Cell2Doc Outputs on Sample Examples

August 18, 2023

Figures 1 , 2, 3 and 4 show output documentation on sample code examples for PLBART [1], CodeBERT [2], UnixCoder [3], and GraphCodeBERT [4] respectively. We can see from these examples that with the Cell2Doc pipeline, all these pre-trained models have provided longer and more informative documentation compared to their base stand-alone versions (shown in the previous columns in the table). These observations highlight that our Cell2Doc pipeline is model agnostic and can improve the effectiveness of other pre-trained models.

Code cell	**PLBART**	**Cell2Doc (PLBART)**

`n_classes = 12`
`data, labels = make_classification(n_samples =2000, n_features=100, n_informative=50, n_classes=n_classes, random_state= random_state)`
`X, X_test, y, y_test = train_test_split(data , labels, test_size=0.2, random_state= random_state)`
`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)`
`print('Data shape:')`
`print('X_train: %s, X_valid: %s, X_test: %s \n' %(X_train.shape, X_valid.shape, X_test.shape))`
	Splitting the data into training and validation data	STEP 1: Number of classes
		STEP 2: Create classification object
		STEP 3: taking holdout set for validating with random state
		STEP 4: Spliting the dataset into training and validation
		STEP 5: Check data shape

`def cel(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1)`
`    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())`
`def accuracy(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1). squeeze()`
`    y_pred = torch.argmax(y_pred, axis=-1). squeeze()`
`    return (y_true == y_pred).float().sum()/ len(y_true)`
	Evaluation of the Model	STEP 1: The function for evaluation is borrowed from
		STEP 2: Cross entropy loss

`train_images = []`
`image_dirs = np.take(os.listdir('../input/ train'), select_rows)`
`for image_dir in tqdm(sorted(image_dirs)):`
`    image = imread('../input/train/'+ image_dir)`
`    train_images.append(image)`
`    del image`
`    gc.collect()`
`train_images = np.array(train_images)`
	Read in the train images	STEP 1: Read in the training images
		STEP 2: Read in all the images
		STEP 2.1: Read in image and save in list
		STEP 2.2: Delete to reduce memory usage

Figure 1: Code and generated documentation examples using PLBART in CoDoc of Cell2Doc

Code cell	CodeBERT	Cell2Doc (CodeBERT)

`n_classes = 12`
`data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)`
`X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)`
`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)`
`print('Data shape:')`
`print('X_train: %s, X_valid: %s, X_test: %s \n' %(X_train.shape, X_valid.shape, X_test.shape))`
	Train a Classifier	STEP 1: Number of classes per image
		STEP 2: Create a Classification object
		STEP 3: Train Validation Split
		STEP 4: Split the dataset into training and validation
		STEP 5: Checking the shape of the data

`def cel(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1)`
`    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())`
`def accuracy(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1).squeeze()`
`    y_pred = torch.argmax(y_pred, axis=-1).squeeze()`
`    return (y_true == y_pred).float().sum()/len(y_true)`
	Cross entropy loss	STEP 1: Cross entropy loss
		STEP 2: The accuracy of the classifier

`train_images = []`
`image_dirs = np.take(os.listdir('../input/train'), select_rows)`
`for image_dir in tqdm(sorted(image_dirs)):`
`    image = imread('../input/train/'+image_dir)`
`    train_images.append(image)`
`    del image`
`    gc.collect()`
`train_images = np.array(train_images)`
	load train images	STEP 1: Get train images
		STEP 2: Read in each image
		STEP 2.1: Read in the image
		STEP 2.2: Delete image to reduce memory usage

Figure 2: Code and generated documentation examples using CodeBERT in CoDoc of Cell2Doc

Code cell	UnixCoder	Cell2Doc (UnixCoder)

`n_classes = 12`
`data, labels = make_classification(n_samples=2000, n_features=100, n_informative=50, n_classes=n_classes, random_state=random_state)`
`X, X_test, y, y_test = train_test_split(data, labels, test_size=0.2, random_state=random_state)`
`X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.25, random_state=random_state)`
`print('Data shape:')`
`print('X_train: %s, X_valid: %s, X_test: %s \n' %(X_train.shape, X_valid.shape, X_test.shape))`
	Prepare data for modeling	STEP 1: Number of classes used for visualization
		STEP 2: Create data and labels for model
		STEP 3: Split the data into train and test data
		STEP 4: Split the dataset into training and validation sets
		STEP 5: Check data types

`def cel(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1)`
`    return nn.CrossEntropyLoss()(y_pred, y_true.squeeze())`
`def accuracy(y_true, y_pred):`
`    y_true = torch.argmax(y_true, axis=-1).squeeze()`
`    y_pred = torch.argmax(y_pred, axis=-1).squeeze()`
`    return (y_true == y_pred).float().sum()/len(y_true)`
	Cross entropy loss	STEP 1: Cross entropy loss
		STEP 2: Defining the accuracy

`train_images = []`
`image_dirs = np.take(os.listdir('../input/train'), select_rows)`
`for image_dir in tqdm(sorted(image_dirs)):`
`    image = imread('../input/train/'+image_dir)`
`    train_images.append(image)`
`    del image`
`    gc.collect()`
`train_images = np.array(train_images)`
	Read and shuffle filenames	STEP 1: Read in the training dataset
		STEP 2: Read in images
		STEP 2.1: Read in the training images
		STEP 2.2: Free up some memory

Figure 3: Code and generated documentation examples UnixCoder in CoDoc of Cell2Doc

<table>
<tr><th>Code cell</th><th>GraphCodeBERT</th><th>Cell2Doc (GraphCodeBERT)</th></tr>
<tr><td>

```
n_classes = 12
data, labels = make_classification(n_samples
     =2000, n_features=100, n_informative=50,
      n_classes=n_classes, random_state=
     random_state)
X, X_test, y, y_test = train_test_split(data
     , labels, test_size=0.2, random_state=
     random_state)
X_train, X_valid, y_train, y_valid =
     train_test_split(X, y, test_size=0.25,
     random_state=random_state)
print('Data shape:')
print('X_train: %s, X_valid: %s, X_test: %s
     \n' %(X_train.shape, X_valid.shape,
     X_test.shape))
```

</td><td>Splitting the dataset into train and validation set</td><td>STEP 1: Define some constants
STEP 2: Use data augmentation
STEP 3: Splitting the into train and test data
STEP 4: Split datas in train and validation set
STEP 5: Overview of the data</td></tr>
<tr><td>

```
def cel(y_true, y_pred):
    y_true = torch.argmax(y_true, axis=-1)
    return nn.CrossEntropyLoss()(y_pred,
        y_true.squeeze())
def accuracy(y_true, y_pred):
    y_true = torch.argmax(y_true, axis=-1).
        squeeze()
    y_pred = torch.argmax(y_pred, axis=-1).
        squeeze()
    return (y_true == y_pred).float().sum()/
        len(y_true)
```

</td><td>NumtaDB Classification Report</td><td>STEP 1: Define loss function
STEP 2: define eval metrics</td></tr>
<tr><td>

```
train_images = []
image_dirs = np.take(os.listdir('../input/
     train'), select_rows)
for image_dir in tqdm(sorted(image_dirs)):
    image = imread('../input/train/'+
        image_dir)
    train_images.append(image)
    del image
    gc.collect()
train_images = np.array(train_images)
```

</td><td>Load the data</td><td>STEP 1: List of images
STEP 2: Iterate over all images
STEP 2.1: Read in the training data
STEP 2.2: Delete unnecessary images</td></tr>
</table>

Figure 4: Code and generated documentation examples using GraphCodeBERT in CoDoc of Cell2Doc

References

[1] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 2655–2668. [Online]. Available: https://aclanthology.org/2021.naacl-main.211

[2] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.139

[3] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," 2022. [Online]. Available: https://arxiv.org/abs/2203.03850

[4] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu *et al.*, "Graphcodebert: Pre-training code representations with data flow," *arXiv preprint arXiv:2009.08366*, 2020.