5_3_Document_Classification_with_CNN

July 30, 2019

1 Using Embeddings for Document Classification

1.1 Imports

```
In [1]: import os
        from argparse import Namespace
        from collections import Counter
        import json
        import re
        import string

        import numpy as np
        import pandas as pd
        import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        from torch.utils.data import Dataset, DataLoader
        from tqdm import tqdm_notebook
```

1.2 Data Vectorization classes

1.2.1 The Vocabulary

```
In [2]: class Vocabulary(object):
            """Class to process text and extract vocabulary for mapping"""

            def __init__(self, token_to_idx=None):
                """
                Args:
                    token_to_idx (dict): a pre-existing map of tokens to indices
                """

                if token_to_idx is None:
                    token_to_idx = {}
                self._token_to_idx = token_to_idx

                self._idx_to_token = {idx: token
```

```
                              for token, idx in self._token_to_idx.items()}

    def to_serializable(self):
        """ returns a dictionary that can be serialized """
        return {'token_to_idx': self._token_to_idx}

    @classmethod
    def from_serializable(cls, contents):
        """ instantiates the Vocabulary from a serialized dictionary """
        return cls(**contents)

    def add_token(self, token):
        """Update mapping dicts based on the token.

        Args:
            token (str): the item to add into the Vocabulary
        Returns:
            index (int): the integer corresponding to the token
        """
        if token in self._token_to_idx:
            index = self._token_to_idx[token]
        else:
            index = len(self._token_to_idx)
            self._token_to_idx[token] = index
            self._idx_to_token[index] = token
        return index

    def add_many(self, tokens):
        """Add a list of tokens into the Vocabulary

        Args:
            tokens (list): a list of string tokens
        Returns:
            indices (list): a list of indices corresponding to the tokens
        """
        return [self.add_token(token) for token in tokens]

    def lookup_token(self, token):
        """Retrieve the index associated with the token

        Args:
            token (str): the token to look up
        Returns:
            index (int): the index corresponding to the token
        """
        return self._token_to_idx[token]

    def lookup_index(self, index):
```

```
                """Return the token associated with the index

                Args:
                    index (int): the index to look up
                Returns:
                    token (str): the token corresponding to the index
                Raises:
                    KeyError: if the index is not in the Vocabulary
                """
                if index not in self._idx_to_token:
                    raise KeyError("the index (%d) is not in the Vocabulary" % index)
                return self._idx_to_token[index]

            def __str__(self):
                return "<Vocabulary(size=%d)>" % len(self)

            def __len__(self):
                return len(self._token_to_idx)

In [3]: class SequenceVocabulary(Vocabulary):
            def __init__(self, token_to_idx=None, unk_token="<UNK>",
                         mask_token="<MASK>", begin_seq_token="<BEGIN>",
                         end_seq_token="<END>"):

                super(SequenceVocabulary, self).__init__(token_to_idx)

                self._mask_token = mask_token
                self._unk_token = unk_token
                self._begin_seq_token = begin_seq_token
                self._end_seq_token = end_seq_token

                self.mask_index = self.add_token(self._mask_token)
                self.unk_index = self.add_token(self._unk_token)
                self.begin_seq_index = self.add_token(self._begin_seq_token)
                self.end_seq_index = self.add_token(self._end_seq_token)

            def to_serializable(self):
                contents = super(SequenceVocabulary, self).to_serializable()
                contents.update({'unk_token': self._unk_token,
                                 'mask_token': self._mask_token,
                                 'begin_seq_token': self._begin_seq_token,
                                 'end_seq_token': self._end_seq_token})
                return contents

            def lookup_token(self, token):
                """Retrieve the index associated with the token
                  or the UNK index if token isn't present.
                Args:
                    token (str): the token to look up
                Returns:
                    index (int): the index corresponding to the token
                Notes:
                    `unk_index` needs to be >=0 (having been added into the Vocabulary)
                      for the UNK functionality
                """
                if self.unk_index >= 0:
                    return self._token_to_idx.get(token, self.unk_index)
                else:
                    return self._token_to_idx[token]
```

1.2.2 The Vectorizer

```
In [4]: class NewsVectorizer(object):
            """ The Vectorizer which coordinates the Vocabularies and puts them to use"""
            def __init__(self, title_vocab, category_vocab):
                self.title_vocab = title_vocab
                self.category_vocab = category_vocab

            def vectorize(self, title, vector_length=-1):
                """
                Args:
                    title (str): the string of words separated by a space
                    vector_length (int): an argument for forcing the length of index vector
                Returns:
                    the vetorized title (numpy.array)
                """
                indices = [self.title_vocab.begin_seq_index]
                indices.extend(self.title_vocab.lookup_token(token)
                               for token in title.split(" "))
                indices.append(self.title_vocab.end_seq_index)

                if vector_length < 0:
                    vector_length = len(indices)

                out_vector = np.zeros(vector_length, dtype=np.int64)
                out_vector[:len(indices)] = indices
                out_vector[len(indices):] = self.title_vocab.mask_index

                return out_vector

            @classmethod
            def from_dataframe(cls, news_df, cutoff=25):
                """Instantiate the vectorizer from the dataset dataframe

                Args:
                    news_df (pandas.DataFrame): the target dataset
                    cutoff (int): frequency threshold for including in Vocabulary
                Returns:
                    an instance of the NewsVectorizer
                """
                category_vocab = Vocabulary()
                for category in sorted(set(news_df.category)):
                    category_vocab.add_token(category)

                word_counts = Counter()
                for title in news_df.title:
                    for token in title.split(" "):
                        if token not in string.punctuation:
                            word_counts[token] += 1

                title_vocab = SequenceVocabulary()
                for word, word_count in word_counts.items():
                    if word_count >= cutoff:
                        title_vocab.add_token(word)

                return cls(title_vocab, category_vocab)

            @classmethod
            def from_serializable(cls, contents):
                title_vocab = \
                    SequenceVocabulary.from_serializable(contents['title_vocab'])
                category_vocab =  \
                    Vocabulary.from_serializable(contents['category_vocab'])

                return cls(title_vocab=title_vocab, category_vocab=category_vocab)

            def to_serializable(self):
                return {'title_vocab': self.title_vocab.to_serializable(),
                        'category_vocab': self.category_vocab.to_serializable()}
```

1.2.3 The Dataset

```
In [5]: class NewsDataset(Dataset):
            def __init__(self, news_df, vectorizer):
                """
                Args:
                    news_df (pandas.DataFrame): the dataset
                    vectorizer (NewsVectorizer): vectorizer instatiated from dataset
                """
                self.news_df = news_df
                self._vectorizer = vectorizer

                # +1 if only using begin_seq, +2 if using both begin and end seq tokens
```

```
        measure_len = lambda context: len(context.split(" "))
        self._max_seq_length = max(map(measure_len, news_df.title)) + 2


        self.train_df = self.news_df[self.news_df.split=='train']
        self.train_size = len(self.train_df)

        self.val_df = self.news_df[self.news_df.split=='val']
        self.validation_size = len(self.val_df)

        self.test_df = self.news_df[self.news_df.split=='test']
        self.test_size = len(self.test_df)

        self._lookup_dict = {'train': (self.train_df, self.train_size),
                             'val': (self.val_df, self.validation_size),
                             'test': (self.test_df, self.test_size)}

        self.set_split('train')

        # Class weights
        class_counts = news_df.category.value_counts().to_dict()
        def sort_key(item):
            return self._vectorizer.category_vocab.lookup_token(item[0])
        sorted_counts = sorted(class_counts.items(), key=sort_key)
        frequencies = [count for _, count in sorted_counts]
        self.class_weights = 1.0 / torch.tensor(frequencies, dtype=torch.float32)


    @classmethod
    def load_dataset_and_make_vectorizer(cls, news_csv):
        """Load dataset and make a new vectorizer from scratch

        Args:
            surname_csv (str): location of the dataset
        Returns:
            an instance of SurnameDataset
        """
        news_df = pd.read_csv(news_csv)
        train_news_df = news_df[news_df.split=='train']
        return cls(news_df, NewsVectorizer.from_dataframe(train_news_df))

    @classmethod
    def load_dataset_and_load_vectorizer(cls, news_csv, vectorizer_filepath):
        """Load dataset and the corresponding vectorizer.
        Used in the case in the vectorizer has been cached for re-use

        Args:
            surname_csv (str): location of the dataset
            vectorizer_filepath (str): location of the saved vectorizer
        Returns:
            an instance of SurnameDataset
        """
        news_df = pd.read_csv(news_csv)
        vectorizer = cls.load_vectorizer_only(vectorizer_filepath)
        return cls(news_csv, vectorizer)

    @staticmethod
    def load_vectorizer_only(vectorizer_filepath):
        """a static method for loading the vectorizer from file

        Args:
            vectorizer_filepath (str): the location of the serialized vectorizer
        Returns:
            an instance of SurnameVectorizer
        """
        with open(vectorizer_filepath) as fp:
            return NameVectorizer.from_serializable(json.load(fp))

    def save_vectorizer(self, vectorizer_filepath):
        """saves the vectorizer to disk using json

        Args:
            vectorizer_filepath (str): the location to save the vectorizer
        """
        with open(vectorizer_filepath, "w") as fp:
            json.dump(self._vectorizer.to_serializable(), fp)

    def get_vectorizer(self):
        """ returns the vectorizer """
        return self._vectorizer

    def set_split(self, split="train"):
        """ selects the splits in the dataset using a column in the dataframe """
        self._target_split = split
        self._target_df, self._target_size = self._lookup_dict[split]

    def __len__(self):
        return self._target_size

    def __getitem__(self, index):
        """the primary entry point method for PyTorch datasets

        Args:
            index (int): the index to the data point
        Returns:
            a dictionary holding the data point's features (x_data) and label (y_targe
        """
        row = self._target_df.iloc[index]

        title_vector = \
            self._vectorizer.vectorize(row.title, self._max_seq_length)

        category_index = \
            self._vectorizer.category_vocab.lookup_token(row.category)

        return {'x_data': title_vector,
                'y_target': category_index}

    def get_num_batches(self, batch_size):
        """Given a batch size, return the number of batches in the dataset

        Args:
            batch_size (int)
        Returns:
            number of batches in the dataset
        """
        return len(self) // batch_size

def generate_batches(dataset, batch_size, shuffle=True,
                     drop_last=True, device="cpu"):
    """
    A generator function which wraps the PyTorch DataLoader. It will
      ensure each tensor is on the write device location.
    """
    dataloader = DataLoader(dataset=dataset, batch_size=batch_size,
                            shuffle=shuffle, drop_last=drop_last)

    for data_dict in dataloader:
        out_data_dict = {}
        for name, tensor in data_dict.items():
            out_data_dict[name] = data_dict[name].to(device)
        yield out_data_dict
```

1.3 The Model: NewsClassifier

```
In [6]: class NewsClassifier(nn.Module):
            def __init__(self, embedding_size, num_embeddings, num_channels,
                         hidden_dim, num_classes, dropout_p,
                         pretrained_embeddings=None, padding_idx=0):
                """
                Args:
                    embedding_size (int): size of the embedding vectors
                    num_embeddings (int): number of embedding vectors
                    filter_width (int): width of the convolutional kernels
                    num_channels (int): number of convolutional kernels per layer
                    hidden_dim (int): the size of the hidden dimension
                    num_classes (int): the number of classes in classification
                    dropout_p (float): a dropout parameter
                    pretrained_embeddings (numpy.array): previously trained word embeddings
                        default is None. If provided,
                    padding_idx (int): an index representing a null position
                """
                super(NewsClassifier, self).__init__()

                if pretrained_embeddings is None:

                    self.emb = nn.Embedding(embedding_dim=embedding_size,
                                            num_embeddings=num_embeddings,
                                            padding_idx=padding_idx)
                else:
                    pretrained_embeddings = torch.from_numpy(pretrained_embeddings).float()
                    self.emb = nn.Embedding(embedding_dim=embedding_size,
                                            num_embeddings=num_embeddings,
                                            padding_idx=padding_idx,
                                            _weight=pretrained_embeddings)


                self.convnet = nn.Sequential(
                    nn.Conv1d(in_channels=embedding_size,
                           out_channels=num_channels, kernel_size=3),
                    nn.ELU(),
                    nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
                           kernel_size=3, stride=2),
                    nn.ELU(),
                    nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
                           kernel_size=3, stride=2),
                    nn.ELU(),
                    nn.Conv1d(in_channels=num_channels, out_channels=num_channels,
                           kernel_size=3),
                    nn.ELU()
                )

                self._dropout_p = dropout_p
                self.fc1 = nn.Linear(num_channels, hidden_dim)
                self.fc2 = nn.Linear(hidden_dim, num_classes)

            def forward(self, x_in, apply_softmax=False):
                """The forward pass of the classifier

                Args:
                    x_in (torch.Tensor): an input data tensor.
                        x_in.shape should be (batch, dataset._max_seq_length)
                    apply_softmax (bool): a flag for the softmax activation
                        should be false if used with the Cross Entropy losses
                Returns:
                    the resulting tensor. tensor.shape should be (batch, num_classes)
                """

                # embed and permute so features are channels
                x_embedded = self.emb(x_in).permute(0, 2, 1)

                features = self.convnet(x_embedded)

                # average and remove the extra dimension
                remaining_size = features.size(dim=2)
                features = F.avg_pool1d(features, remaining_size).squeeze(dim=2)
                features = F.dropout(features, p=self._dropout_p)

                # mlp classifier
                intermediate_vector = F.relu(F.dropout(self.fc1(features), p=self._dropout_p))
                prediction_vector = self.fc2(intermediate_vector)

                if apply_softmax:
                    prediction_vector = F.softmax(prediction_vector, dim=1)

                return prediction_vector
```

1.4 Training Routine

1.4.1 Helper functions

```
In [13]: def make_train_state(args):
             return {'stop_early': False,
                     'early_stopping_step': 0,
                     'early_stopping_best_val': 1e8,
                     'learning_rate': args.learning_rate,
                     'epoch_index': 0,
                     'train_loss': [],
                     'train_acc': [],
                     'val_loss': [],
                     'val_acc': [],
                     'test_loss': -1,
                     'test_acc': -1,
                     'model_filename': args.model_state_file}

         def update_train_state(args, model, train_state):
             """Handle the training state updates.

             Components:
              - Early Stopping: Prevent overfitting.
              - Model Checkpoint: Model is saved if the model is better

             :param args: main arguments
             :param model: model to train
             :param train_state: a dictionary representing the training state values
             :returns:
                 a new train_state
             """

             # Save one model at least
             if train_state['epoch_index'] == 0:
                 torch.save(model.state_dict(), train_state['model_filename'])
                 train_state['stop_early'] = False

             # Save model if performance improved
             elif train_state['epoch_index'] >= 1:
                 loss_tm1, loss_t = train_state['val_loss'][-2:]

                 # If loss worsened
                 if loss_t >= train_state['early_stopping_best_val']:
                     # Update step
                     train_state['early_stopping_step'] += 1
                 # Loss decreased
                 else:
                     # Save the best model
                     if loss_t < train_state['early_stopping_best_val']:
                         torch.save(model.state_dict(), train_state['model_filename'])

                     # Reset early stopping step
                     train_state['early_stopping_step'] = 0

                 # Stop early ?
                 train_state['stop_early'] = \
                     train_state['early_stopping_step'] >= args.early_stopping_criteria

             return train_state

         def compute_accuracy(y_pred, y_target):
             _, y_pred_indices = y_pred.max(dim=1)
             n_correct = torch.eq(y_pred_indices, y_target).sum().item()
             return n_correct / len(y_pred_indices) * 100
```

general utilities

```
In [14]: def set_seed_everywhere(seed, cuda):
             np.random.seed(seed)
             torch.manual_seed(seed)
             if cuda:
```

```python
        torch.cuda.manual_seed_all(seed)

def handle_dirs(dirpath):
    if not os.path.exists(dirpath):
        os.makedirs(dirpath)

def load_glove_from_file(glove_filepath):
    """
    Load the GloVe embeddings

    Args:
        glove_filepath (str): path to the glove embeddings file
    Returns:
        word_to_index (dict), embeddings (numpy.ndarary)
    """

    word_to_index = {}
    embeddings = []
    with open(glove_filepath, "r") as fp:
        for index, line in enumerate(fp):
            line = line.split(" ") # each line: word num1 num2 ...
            word_to_index[line[0]] = index # word = line[0]
            embedding_i = np.array([float(val) for val in line[1:]])
            embeddings.append(embedding_i)
    return word_to_index, np.stack(embeddings)

def make_embedding_matrix(glove_filepath, words):
    """
    Create embedding matrix for a specific set of words.

    Args:
        glove_filepath (str): file path to the glove embeddigns
        words (list): list of words in the dataset
    """
    word_to_idx, glove_embeddings = load_glove_from_file(glove_filepath)
    embedding_size = glove_embeddings.shape[1]

    final_embeddings = np.zeros((len(words), embedding_size))

    for i, word in enumerate(words):
        if word in word_to_idx:
            final_embeddings[i, :] = glove_embeddings[word_to_idx[word]]
        else:
            embedding_i = torch.ones(1, embedding_size)
            torch.nn.init.xavier_uniform_(embedding_i)
            final_embeddings[i, :] = embedding_i

    return final_embeddings
```

1.4.2 Settings and some prep work

```
In [15]: from argparse import Namespace
```

```
In [16]: args = Namespace(
             # Data and Path hyper parameters
             news_csv="data/ag_news/news_with_splits.csv",
             vectorizer_file="vectorizer.json",
             model_state_file="model.pth",
             save_dir="model_storage/ch5/document_classification",
             # Model hyper parameters
             glove_filepath='data/glove/glove.6B.100d.txt',
             use_glove=False,
             embedding_size=100,
             hidden_dim=100,
             num_channels=100,
             # Training hyper parameter
             seed=1337,
             learning_rate=0.001,
             dropout_p=0.1,
             batch_size=128,
             num_epochs=100,
             early_stopping_criteria=5,
             # Runtime option
             cuda=True,
             catch_keyboard_interrupt=True,
             reload_from_files=False,
             expand_filepaths_to_save_dir=True
         )

         if args.expand_filepaths_to_save_dir:
             args.vectorizer_file = os.path.join(args.save_dir,
                                                 args.vectorizer_file)

             args.model_state_file = os.path.join(args.save_dir,
                                                  args.model_state_file)

             print("Expanded filepaths: ")
             print("\t{}".format(args.vectorizer_file))
             print("\t{}".format(args.model_state_file))

         # Check CUDA
         if not torch.cuda.is_available():
             args.cuda = False

         args.device = torch.device("cuda" if args.cuda else "cpu")
         print("Using CUDA: {}".format(args.cuda))

         # Set seed for reproducibility
         set_seed_everywhere(args.seed, args.cuda)

         # handle dirs
         handle_dirs(args.save_dir)

Expanded filepaths:
        model_storage/ch5/document_classification/vectorizer.json
        model_storage/ch5/document_classification/model.pth
Using CUDA: False
```

1.4.3 Initializations

```
In [17]: args.use_glove = True

In [18]: if args.reload_from_files:
             # training from a checkpoint
             dataset = NewsDataset.load_dataset_and_load_vectorizer(args.news_csv,
                                                                       args.vectorizer_file)
         else:
             # create dataset and vectorizer
             dataset = NewsDataset.load_dataset_and_make_vectorizer(args.news_csv)
             dataset.save_vectorizer(args.vectorizer_file)
         vectorizer = dataset.get_vectorizer()

         # Use GloVe or randomly initialized embeddings
         if args.use_glove:
             words = vectorizer.title_vocab._token_to_idx.keys()
             embeddings = make_embedding_matrix(glove_filepath=args.glove_filepath,
                                                words=words)
             print("Using pre-trained embeddings")
         else:
             print("Not using pre-trained embeddings")
             embeddings = None

         classifier = NewsClassifier(embedding_size=args.embedding_size,
                                     num_embeddings=len(vectorizer.title_vocab),
                                     num_channels=args.num_channels,
                                     hidden_dim=args.hidden_dim,
                                     num_classes=len(vectorizer.category_vocab),
                                     dropout_p=args.dropout_p,
                                     pretrained_embeddings=embeddings,
                                     padding_idx=0)

Using pre-trained embeddings
```

1.4.4 Training loop

```
In [19]: classifier = classifier.to(args.device)
         dataset.class_weights = dataset.class_weights.to(args.device)

         loss_func = nn.CrossEntropyLoss(dataset.class_weights)
         optimizer = optim.Adam(classifier.parameters(), lr=args.learning_rate)
         scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer,
                                                    mode='min', factor=0.5,
                                                    patience=1)

         train_state = make_train_state(args)

         epoch_bar = tqdm_notebook(desc='training routine',
                                   total=args.num_epochs,
                                   position=0)

         dataset.set_split('train')
         train_bar = tqdm_notebook(desc='split=train',
                                   total=dataset.get_num_batches(args.batch_size),
                                   position=1,
                                   leave=True)
         dataset.set_split('val')
         val_bar = tqdm_notebook(desc='split=val',
                                 total=dataset.get_num_batches(args.batch_size),
                                 position=1,
                                 leave=True)

         try:
             for epoch_index in range(args.num_epochs):
                 train_state['epoch_index'] = epoch_index

                 # Iterate over training dataset

                 # setup: batch generator, set loss and acc to 0, set train mode on

                 dataset.set_split('train')
                 batch_generator = generate_batches(dataset,
                                                    batch_size=args.batch_size,
                                                    device=args.device)
                 running_loss = 0.0
                 running_acc = 0.0
                 classifier.train()

                 for batch_index, batch_dict in enumerate(batch_generator):
                     # the training routine is these 5 steps:

                     # --------------------------------------
```

```
    # step 1. zero the gradients
    optimizer.zero_grad()

    # step 2. compute the output
    y_pred = classifier(batch_dict['x_data'])

    # step 3. compute the loss
    loss = loss_func(y_pred, batch_dict['y_target'])
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # step 4. use loss to produce gradients
    loss.backward()

    # step 5. use optimizer to take gradient step
    optimizer.step()
    # -----------------------------------------
    # compute the accuracy
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)

    # update bar
    train_bar.set_postfix(loss=running_loss, acc=running_acc,
                          epoch=epoch_index)
    train_bar.update()

train_state['train_loss'].append(running_loss)
train_state['train_acc'].append(running_acc)

# Iterate over val dataset

# setup: batch generator, set loss and acc to 0; set eval mode on
dataset.set_split('val')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):

    # compute the output
    y_pred =  classifier(batch_dict['x_data'])

    # step 3. compute the loss
    loss = loss_func(y_pred, batch_dict['y_target'])
    loss_t = loss.item()
```

```
                     running_loss += (loss_t - running_loss) / (batch_index + 1)

                     # compute the accuracy
                     acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
                     running_acc += (acc_t - running_acc) / (batch_index + 1)
                     val_bar.set_postfix(loss=running_loss, acc=running_acc,
                                     epoch=epoch_index)
                     val_bar.update()

                 train_state['val_loss'].append(running_loss)
                 train_state['val_acc'].append(running_acc)

                 train_state = update_train_state(args=args, model=classifier,
                                                  train_state=train_state)

                 scheduler.step(train_state['val_loss'][-1])

                 if train_state['stop_early']:
                     break

                 train_bar.n = 0
                 val_bar.n = 0
                 epoch_bar.update()
         except KeyboardInterrupt:
             print("Exiting loop")
```

```
HBox(children=(IntProgress(value=0, description='training routine', style=ProgressStyle(descri


HBox(children=(IntProgress(value=0, description='split=train', max=656, style=ProgressStyle(de


HBox(children=(IntProgress(value=0, description='split=val', max=140, style=ProgressStyle(desc
```

```
In [20]: # compute the loss & accuracy on the test set using the best available model

         classifier.load_state_dict(torch.load(train_state['model_filename']))

         classifier = classifier.to(args.device)
         dataset.class_weights = dataset.class_weights.to(args.device)
         loss_func = nn.CrossEntropyLoss(dataset.class_weights)

         dataset.set_split('test')
         batch_generator = generate_batches(dataset,
                                            batch_size=args.batch_size,
                                            device=args.device)
         running_loss = 0.
         running_acc = 0.
         classifier.eval()

         for batch_index, batch_dict in enumerate(batch_generator):
             # compute the output
             y_pred =  classifier(batch_dict['x_data'])

             # compute the loss
             loss = loss_func(y_pred, batch_dict['y_target'])
             loss_t = loss.item()
             running_loss += (loss_t - running_loss) / (batch_index + 1)

             # compute the accuracy
             acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
             running_acc += (acc_t - running_acc) / (batch_index + 1)

         train_state['test_loss'] = running_loss
         train_state['test_acc'] = running_acc
```

```
In [21]: print("Test loss: {};".format(train_state['test_loss']))
         print("Test Accuracy: {}".format(train_state['test_acc']))
```

```
Test loss: 0.623714632647378;
Test Accuracy: 79.8214285714286
```

1.4.5 Inference

```
In [22]: # Preprocess the reviews
         def preprocess_text(text):
             text = ' '.join(word.lower() for word in text.split(" "))
             text = re.sub(r"([.,!?])", r" \1 ", text)
             text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)
             return text
```

```
In [23]: def predict_category(title, classifier, vectorizer, max_length):
             """Predict a News category for a new title

             Args:
                 title (str): a raw title string
                 classifier (NewsClassifier): an instance of the trained classifier
                 vectorizer (NewsVectorizer): the corresponding vectorizer
                 max_length (int): the max sequence length
                     Note: CNNs are sensitive to the input data tensor size.
                           This ensures to keep it the same size as the training data
             """
             title = preprocess_text(title)
             vectorized_title = \
                 torch.tensor(vectorizer.vectorize(title, vector_length=max_length))
             result = classifier(vectorized_title.unsqueeze(0), apply_softmax=True)
```

```
            probability_values, indices = result.max(dim=1)
            predicted_category = vectorizer.category_vocab.lookup_index(indices.item())

            return {'category': predicted_category,
                    'probability': probability_values.item()}
```

```
In [24]: def get_samples():
             samples = {}
             for cat in dataset.val_df.category.unique():
                 samples[cat] = dataset.val_df.title[dataset.val_df.category==cat].tolist()[:5]
             return samples

         val_samples = get_samples()
```

1.4.6 Original Titles in Validation Samples

```
In [26]: #title = input("Enter a news title to classify: ")
         classifier = classifier.to("cpu")

         for truth, sample_group in val_samples.items():
             print(f"True Category: {truth}")
             print("="*30)
             for sample in sample_group:
                 prediction = predict_category(sample, classifier,
                                               vectorizer, dataset._max_seq_length + 1)
                 print("Prediction: {} (p={:0.2f})".format(prediction['category'],
                                                           prediction['probability']))
                 print("\t + Sample: {}".format(sample))
             print("-"*30 + "\n")
```

```
True Category: Business
==============================
Prediction: Business (p=0.85)
         + Sample: AZ suspends marketing of cancer drug
Prediction: Business (p=1.00)
         + Sample: Business world has mixed reaction to Perez move
Prediction: Sports (p=0.66)
         + Sample: Betting Against Bombay
Prediction: Sports (p=0.36)
         + Sample: Malpractice Insurers Face a Tough Market
Prediction: Sports (p=0.70)
         + Sample: NVIDIA Is Vindicated
------------------------------

True Category: Sci/Tech
==============================
Prediction: World (p=0.52)
         + Sample: Spies prize webcam #39;s eyes
```

```
Prediction: Sci/Tech (p=0.99)
         + Sample: Sober worm causes headaches
Prediction: World (p=0.79)
         + Sample: Local Search: Missing Pieces Falling into Place
Prediction: Sci/Tech (p=1.00)
         + Sample: Hackers baiting Internet users with Beckham pix
Prediction: Sports (p=0.96)
         + Sample: Nokia adds BlackBerry support to Series 80 handsets
------------------------------

True Category: Sports
==============================
Prediction: Sci/Tech (p=0.74)
         + Sample: Is Meyer the man to get Irish up?
Prediction: Sci/Tech (p=0.34)
         + Sample: Who? Who? And Clemens
Prediction: Sports (p=0.99)
         + Sample: Baseball Today (AP)
Prediction: World (p=0.82)
         + Sample: Mark Kreidler: Yao Ming epitomizes the Chinese athlete who is &lt;b&gt;...&l
Prediction: Sports (p=0.87)
         + Sample: No. 5 Miami Rebounds to Beat FSU in Overtime
------------------------------

True Category: World
==============================
Prediction: Sports (p=0.65)
         + Sample: Arafat in pain but expected to recover-Shaath
Prediction: World (p=1.00)
         + Sample: Maoist rebels bomb Kathmandu building, no injuries (Reuters)
Prediction: World (p=0.99)
         + Sample: Son Running for Ill. Rep.'s House Seat (AP)
Prediction: World (p=0.62)
         + Sample: Strong Quake Hits in Japan
Prediction: World (p=1.00)
         + Sample: Israel assassinates Hamas militant in Damascus
------------------------------
```

1.4.7 News Titles Harvested from Recent New York Times

Fetch 15 news titles from recent New York Times. Choose 3 moderately long titles from each of five categories: Sports, Science/Technology, World, Business and Cooking. Tell us how is your network classifying each one of those titles. Your network is trained on Sports, Science/Technology, World, and Business and not Cooking. Where does it places Cooking titles?

```
In [31]: newsTitles = {
         "Sports":
         ["Jin Young Ko Wins Evian Championship, Her Second Major This Season",
         "At the Fortnite World Cup, Slurp Juice, Zip Lines and a Teenage Millionaire Winner",
         "Senators Will Push for Increased Oversight of Olympic Sports"],
         "Technology":
         ["Would You Want a Computer to Judge Your Risk of H.I.V. Infection?",
         "Apple Reports Declining Profits and Stagnant Growth, Again",
         "Facebook Connected Her to a Tattooed Soldier in Iraq. Or So She Thought."],
         "World":
         ["Doubt Greets Chinas Claim That Muslims Have Been Released From Camps",
         "They Survived Colonization and War. But Venezuelas Collapse Was Too Much.",
         "Costa Rica and Panama Arrest Dozens on Suspicion of People Smuggling"],
         "Business":
         ["Your Next iPhone Might Be Made in Vietnam. Thank the Trade War.",
         "Secret of a New York Farm Stands Success: An Eye for the Next Big Thing",
         "Apple Reports Declining Profits and Stagnant Growth, Again"],
         "Cooking":
         ["After 34 Years, Gotham Bar and Grill Has a New Head Chef",
         "This Pasta May Actually Taste Better Outside",
         "The Dim Sum Palace That Has Reigned Over Elizabeth Street Since 1978"]
         }
```

```
In [33]: #title = input("Enter a news title to classify: ")
         classifier = classifier.to("cpu")

         for truth, sample_group in newsTitles.items():
             print(f"True Category: {truth}")
             print("="*30)
             for sample in sample_group:
                 prediction = predict_category(sample, classifier,
                                                  vectorizer, dataset._max_seq_length + 1)
                 print("Prediction: {} (p={:0.2f})".format(prediction['category'],
                                                           prediction['probability']))
                 print("\t + Sample: {}".format(sample))
             print("-"*30 + "\n")
```

```
True Category: Sports
==============================
Prediction: Sports (p=1.00)
         + Sample: Jin Young Ko Wins Evian Championship, Her Second Major This Season
Prediction: Sci/Tech (p=0.56)
         + Sample: At the Fortnite World Cup, Slurp Juice, Zip Lines and a Teenage Millionaire
Prediction: World (p=0.42)
         + Sample: Senators Will Push for Increased Oversight of Olympic Sports
------------------------------

True Category: Technology
```

```
==============================
Prediction: Sci/Tech (p=0.94)
         + Sample: Would You Want a Computer to Judge Your Risk of H.I.V. Infection?
Prediction: Business (p=0.96)
         + Sample: Apple Reports Declining Profits and Stagnant Growth, Again
Prediction: Sports (p=0.63)
         + Sample: Facebook Connected Her to a Tattooed Soldier in Iraq. Or So She Thought.
------------------------------

True Category: World
==============================
Prediction: Sci/Tech (p=0.39)
         + Sample: Doubt Greets Chinas Claim That Muslims Have Been Released From Camps
Prediction: World (p=0.60)
         + Sample: They Survived Colonization and War. But Venezuelas Collapse Was Too Much.
Prediction: World (p=0.76)
         + Sample: Costa Rica and Panama Arrest Dozens on Suspicion of People Smuggling
------------------------------

True Category: Business
==============================
Prediction: Sci/Tech (p=0.70)
         + Sample: Your Next iPhone Might Be Made in Vietnam. Thank the Trade War.
Prediction: Sports (p=0.89)
         + Sample: Secret of a New York Farm Stands Success: An Eye for the Next Big Thing
Prediction: Business (p=0.88)
         + Sample: Apple Reports Declining Profits and Stagnant Growth, Again
------------------------------

True Category: Cooking
==============================
Prediction: World (p=0.35)
         + Sample: After 34 Years, Gotham Bar and Grill Has a New Head Chef
Prediction: Sports (p=0.97)
         + Sample: This Pasta May Actually Taste Better Outside
Prediction: Business (p=0.51)
         + Sample: The Dim Sum Palace That Has Reigned Over Elizabeth Street Since 1978
------------------------------
```

1.4.8 Summarization of the performance of the predictive neural network model

Sports: It classifies the first typical sports news title correctly (and with 100% confidence), but it fails to recognize the latter two titles and wrongly classifies them into **Sci/Tech** and **World**, though it actually makes some sense to put the title “Senators Will Push for Increased Oversight of Olympic Sports” under **world**.

Technology: It classifies the first typical technology news title correctly (and with 90% confidence). The second one is a bit tricky (intentionally chosen) as it has two true categories: both **Technology** and **Business**. The model is good enough to classify the title as **Business** (and again with high confidence). The third one seems to be easy for human due to "Facebook" but the model somehow gets it wrong into **Sports**.

World: The model classifies 2 out of 3 correctly. For the wrong one, it's apparent for humans to classify it into politics or world as it includes several keywords: "China", "Muslisms" and "Camps", but the model put it under **Sci/Tech**.

Business: It classifies all three titles correctly with decent probabilities!

Cooking: The model gives somewhat funny results for this category as it's unseen in the training data. We should expect the probabilities to be low whatever the category the model gives. For the first one "After 34 Years, Gotham Bar and Grill Has a New Head Chef", it could be related to **World** if "Gotham Bar and Grill" is a global restaurant. For the second title "This Pasta May Actually Taste Better Outside", the model put it under **Sports** with 96% probability, which is hilarious. For the third one "The Dim Sum Palace That Has Reigned Over Elizabeth Street Since 1978", it's definitely related to **Cooking** but it's actually also about the family business. So the model's classification of **Business** is not that bad.

In short, classifying documents solely based on title is not an easy task as the title could be ambiguious or too abstract; plus, a document could be reasonably classified into several different categories (as we have seen in the example of "Apple Reports Declining Profits and Stagnant Growth, Again", it's in both Business and Technology), which increases the difficulty of the task.