Languages | uncased |
| st5-base | 188 | English | uncased |
| st5-large | 569 | English | uncased |
| st5-3b | 2050 | English | uncased |
| electra\_small | 48 | English | uncased |
| electra\_large | 1160 | English | uncased |
| LaBSE | 1630 | Multiple Languages | uncased |
| MuRIL | 842 | Indian Languages | uncased |
| MuRIL-Large | 842 | Indian Languages | uncased |
| bert\_en\_cased\_small | 386 | English | cased |
| bert\_en\_cased\_large | 1160 | English | cased |
| bert\_en\_uncased\_small | 390 | English | uncased |
| bert\_en\_uncased\_large | 1160 | English | uncased |
| bert\_multi\_cased | 633 | Multiple Languages | cased |
| albert\_en\_large | 64 | English | uncased |
| albert\_en\_xlarge | 210 | English | uncased |
| albert\_en\_xxlarge | 790 | English | uncased |
| mobilebert\_en\_uncased | 134 | English | uncased |
| mobilebert\_multi\_cased | 297 | Multiple Languages | cased |
| funnel-transformer\_small | 524 | English | uncased |
| funnel-transformer\_large | 1550 | English | uncased |
| longformer-base | 765 | English | uncased |
| longformer-large | 1970 | English | uncased |
| distilbert-base-en\_uncased | 363 | English | uncased |
| distilbert-base-en\_cased | 363 | English | cased |
| distilbert-base-multi\_cased | 911 | Multiple Languages | cased |
| roberta-base | 657 | English | cased |
| roberta-large | 1630 | English | cased |
| bart-base | 558 | English | cased |
| bart-large | 1630 | English | cased |
| xlnet-base-cased | 565 | English | cased |
| gpt2 | 498 | English | cased |
| tfidf\_lin\_svc | 0 | English | uncased |
| bow\_md\_lr | 40 | English | cased |
| bow\_lg\_svc | 560 | English | cased |
| word2vec\_nb | 0 | English | cased |
| glove\_lgbm | 105 | English | cased |
| fasttext\_knn | 958 | English | cased |
| sent\_trf\_rf | 91 | English | cased |
| sent\_trf\_sgd | 438 | English | cased |
| use\_stack | 916 | English | uncased |
| use\_multi\_gb | 245 | Multiple Languages | uncased |

## 

## *User Input*

Users can select models by giving specific configuration. The parameters that can be selected are training mode, languages, model size and content type.

|  |  |  |  |
| --- | --- | --- | --- |
| **Selection Parameters** | **User Inputs** | **Selected Configuration** | **Comments** |
| Training mode | settings['fast\_training'] | mode = 'emb\_fit' | Embedding based shallow learning on pre-trained model will be trained. |
| settings['fast\_training',  'small\_data'] |
| settings['small\_data'] | mode = 'top\_tune' | Only the layers added on top of the pre-trained model will be trained. |
| Default | mode = 'fine\_tune' | Layers added on top of the pre-trained model will be trained, followed by a full fine-tuning of all the layers including the pre-trained model. |
| Languages | settings['multi\_lang'] | lang = 'multi' | Models pre-trained in multiple languages will be trained. |
| settings['indian\_lang'] | lang = 'indian' | Models pre-trained in Indian languages will be trained. |
| Default | lang = 'en', 'multi', 'indian' | Models that support all languages will be trained |
| Model Size | settings['light'] | size\_less\_than = 500,  size\_more\_than = 100 | Models of sizes less than 500 MB and more than 100 MB will be trained. |
|  |
| settings['very\_light'] | size\_less\_than = 100 | Models of sizes less than 100 MB will be trained. |  |
| settings['light','very\_light'] | size\_less\_than = 500 | Models of sizes less than 500 MB will be trained. |  |
| Default | All sizes | Models of all sizes will be trained. |  |
| Content Type | settings['formal'] | type = 'cased' | Models that are case-sensitive will be trained. |  |
| Default | type = 'cased',  type = 'uncased' | Case-sensitive and case-ignorant models will be trained. |  |

# Data Connectors

* **data\_path:** location of the training data (should point to the file in case of a single file, should point to the directory in case data exists in multiple files in a directory structure)
* **data\_type:** type of the training data ('csv' / 'json' / 'txt'); currently 'csv' and 'txt' are supported
* **dat\_org [optional]:** organization of the data ('single\_file' / 'multi\_dir')
* **feature\_cols [optional]:** attribute names in the data to be used as features during training; currently only a single feature column is supported; it is optional in cases such as when the data is present in text files under class-specific directories
* **target\_cols [optional]:** attribute names in the data to be used as targets during training (multiple target columns supported only in cases where target data is present across multiple columns in one-hot encoded format, which is possible if it is a multi-label classification task or single label classification task with target data present across multiple columns in one-hot encoded format), it is optional in cases such as when the data is present in text files under class-specific directories
* **val\_data\_path [optional]:** location of validation data (if separate validation data is provided by the user)
* **dir\_to\_save\_model [optional]:** location where trained model should be saved

# Basic Structure of Files

The structural format of the files and folders present in the text classifier module are as below:

# Code Module

## *TextClfConfig*

This class belongs to ‘\text\_clf\cfg\config.py’. The details of the functions under this class are as follows,

* ***set\_user\_cfg(self, user\_cfg)***

This function sets the mode parameter specified by the user.

* ***get\_user\_cfg(self)***

This function returns an empty dictionary for user to specify the mode parameter for training.

* ***get\_models\_by\_names(self, names:[str]=[])***

This function returns the model names and its parameters in list of dictionaries.

* ***get\_models\_by\_attrs(self, \*\*kwargs)***

This function returns the model names according to the model parameters specified by the user.

* ***get\_model\_settings(self, setting\_name:str=None)***

This function returns settings about various model configuration parameters.

* ***get\_default\_arch\_param\_settings(self, setting\_name:str=None)***

This function returns default settings about various architectural parameters

* ***get\_default\_arch\_hyper\_param\_settings(self, setting\_name:str=None)***

This function returns default settings about various architectural hyper-parameters

## *TextClfArch*

This class belongs to ‘\text\_clf\src\arch.py’. The details of the functions under this class are as follows,

* ***create\_model(self,pretrained\_model:dict,is\_trn:bool,***

***fine\_tune:bool,source:str,multi\_label:bool=False, target\_one\_hot\_encoded:bool=False,params:dict={}, hyper\_params:dict={})***

This function creates fine-tuning model architecture for pretrained model and returns the model architecture.

* ***save\_arch\_details(self,model:dict,dir\_to\_save\_model:str, save\_weights\_only:bool=False)***

This function saves architecture data details to be used during prediction and returns the path.

* ***load\_arch\_details(self,model\_arch:dict,model\_weights\_path:str)***

This function is used to Load architecture data details saved during training and returns model.

* ***tokenize\_input(self,model\_arch:dict,sequences:[str], targets:[int]=None,tokenizer:object=None)***

This function tokenizes the input. tokenizer is in charge of preparing the inputs for a model and relevant only for models loaded from hfhub.

* ***\_sanitize\_arch\_inputs(configs:object, is\_trn:bool, fine\_tune:bool, params:dict,hyper\_params:dict,multi\_label:bool, target\_one\_hot\_encoded:bool, num\_classes:int)***

This function verifies the architectural inputs and it rectifies the inputs if not compatible.

* ***\_sanitize\_params(params:dict,multi\_label:bool,***

***target\_one\_hot\_encoded:bool)***

This function verifies the parameters and it rectifies the parameters if not compatible.

* ***\_check\_layer\_config(supported\_params: [str], received\_params: [str])***

This function checks whether the received Layer parameters from user are supported and returns appropriate values.

* ***\_build\_layers(outputs:object,hidden\_layers:(dict),***

***output\_layer:dict)***

This function defines the hidden layers and output layers.

* ***\_translate\_metrics(metrics:[str]) -> [object]***

This function defines the metrics for accuracy calculation

## *TextClfPred*

This class belongs to ‘\text\_clf\src\pred.py’. The details of the functions under this class are as follows,

* ***config\_connectors(self, \*args, \*\*kwargs)***

This function initializes connectors.

* ***pred(self,sequence:str,multi\_label:bool=False,proba\_thr:float=None, top\_n:int=None)***

This function predicts the value of prediction data using the best model stored in the directory and returns the prediction score.

* ***\_load\_model\_details(saved\_model\_dir:str)***

This function loads the model details saved during the training.

## *TextClfPrep*

This class belongs to ‘\text\_clf\src\prep.py’. The details of the functions under this class are as follows,

* ***prep\_data\_trn(self,multi\_label:bool=False,val\_size:float=None, class\_balance:bool=False)***

This function checks whether train data is from a single directory or multi-directory and reads the data.

* ***\_process\_single\_file\_data\_trn(self, multi\_label:bool=False)***

This function Processes the training data if it is in a single file.

* ***def \_numericalize\_single\_label(self)***

This function numericalize the target column which does not involve multi-label.

* ***\_numericalize\_multi\_label(self)***

This function numericalize the target column which involves multi-label.

* ***\_handle\_multi\_label\_single\_target(self)***

This function handles if the training task involves multi-label for a single target column.

* ***\_process\_multi\_dir\_data\_trn(self, multi\_label:bool=False)***

This function processes the training data if it is contained in a directory structure across multiple files.

* ***\_split\_single\_file\_data\_for\_val(self, multi\_label:bool=False)***

This function splits the data into training and validation.

* ***gen\_multi\_dir\_data(self, batch\_size:int)***

This function generates cases where data is present in the directory structure.

* ***save\_prep\_data\_details(self)***

This function saves pre-processing data details to be used during prediction and returns the path.

* ***load\_prep\_data\_details(self)***

This function loads pre-processing data details saved during training.

## *TextClfSelect\_Model*

This class belongs to ‘\text\_clf\src\select\_model.py’. The details of the functions under this class are as follows,

* ***\_select\_candidate\_models(configs:object,settings:dict, list\_settings:[str]=[])***

This function loads the candidate pre-trained models to be fine-tuned and evaluated for this training task before selecting the best model.

* ***\_set\_settings(settings:dict, list\_settings:[str])***

This function defines user-specified settings for types of pretrained models to select as candidate models.

* ***\_set\_item\_in\_settings(settings:dict, item:str, list\_settings:[str])***

This function sets boolean value for user-specified settings.

* ***\_get\_candidate\_models(configs:object, settings:dict)***

This function returns the model with user-specified settings. If the user didn’t specify any settings then it fetches all models.

* ***\_translate\_size\_settings(configs:object, setting\_name:str)***

This function sets the size range of the model.

* ***Load\_ candidate\_models(candidate\_models:[dict])***

This function loads and returns the pretrained models.

## *TextClfTrn*

This class belongs to ‘\text\_clf\src\trn.py’. The details of the functions under this class are as follows,

* ***config\_connectors(self,data\_path:str,data\_type:str, data\_org:str='multi\_dir',feature\_cols:[str]=[],target\_cols:[str]=[], val\_data\_path:str=None,dir\_to\_save\_model:str=None,\*args, \*\*kwargs)***

This function sets all the data connector parameters passed from the user.

* ***trn(self,epochs:int=20,val\_size:float=None, class\_balance:bool=False, \*args, \*\*kwargs)***

This function trains the model with training data according to user specified hyper-parameters. If the user didn’t specify, then it takes the default hyper-parameters.

* ***\_get\_params(kwargs)***

This function sets all user-specified parameters.

* ***\_permute\_hyper\_params(kwargs)***

This function sets all user-specified hyper-parameters.

* ***\_tune\_hyper\_params(pretrained\_model:dict,params:dict, hyper\_params\_permute:list,arch:object,prep:object,multi\_label:bool, epochs:int, verbose:bool)***

This function returns the best model and its best hyper-parameters

* ***\_fit\_model(model:dict, pretrained\_model:dict, arch:object, prep:object, multi\_label:bool, epochs:int, verbose:bool, params:dict, hyper\_params:dict, fine\_tune:bool)***

This function fits pretrained model with training data.

* ***\_save\_model\_details(saved\_model\_dir:str, best\_model:object, prep:object, arch:object)***

This function saves the model details in the specified path.

* ***\_cleanup()***

This function deletes unwanted objects (built-in types or class instances) automatically to free the memory space.

# Conclusion

Thus, text classifier is used to predict predefined categories to a text using best performing model in training dataset. This classifier pipeline enables user to spend the least effort and get the most effective output.