NLP Final Project Report
Group 14

Task 1:

Method1 --- BERT:

Task 1 is a sequence classification problem, so we use deep learning method to finish our project. Since BERT is currently the standard model for NLP tasks, we build a sequence classification model with BERT, and finetune the model parameter from the pretrained weight of BERT.

For data preprocessing, we basically read the training data from the train.csv into our custom dataloader for this task. For the testing data, the preprocessing process is the same as the training data but without loading the label.

The model in our first method the build from the Hugginface's BertForSequenceClassification model and we load the "bert-base-uncased" for the pretrained parameter.

```
class FinCausalBert(BaseModel):
    def __init__(self):
        super().__init__()
        self.bert = BertForSequenceClassification.from_pretrained(
                        "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
                        num_labels = 2, # The number of output labels--2 for binary classification.
                                        # You can increase this for multi-class tasks.
                        output_attentions = False, # Whether the model returns attentions weights.
                        output_hidden_states = False, # Whether the model returns all hidden-states.
                       )
        # self.bert.config.__dict__['hidden_dropout_prob'] = 0.3
        # self.fc = nn.Sequential(
        #                         nn.Linear(768, 256),
        #                         nn.ReLU(),
        #                         nn.Linear(256, 128),
        #                         nn.ReLU(),
        #                         nn.Linear(128, 1),
        #                         nn.Sigmoid()
        #                          )

    def forward(self, x, labels=None):
        output = self.bert(x, labels=labels)
        #output = output[0].mean(1)
        # output = self.fc(output)
        return output
```

Method2 --- RoBerta:

Since our method1 couldn't reach the bonus baseline, we tried the second method. We still used the transformer pretrained model, and this time we used RoBerta. For preprocessing, it's same as the method1, just changed the tokenizer to the RoBerta's tokenizer. Then the model, input sequence usage of Bert and RoBerta is almost same, so we loaded the "RobertaForSequenceClassification" model and load "roberta-base" as the pretrained parameters. We'd try to use RoBerta and add dense layer to classification, but the score is not high.

```
class FinCausalBert(BaseModel):
    def __init__(self):
        super().__init__()

        self.roberta = RobertaForSequenceClassification.from_pretrained('roberta-base')

    def forward(self, x, labels=None):
        output = self.roberta(x, labels=labels)

        return output
```

Comparision of different parameter/model:

Bert

learning rate	1e-5	3e-5	7e-5
public	0.96014	0.95940	0.96236
private	0.95424	0.96088	0.95498

Roberta

optimizer: AdamW (weight_decay = 0.1, eps = 1e-6) lr is presented below

learning rate	1e-5	2e-5	3e-5
public	0.97195	0.97121	0.96383
private	0.96088	0.96236	0.95645

How to run our code:

code: https://github.com/vincent861223/FinCausal

Please check README for detail.

Task 2:

Method1 --- BERT:

Task 2 is basically a question answering problem, since BERT is currently the standard model for NLP, we tried to finish this task by building a model for question answering and load the BERT pretrained parameter for fine-tuning.

Task 2 can be formulated as two question answering problem, one question is the “Casual” and the other is the “Effect”. So be build two question answering model for this task, one answers the “Causal” and the other answers the “Effect”.

For data preprocessing we read the training data and the validation data from the file. Since BERT is trained by word-level sequence instead of character-level sequence, we need to tokenize the Text sentence into word-level tokens, and we also need to convert the character-based position provided to word-level position.

Our model is formed by 2 BERT model, one for Causal sentence, and the other for Effect sentence.

```
class FinCausalBert(BaseModel):
    def __init__(self):
        super().__init__()
        self.cause_bert = BertForQuestionAnswering.from_pretrained('bert-base-cased')
        self.effect_bert = BertForQuestionAnswering.from_pretrained('bert-base-cased')
        # self.fc = nn.Sequential(
        #                         nn.Linear(768, 256),
        #                         nn.ReLU(),
        #                         nn.Linear(256, 128),
        #                         nn.ReLU(),
        #                         nn.Linear(128, 4),
        #                          )

    def forward(self, input_ids, cause_start=None, cause_end=None, effect_start=None, effect_end=None):
        cause_output = self.cause_bert(input_ids=input_ids, start_positions=cause_start, end_positions=cause_end)
        effect_output = self.effect_bert(input_ids=input_ids, start_positions=effect_start, end_positions=effect_end)
        if(cause_start != None):
            loss = cause_output[0] + effect_output[0]
            score = {'cause_start': cause_output[1], 'cause_end': cause_output[2], 'effect_start': effect_output[1], 'effect_end':effect_output[2]}
            return loss, score
        else:
            score = {'cause_start': cause_output[0], 'cause_end': cause_output[1], 'effect_start': effect_output[0], 'effect_end':effect_output[1]}
            return score
```

When predicting, our model need to first convert the word-based position to character-based position. Then, we can use the predicted position by the model to find the Causal or Effect sentence. However, because the sentence is tokenized, we can not reconstruct the sentence back to the original sentence. For example, original text is “$17.7”, then our model will predict “$ 17 . 7”. This is why our exact match score is 0.0.

Method2 --- CRF:

The second method is CRF, it's full name is Conditional Random Fields. CRF is a probabilistic discriminative model that has a wide range of applications in NLP, such as task specific predictions.

For preprocessing we first tokenized words by NLTK, then tagged every tokens with tag name C(cause) / E(effect) / _(else), these tags are our labels. Then tagged POS with "nltkPOS". For training we import "pycrfsuite" and used it's trainer to train the model. Below is our input

```
features = {
    'bias': 1.0,
    'word.lower()': word.lower(),
    'word[-3:]': word[-3:],
    'word[-2:]': word[-2:],
    'word.isupper()': word.isupper(),
    'word.istitle()': word.istitle(),
    'word.isdigit()': word.isdigit(),
    'postag': postag,
    'postag[:2]': postag[:2],
}
```

```
def make_data(df):
    lodict_ = []
    for rows in df.itertuples():
        list_ = [rows[2], rows[3], rows[4]]
        map1 = ['sentence', 'cause', 'effect']
        dict_ = s2dict(list_, map1)
        lodict_.append(dict_)

    map_ = [('cause', 'C'), ('effect', 'E')]
    ##

    ##
    hometags = make_causal_input(lodict_, map_)

    ds = [[(k, v) for k, v in x_] for x_ in hometags]
    logging.debug(pformat(ds))
    postags = nltkPOS([i['sentence'] for i in lodict_])
    tokens = ((token for token in word_tokenize(sent)) for sent in df.Text.tolist())

    data = []
    for i, (j, k) in enumerate(zip(hometags, postags)):
        data.append([(w, pos, label) for (w, label), (word, pos) in zip(j, k)])

    X = [extract_features(doc) for doc in data]
    y = [get_multi_labels(doc) for doc in data]

    return X, y, tokens
```

When predict, we used pycrfsuite's tagger to predict every words tag (C/E/_), with these tags we can score these predict.

* * Our task2_CRF is modified from the baseline TA provided.

Comparision of different parameter/model:

Bert model

```
F1: 0.805301
Recall: 0.598316
Precision: 0.556317
ExactMatch: 0.000000
```

CRF model:

```
F1: 0.837225
Recall: 0.837181
Precision: 0.841004
ExactMatch: 0.690344
```

How to run our code:

code: https://github.com/vincent861223/FinCausal

Please check README for detail.

Things we learned:

Before this project we has never used BERT, we only tried some simple sequence to sequence model. During this project we learned to use BERT to solve different tasks, such as sequence classification in task 1 and question answering problem in task 2. We also tried new state of the art model -- Roberta and achieve better performance. We also tried traditional method in NLP such as CRF to solve problem. We learned a lot from this amazing project.

Team Cooperation:

task1_method1: 林晉辰

task1_method2: 陳柏儒

task2_method1: 謝承軒

task2_method2: 蔡易霖