FEDERAL FUNDS RATE PREDICTION: BERT SEQUENCE CLASSIFICATION ON FED CORPORA

AUTHOR: THEO DIMITRASOPOULOS* | ADVISOR: ZACHARY FEINSTEIN*

* Department of Financial Engineering; Stevens Institute of Technology Babbio School of Business

ABSTRACT — This paper focuses on extracting sentiment from Federal Reserve corpora in order to predict the federal funds rate. More specifically, it investigates minutes, statements, speeches and testimonies delivered by the Federal Reserve boards since 1980, which are preprocessed in short chunks that are then benchmarked against the Loughran-McDonald dictionary of financial terms for sentiment. Then, a base BERT model is trained on the preprocessed dataset and train/validation losses are recorded to estimate the accuracy of the model. The result highlights the importance of a wealth of data to train such a model. Additional finetuning or the use of a pre-trained BERT-model can provide insightful commentary on the prediction of the Federal Funds rate for use in trading strategies (mean-reversion, moving average etc.) and other applications of NLP.

KEYWORDS — *Federal Reserve, Federal Funds Rate, Interest Rate, Prediction, Sequence Classification, Bidirectional Encoders, Transformers,*

I. INTRODUCTION

The Federal Open Market Committee (FOMC) meetings aim to discuss, implement and communicate monetary policy to the markets. The Federal Funds Rate, or the formal definition of the well-known Fed interest rate could be considered a latent feature in an NLP model which attempts to extract sentiment from the data and predict the direction of the interest rate at future dates.

II. TRANSFORMERS

Transformers are a Deep Learning innovation that builds beyond recurrent neural networks with the ultimate goal of reducing processing times of even larger datasets, with equal or higher accuracy [1]. Gated RNN's were the most sophisticated model before the introduction of transformers, require that the text tokens be processed sequentially, which greatly reduces the ability to parallelize the task. In the case of a transformer, a encoder-decoder architecture is utilized in order to enlarge the scope of data analysis and allow for bidirectional processing without the need to account for the beginning and end of a token [1].

The transformer model is structured as one large matrix calculation as follows,

$$\boldsymbol{Attn(Q,K,V) = softmax_{layer}\left(\frac{(QK_T)}{\sqrt{d_k}}\right)V}$$

where Q,K,V are the vectors the of the i^{th} rows of the tokens fed into the model.

In the case of BERT in particular, the innovation is bidirectional training, or the encoder-decoder architecture mentioned above. Similarly to Next Sentence Classification, the classification task modeled in this research is performed by adding a classification layer on the transformer output for the [CLS] tokens [4].

III. RESULTS & DISCUSSION

The data was sectioned in 200-word segments in order to ease processing and was grouped by speaker. The main speakers chosen were the chairpersons of the Federal Reserve, while all other speaker content was dropped from the data. Sentiment was added to each of the word segments using the Loughran-McDonald

Dictionary of Financial Terms in order to identify the general stance towards interest rates (increase, decrease or no change) [3]. BERT was then deployed on the preprocessed data to evaluate the model on the sourced data from the Federal Reserve Archives. Fig. 1 shows the training/validation loss result after 3 rounds of training.

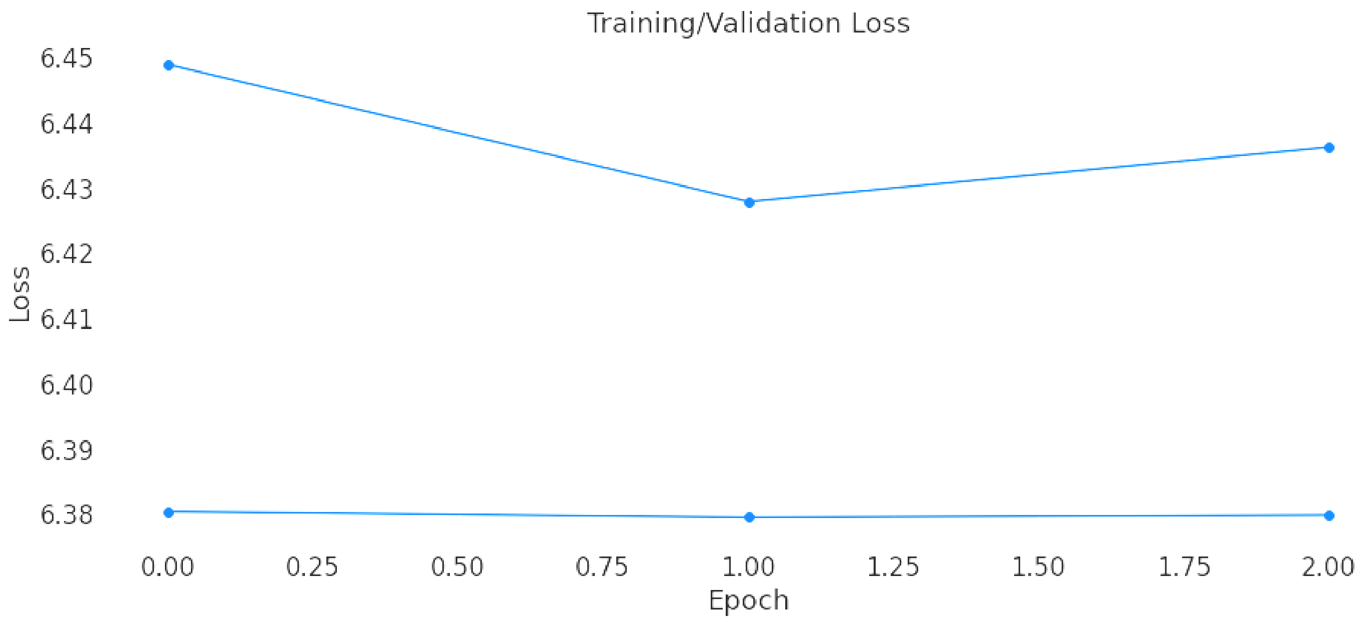

Figure 1 Training/Validation during BERT's training on the Fed Data.

The model eventually became very cumbersome to the GPU within Google Collaboratory, leading to various runtime crashes past the 3rd fold validation. In cases where a TPU or more capable processing engine is available, the training/validation loss could be further decreased. The ultimate bottleneck however is the availability of data. This might make the selection of a pre-trained model more prudent.

IV. NEXT STEPS

As seen in the analysis above, training a BERT model requires a wealth of data. In the next steps of this research, it is an imperative to source more data and perform more thorough preprocessing with various intervals in the sectioning for better parsing. There is also a lot of room for using pre-trained BERT models and fine-tuning the model's hyperparameters. Lastly, it is worth exploring other models in parallel with BERT in order to identify the one with the highest accuracy before moving forward with various integrations with trading systems (one potential path forward here is to inform a mean-reversion strategy with the sentiment extracted from the NLP methodologies).

V. ACKNOWLEDGMENTS

I would like to thank professor Feinstein for his invaluable guidance through his teachings in FE 690: Machine Learning in Finance, and his continued support in identifying the pain points and potential developments in the project proposal; The Stevens Institute of Technology Department of Financial Engineering for its provision of laptops and technological equipment to undertake this project; The Google Collaboratory for providing a streamlined platform for the testing and execution of the models; and lastly, my colleagues at the Nomura Research Institute for providing me with helpful tips and guidance on how to preprocess the data in a sound way and execute the model in this research.

VI. REFERENCES

[1] Jacob Devlin, Ming-Wei Chang. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. Accessed October 25th, 2020. Online.

[2] Transformers. HuggingFace Documentation. Accessed October 20th, 2020. Online.

[3] Takahashi, Yuki. Analyze Central Bank Announcements. Nomura Research Institute. Accessed October 20th, 2020. Online.

[4] Horev, Rani. BERT Explained: State-of-the-art language model for NLP. Accessed October 28th 2020. Online.

VII. APPENDIX

Code for data preprocessing and BERT Sequence Classification Training:

```
import sys
IN_COLAB = 'google.colab' in sys.modules
IN_COLAB

if IN_COLAB:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

!pip install numpy
!pip install pandas
!pip install tqdm
!pip install torch
!pip install scikit-plot
!pip install transformers
import pprint
import numpy as np
import pandas as pd
import datetime as dt
import os
import codecs
import io
from lxml import etree
from dateutil.relativedelta import *
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import matplotlib.ticker as ticker
import re
import pickle
from tqdm.notebook import tqdm
import nltk
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler, TensorDataset)
from transformers import BertTokenizer, BertForSequenceClassification, BertModel

def get_word_count(x):
    x = x.replace("[SECTION]", "")
    return len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', x))

def extract_r_change(x):
    if type(x) is str:
        try:
            x = dt.datetime.strptime(x, '%Y-%m-%d')
        except:
            return None

    if x in calendar.index:
        return calendar.loc[x]['RateDecision']
    else:
        return None

def extract_r(x):
    if type(x) is str:
        try:
```

```
            x = dt.datetime.strptime(x, '%Y-%m-%d')
        except:
            return None

    if x in calendar.index:
        return calendar.loc[x]['Rate']
    else:
        return None

def meeting_new(x):
    if type(x) is str:
        try:
            x = dt.datetime.strptime(x, '%Y-%m-%d')
            print(type(x))
        except:
            return None

    x = x + dt.timedelta(days=2)

    calendar.sort_index(ascending=True, inplace=True)

    if calendar['date'].iloc[0] > x:
        return None
    else:
        for i in range(len(calendar)):
            if x < calendar['date'].iloc[i]:
                return calendar['date'].iloc[i]
```

```
        return None

def chair(x):
    if type(x) is str:
        try:
            x = dt.datetime.strftime(x, '%Y-%m-%d')
            print(type(x))
        except:
            return None

    chairr = chairs.loc[chairs['FromDate'] <= x].loc[x <= chairs['ToDate']]
    return list(chairr.FirstName)[0] + " " + list(chairr.Surname)[0]

def preprocess(df, doc_type):
    if doc_type in ('statement', 'minutes', 'press', 'meeting_script'):
        is_meeting_doc = True
    elif doc_type in ('speech', 'testimony'):
        is_meeting_doc = False
    else:
        return None

    dict = {
        'type': doc_type,
        'date': df['date'],
        'title': df['title'],
        'speaker': df['speaker'],
        'word_count': df['contents'].map(get_word_count),
```

```
        'decision': df['date'].map(lambda x: extract_r_change(x) if is_meeting_doc else None),
        'rate': df['date'].map(lambda x: extract_r(x) if is_meeting_doc else None),
        'next_meeting': df['date'].map(meeting_new),
        'decision_n': df['date'].map(meeting_new).map(extract_r_change),
        'next_rate': df['date'].map(meeting_new).map(extract_r),
        'text': df['contents'].map(lambda x: x.replace('\n','').replace('\r','').strip()),
        'text_sections': df['contents'].map(lambda x: x.replace('\n','').replace('\r','').strip().split("[SECTION]")),
        'processed': df['contents']
    }

    new_df = pd.DataFrame(dict)
    new_df['decision'] = new_df['decision'].astype('Int8')
    new_df['decision_n'] = new_df['decision_n'].astype('Int8')
    return new_df

def split(text, split_len=200, overlap=50):
    l_total = []
    words = re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text)

    if len(words) < split_len:
        n = 1
    else:
        n = (len(words) - overlap) // (split_len - overlap) + 1

    for i in range(n):
        l_parcial = words[(split_len - overlap) * i: (split_len - overlap) * i + split_len]
        l_total.append(" ".join(l_parcial))
```

```
    return l_total


def split_df(df, split_len=200, overlap=50):
    split_data_list = []


    for i, row in tqdm(df.iterrows(), total=df.shape[0]):
        text_list = split(row["text"], split_len, overlap)
        for text in text_list:
            row['text'] = text
            row['word_count'] = len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text))
            split_data_list.append(list(row))


    split_df = pd.DataFrame(split_data_list, columns=df.columns)
    return split_df


chairs = pd.DataFrame(
    data=[["Volcker", "Paul", dt.datetime(1979,8,1), dt.datetime(1987,8,1)],["Greenspan", "Alan",
dt.datetime(1987,8,1), dt.datetime(2006,1,31)],["Bernanke", "Ben", dt.datetime(2006,2,1),
dt.datetime(2014,1,31)],["Yellen", "Janet", dt.datetime(2014,2,1), dt.datetime(2018,1,31)],["Powell", "Jerome",
dt.datetime(2018,2,2), dt.datetime(2022,2,2)]],
    columns=["Surname", "FirstName", "FromDate", "ToDate"])
chairs


file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/calendar.pickle', 'rb')
#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/calendar.pickle', 'rb')
calendar = pickle.load(file)
file.close()
calendar
```

```
file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/statement.pickle', 'rb')
#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/statement.pickle', 'rb')
statement_df = pickle.load(file)
file.close()
statement_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/minutes.pickle', 'rb')
#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/minutes.pickle', 'rb')
minutes_df = pickle.load(file)
file.close()
minutes_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/speech.pickle', 'rb')
#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/speech.pickle', 'rb')
speech_df = pickle.load(file)
file.close()
speech_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/testimony.pickle', 'rb')
#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/testimony.pickle', 'rb')
testimony_df = pickle.load(file)
file.close()
testimony_df

statement_clean = preprocess(statement_df, 'statement')
minutes_clean = preprocess(minutes_df, 'minutes')
```

```
speech_clean = preprocess(speech_df, 'speech')
testimony_clean = preprocess(testimony_df, 'testimony')
testimony_sections = split_df(statement_clean)
minutes_sections = split_df(minutes_clean)
testimony_sections_chair_only = split_df(testimony_chair_only_raw)
tmp_list = []
for i, row in speech_clean.iterrows():
    chairr = chair(row['date'])
    if chairr.lower().split()[-1] in row['speaker'].lower():
        row['speaker'] = chairr
        tmp_list.append(list(row))

col_names = speech_clean.columns
speech_chair_df = pd.DataFrame(data=tmp_list, columns=col_names)
speech_sections = split_df(speech_chair_df)
speech_sections.reset_index(drop=True, inplace=True)

speech_chair_df
tmp_list = []
for i, row in testimony_clean.iterrows():
    chairr = chair(row['date'])
    if chairr.lower().split()[-1] in row['speaker'].lower():
        row['speaker'] = chairr
        tmp_list.append(list(row))

col_names = testimony_clean.columns
testimony_chair_only_raw = pd.DataFrame(data=tmp_list, columns=col_names)
```

```
testimony_chair_only_raw
data_full = pd.concat([statement_clean,
                       minutes_clean,
                       speech_chair_df,
                       testimony_chair_only_raw], sort=False)
data_full.reset_index(drop=True, inplace=True)

data_sections = pd.concat([testimony_sections,
                           minutes_sections,
                           speech_sections,
                           testimony_sections_chair_only], sort=False)
data_sections.reset_index(drop=True, inplace=True)
#def save_data(df, file_name, dir_name='C:/Users/theon/Desktop/proj2/data/preprocessed/'):
def save_data(df, file_name, dir_name='/content/drive/My Drive/Colab Notebooks/proj2/data/'):
    if not os.path.exists(dir_name):
        os.mkdir(dir_name)
    file = open(dir_name + file_name + '.pickle', 'wb')
    pickle.dump(df, file)
    file.close()
    df.to_csv(dir_name + file_name + '.csv', index=True)
save_data(data_full, 'data_full')
save_data(data_sections, 'data_sections')

# BERT
class InputFeature(object):
    def __init__(self, id, input_ids, masks, segments, meta, label=None):
        self.id = id
```

```
        self.features = {
            'input_ids': input_ids,
            'input_mask': masks,
            'segment_ids': segments,
            'meta': meta
        }
        self.label = label

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
def bert_encoder(text, max_len=200):
    tokens = tokenizer.tokenize(text)
    tokens = tokens[:max_len-2]
    tokens = ["[CLS]"] + tokens + ["[SEP]"]
    ids = tokenizer.convert_tokens_to_ids(tokens)
    ids += [0] * (max_len - len(tokens))
    pad_masks = [1] * len(tokens) + [0] * (max_len - len(tokens))
    segment_ids = [0] * len(tokens) + [0] * (max_len - len(tokens))
    return ids, pad_masks, segment_ids

train_set = []
max_seq_length = 200
meta_size = 10
for index, row in tqdm(train_data_sections.iterrows(), total=train_data_sections.shape[0]):
    input_ids, masks, segments = bert_encoder(row['text'], max_seq_length)
    train_set.append(InputFeature(row.index, input_ids, masks, segments, row[nontext_columns + ['tone']],
int(row['target'])))
labels = train_data_sections['target'].astype(int).values
ids_in = np.array([data.features['input_ids'] for data in train_set])
```

```
masks_in = np.array([data.features['input_mask'] for data in train_set])
segids_in =np.array([data.features['segment_ids'] for data in train_set])
metadata_in =np.array([data.features['meta'] for data in train_set], dtype=np.float64)
labels_in = np.array([data.label for data in train_set])
train_dataset = np.zeros((len(train_data_sections), 3), dtype=np.float32)

print(metadata_in[0])
print(metadata_in[1])

class BertSeq(nn.Module):
    def __init__(self, hsize, dsize, meta_size, osize, dop=0.1):
        """
        Initialize the model
        """
        super().__init__()
        self.osize = osize
        self.dop = dop

        self.bert = BertModel.from_pretrained('bert-base-
uncased',output_hidden_states=True,output_attentions=True)
        for param in self.bert.parameters():
            param.requires_grad = True
        self.weights = nn.Parameter(torch.rand(13, 1))
        self.dop = nn.dop(dop)
        self.fc1 = nn.Linear(hsize, dsize)
        self.fc2 = nn.Linear(dsize + meta_size, osize)
        self.softmax = nn.LogSoftmax(dim=1)
```

```
    def forward(self, input_ids, nn_input_meta):
        hidden_states, attt = self.bert(input_ids)[-2:]
        batch_size = input_ids.shape[0]
        ht_cls = torch.cat(hidden_states)[:, :1, :].view(13, batch_size, 1, 768)
        att = torch.sum(ht_cls * self.weights.view(13, 1, 1, 1), dim=[1, 3])
        att = F.softmax(att.view(-1), dim=0)
        feature = torch.sum(ht_cls * att.view(13, 1, 1, 1), dim=[0, 2])
        dense_out = self.fc1(self.dop(feature))
        concat_layer = torch.cat((dense_out, nn_input_meta.float()), 1)
        out = self.fc2(concat_layer)
        return out

bert_seq = BertSeq(768, 128, meta_size, 3, dop=0.1)

learning_rate = 1e-5
num_runtime_0s = 3
batch_size = 32
patience =2
file_name = 'model'
use_skf = True
bert_hsize = 768
bert_dsize =128

def train_bert(fold, tind, vind):
    logger.info('layer{}'.format(fold))
    tids_in = torch.tensor(ids_in[tind], dtype=torch.long)
    tmask_in = torch.tensor(masks_in[tind], dtype=torch.long)
```

```
tseg_in = torch.tensor(segids_in[tind], dtype=torch.long)
tlabel_in = torch.tensor(labels_in[tind], dtype=torch.long)
tmeta_in = torch.tensor(metadata_in[tind], dtype=torch.long)
vids_in = torch.tensor(ids_in[vind], dtype=torch.long)
vmask_in = torch.tensor(masks_in[vind], dtype=torch.long)
vseg_in = torch.tensor(segids_in[vind], dtype=torch.long)
vlabel_in = torch.tensor(labels_in[vind], dtype=torch.long)
vmeta_in = torch.tensor(metadata_in[vind], dtype=torch.long)
train = torch.utils.data.TensorDataset(tids_in, tmask_in, tseg_in, tmeta_in, tlabel_in)
valid = torch.utils.data.TensorDataset(vids_in, vmask_in, vseg_in, vmeta_in, vlabel_in)
tload = torch.utils.data.DataLoader(train, batch_size=batch_size, shuffle=True)
vload = torch.utils.data.DataLoader(valid, batch_size=batch_size, shuffle=False)
bert_seq = BertSeq(bert_hsize, bert_dsize, meta_size, 3, dop=0.1)
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
bert_seq = bert_seq.to(device)
loss_fn = torch.nn.CrossEntropyLoss()
param_opt = list(model.named_parameters())
no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
adam_params = [{'params': [p for n, p in param_opt if not any(nd in n for nd in no_decay)], 'weight_decay':
0.01},{'params': [p for n, p in param_opt if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}]
opt = AdamW(adam_params, lr=learning_rate, eps=1e-6)
bert_seq.train()
best_f1 = 0.
vchoose = np.zeros((vlabel_in.size(0), 2))
segfault = 0
tlosss = []
vlosss = []
```

```
for runtime_0 in range(num_runtime_0s):
    logger.info('batch{}'.format(runtime_0+1))
    train_loss = 0.
    for i, batch in tqdm(enumerate(tload), total=len(tload), desc='Training'):
        batch = tuple(t.to(device) for t in batch)
        x_ids, x_mask, x_sids, x_meta, y_truth = batch
        y_pred = bert_seq(x_ids, x_meta)
        loss = loss_fn(y_pred, y_truth)
        opt.zero_grad()
        loss.backward()
        opt.step()
        train_loss += loss.item() / len(tload)
        logger.debug('train batch: %d, train_loss: %8f\n' % (i, train_loss))
    tlosss.append(train_loss)
    model.eval()
    vloss = 0.
    vpred = np.zeros((vlabel_in.size(0), 3))
    with torch.no_grad():
        for i, batch in tqdm(enumerate(vload), total=len(vload), desc='Validation'):
            batch = tuple(t.to(device) for t in batch)
            x_ids, x_mask, x_sids, x_meta, y_truth = batch
            y_pred = bert_seq(x_ids, x_meta).detach()
            loss = loss_fn(y_pred, y_truth)
            vloss += loss.item() / len(vload)
            vpred[i * batch_size:(i + 1) * batch_size] = F.softmax(y_pred, dim=1).cpu().numpy()

            logger.debug('validation batch: {}, vloss: {}, vpred: {}'.format(i, vloss, vpred[i * batch_size:(i + 1) *
batch_size]))
```

```
            vlosss.append(vloss)
        acc, f1 = metric(labels_in[vind], np.argmax(vpred, axis=1))
        if best_f1 < f1:
            segfault = 0
            best_f1 = f1
            vchoose = vpred
            torch.save(bert_seq.state_dict(), output_dir + 'model_fold_{}.dict'.format(fold))
        else:
            segfault += 1
        logger.info('runtime_0: %d, train loss: %.8f, valid loss: %.8f, acc: %.8f, f1: %.8f, best_f1: %.8f\n' %
            (runtime_0, train_loss, vloss, acc, f1, best_f1))
        if device == 'cuda:0':
            torch.cuda.empty_cache()
        if segfault >= patience:
            break
        model.train()
    vpred = np.zeros((vlabel_in.size(0), 3))
    sns.set(font_scale=1.5)
    plt.rcParams["figure.figsize"] = (15,6)
    plt.plot(tlosss, 'b-o')
    plt.plot(vlosss, 'b-o')
    plt.title("Training/Validation Loss")
    plt.xlabel("Runtime_0")
    plt.ylabel("Loss")
    plt.show()
    bert_seq.load_state_dict(torch.load(output_dir + 'model_fold_{}.dict'.format(fold)))
    bert_seq.eval()
```

```
    with torch.no_grad():
        for i, batch in tqdm(enumerate(vload), total=len(vload)):
            batch = tuple(t.to(device) for t in batch)
            x_ids, x_mask, x_sids, x_meta, y_truth = batch
            y_pred = bert_seq(x_ids, x_meta).detach()
            vpred[i * batch_size:(i + 1) * batch_size] = F.softmax(y_pred, dim=1).cpu().numpy()
    vchoose = vpred
    train_dataset[vind] = vchoose
    acc, f1 = metric(labels_in[vind], np.argmax(vchoose, axis=1))
    logger.info('runtime_0: best, acc: %.8f, f1: %.8f, best_f1: %.8f\n' % (acc, f1, best_f1))

if use_skf:
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    for fold, (tind, vind) in enumerate(skf.split(labels_in, labels_in)):
        train_bert(fold, tind, vind)

else:
    tbal = 0.7
    tind = np.arange(0, int(len(labels_in)*tbal))
    vind = np.arange(int(len(labels_in)*tbal), len(labels_in))
    train_bert(0, tind, vind)

def save_data(df, file_name, dir_name=train_dir):
    if not os.path.exists(dir_name):
        os.mkdir(dir_name)
    file = open(dir_name + file_name + '.pickle', 'wb')
```

```
    pickle.dump(df, file)
    file.close()
    df.to_csv(dir_name + file_name + '.csv', index=True)

save_data(train_data, 'train_data')
save_data(txt_data, 'txt_data')
save_data(train_data, 'train_data_sections')
```