Code Structure

Main File (__main__)

question_classifier.py

Dependencies

1. `sys` used for parsing arguments from the CLI
2. `getopt` used for parsing arguments from the CLI
3. `configparser` used for reading config files

Class

no classes

Functions

1. `main` used for reading inputs from CLI. Also reads the config file and converts it into a dictionary. Depending on the config file, it chooses the model (BoW/BiLSTM) to be trained/tested.

Pre Processing Pipeline

preprocessing.py

Dependencies

1. `torch` saving/loading pre-processed files/tensors
2. `numpy` used for correctly loading pre-trained glove embeddings file
3. `string.puncatuations` imported all punctuations that need to be removed

4. `re` to remove punctuations using regular expressions

Class

1. `PreProcessData` stores the preprocessed data
    1. `__init__(self, file_path: str, pre_train_file_path: str, unk_token: str, is_train: bool)` runs the preprocessing pipeline on the given file and stores the result in the variables below
    2. object variables: `labels`, `sentences`, `vocabulary`, `vocabulary_embed`, `sentence_representation`, `label_index`, `label_representation`

Functions

1. `split_tokenize(X: str)` split sentences into plus convert to lower case
2. `remove_stop(word_list: list)` removes all stop words from `word_list`
3. `remove_punc(word_list: list)` remove all punctuations
4. `process_sentence(sentence: str)` tokenises the sentence, removes punctuations and removes stop words
5. `create_vocab(data: list)` given an array questions, generate a vocabulary
6. `create_embeddings(vocab: list, pre_train_file_path: str, unk_token: str)` get the words from pre-trained embeddings based on the generated vocabulary
7. `create_sentence_representation(sentences: list, vocab: list, vocab_embeddings: list)` generate sentence representation using the given vocabulary
8. `create_labels_index(labels: list)` get the label to index conversion
9. `create_labels_representation(labels: list, label_indices: dict)` get the label representation based on label indices
10. `preprocess_pipeline(file_path: str, pre_train_file_path: str, unk_token: str, is_train: bool)` perform all preprocessing on the given dataset, save the preprocessed data in the data directory for future use.
11. `reload_preprocessed()` reload the preprocessed data from the data directory

Bag of Words

bow.py

Dependencies

1. `torch`
2. `numpy`
3. `collections.defaultdict`
4. `time`
5. `matplotlib.pyplot.plt`
6. `itertools`

Class

1. `BoWTextClassifierModule(nn.Module)` This class implements our Bag of Words neural net model. It creates a neural net with 2 layers : the embedding layer and the top linear layer (number of classes).
    1. `__init__(self, text_field_vocab, class_field_vocab, emb_dim, pre_trained_emb_file_path: str, unk_token: str, dropout=0.5, pretrained=False , fr=True)` Initialise the parameters for our model , along with loading the Pre_trained weights if necessary. Provides a freeze option for the pre-trained weights . Creates the layers for the neural net.
    2. `forward(self, docs)` Used for the forward pass of the neural network. Inputs are passed from one layer to another. The embedding of words are created and the mean is calculated for each sentence.
2. `BagOfWords(Classifier)` This is our wrapper class that uses the above classes. We store the config file and implement train and test function that uses the `BoWTextClassifierModule`
    1. `evaluate_validation(scores, loss_function, correct)` Helps evaluate the validation data set. Calculates the accuracy and loss with the appropriate loss function. Used to load the Pre-trained weights from a '.glove' file. These weights are loaded into the model which helps improve its performance.
    2. `train(self)` Function that implements the training loop for our classifier. The data is read using the read_data function. The indexed words are

padded with 0s to make sure all inputs have the same dimensions. This data is split into a training set and validation set in the ratio of 0.9:0.1 respectively. The training loop consists of loading the data in batches and performing the forward pass. The gradient and losses are calculated with a backward pass optimising the parameters using the Adam optimiser. The model and dictionaries are then stored in ‘data folder’

3. `test(self)` This function calculates the accuracy and generates a confusion matrix after evaluating the model with the Testing data set. The model and the training vocabulary are loaded from ‘data folder’. The test data is pre processed and converted into indexed word format.

Functions

1. `plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues)` Function is used to plot the confusion matrix after Testing. Takes in a confusion matrix and plots a graph using ‘matplotlib’ library from python.
2. `loadGloveModel(File)` Used to load the Pre-trained weights from a ‘.glove’ file. These weights are loaded into the model which helps improve its performance.
3. `read_data(filename)` Function used to read data (test and train data). The contents of the file are split into labels and data. The sentences in data are pre-processed using Preprocess.process_sentance(sentance). Each word in a sentence is converted into indexes and stored as a list. Creates Vocabulary of Words and Labels used in the whole file. It returns labels , indexed words , word_vocab , label_vocab

BiLSTM

bilstm.py

Dependencies

1. `torch`
2. `numpy` used to create numpy.array for calculating accuracy

Class

1. `BiLSTM(Classifier)` set up the BiLSTM classifier based on the configuration file, subclass of `Classifier`
    1. `__init__(self, config)` initialise the model with the config file
    2. `collate_fxn(self, input_dataset)` collate function for BiLSTM used by `ConcatDataset` (data loader)
    3. `accuracy_fxn(self, model, loader)` calculate the accuracy of a given BiLSTM model
    4. `output_results(self, accuracy, confusion_matrix, micro_f1, macro_f1, fp)` save the accuracy score, confusion matrix, micro and macro f1 scores in the file pointed by fp
    5. `train(self, processedData=None, trainer_obj=None, save_file_name=None)` this functions preprocesses the data, creates a `Train` object for the purpose of training. Then it selects the appropriate BiLSTM classifier based on the configuration file. It also sets the loss function (default: Cross Entropy Loss) and optimiser (SGD) to be used. Finally it trains the model and saves the result (accuracy, f1 scores, confusion matrix) in a file.
    6. `test(self)` this function preprocesses the testing data, loads a model, creates a `Test` object and performs model testing. Finally it outputs the result (accuracy, f1 scores, confusion matrix) in a file.

Functions

no functions

train.py

Dependencies

1. `torch`
2. `concat_dataset.ConcatDataset` used for loading/reading datasets in batches

Class

1. `Train` sets up the environment and data required in the right format for the purpose of training, allows for training `BilstmModel` type models

1. `__init__(self, config, preProcessedData, collate_fn)` splits the data into train/validation datasets and sets up the necessary data loaders for batch training
2. `doTraining(self, model, model_name, loss_fxn, optimizer, accuracy_fxn, fp, save_file_name=None)` trains the given model of type `BilstmModel`

Functions

no functions

test.py

Dependencies

1. `torch`
2. `numpy` used to create numpy.array for calculating accuracy

Class

1. `Test` sets up the testing environment, loads the data in the correct format and allows for testing `BilstmModel` type models
    1. `__init__(self, preProcessedData, model, model_type: str)` configures the parameters required to test the model
    2. `accuracy_fxn(self, x)` calculate the accuracy of a given BiLSTM model
    3. `doTesting(self)` performs testing of the given `BilstmModel` type model

Functions

no functions

feedforward.py

Dependencies

1. `torch`

Class

1. `Feedforward(torch.nn.Module)` Feedforward class, subclass of `torch.nn.Module`
    1. `__init__(self, input_size, hidden_zie, output_size)` initialise the structure of feed forward neural network
    2. `forward(self, x)` sequence of operations/calculations to be performed at each forward iteration

Functions

no functions

bilstm_model.py

Dependencies

1. `torch`
2. `bilstm.feedforward.Feedforward` to make LSTM use the feedforward neural network

Class

1. `BilstmModel(nn.Module)` define a general BiLSTM model which acts as a skeleton for other model, subclass of `torch.nn.Module`
    1. `__init__(self, input_size, hidden_zie, forward_hidden_zie, forward_output_size, enable_grad=True)` initialise the base BiLSTM model, which sets up the shared functionality for random and pre-trained variants of the model
    2. `forward(self, x, l)` sequence of operations/calculations to be performed at each forward iteration

Functions

no functions

bilstm_random.py

Dependencies

1. `torch`
2. `bisltm_model.BilstmModel` random is a type of BiLSTM model

Class

1. `BilstmRandom(BilstmModel)` BiLSTM model using randomly initialised embeddings, subclass of `BilstmModel`
    1. `__init__(self, input_size, hidden_zie, vocabulary_size, forward_hidden_zie, forward_output_size, enable_grad=True)` initialise the structure of the neural network with the BiLSTM layer and embeddings generated randomly.
2. `BilstmRandomEnsemble(nn.Module)` An ensemble of `BilstmRandom` classifiers, subclass of `torch.nn.Module`
    1. `__init__(self, n_models, input_size, hidden_zie, vocabulary_size, forward_hidden_zie, forward_output_size, enable_grad=True)` initialise the n `BilstmRandom` models and their neural network structures.
    2. `forward(self, x, l)` sequence of operations/calculations to be performed at each forward iteration

Functions

no functions

bilstm_pretrain.py

Dependencies

1. `torch`
2. `bisltm_model.BilstmModel` pretrain is a type of BiLSTM model

Class

1. `BilstmRandom(BilstmModel)` BiLSTM model using pre-trained embeddings, subclass of `BilstmModel`
    1. `__init__(self, embed, hidden_zie, forward_hidden_zie, forward_output_size, enable_grad=True)` initialise the structure of the neural network with the BiLSTM layer and pre-trained embeddings.
2. `BilstmPretrainEnsemble(nn.Module)` An ensemble of `BilstmPretrain` classifiers, subclass of `torch.nn.Module`
    1. `__init__(self, embed, hidden_zie, forward_hidden_zie, forward_output_size, enable_grad=True)` initialise the n `BilstmPretrain` models and their neural

network structures.

2. `forward(self, x, l)` sequence of operations/calculations to be performed at each forward iteration

Functions

no functions

Others

classifier.py

Dependencies

none

Class

1. `Classifier` used as parent class for BoW and BiLSTM classifiers
    1. `__init__(self, config)` sets up and loads the config file for the classifier

Functions

no functions

concat_dataset.py

Dependencies

1. `torch.utils.data.Dataset` used as parent class (of `ConcatDataset`) for loading data in batch`

Class

1. `ConcatDataset(Dataset)` used for loading data in batches, subclass of `Dataset`
    1. `__init__(self, xy)` store labels and data

2. `__getitem__(self, index)` get the data and label of item corresponding to index
3. `__len__(self)` get the total number of items

Functions

no functions

eval.py

Dependencies

1. `numpy` used for creating numpy.arrays

Class

no classes

Functions

1. `get_confusion_matrix(y_actual, y_pred, size)` generate a confusion matrix using actual and predicted labels
2. `get_micro_f1(confusion_matrix)` calculate micro f1 score using confusion matrix
3. `get_macro_f1(confusion_matrix)` calculate macro f1 using confusion matrix