Project Review

Background AI CUP NLP NER Contest

Motivation Jupyter Notebook Problems

Problem 1 依賴 Jupyter notebook 的特性一條龍式地完成一個方法、專案，而沒有使用到任何軟體設計技巧。

Problem 2 為了想要保存每個 NOTEBOOK 的特徵、成果，用土法煉鋼的方式，寫出好幾個差不多的程式。

Initial Structure

CRF

data generate...
...
preprocess... (encoded)
...
training...
(sklearn-style)
...
predicting... (raw text)
uploading... (str)

BiLSTM

data generate...
...
preprocess... (text)
training...
(tensorflow-style)
...
predicting... (encoded)
uploading... (pandas)

BiLSTM-CRF

data generate...
...
preprocess... (encoded)
...
...
training...
(tensorflow-style2)
...
predicting... (text)
uploading... (pandas)

Reconstructuring

Layer1 Code Smell

Layer2 Polymorphism

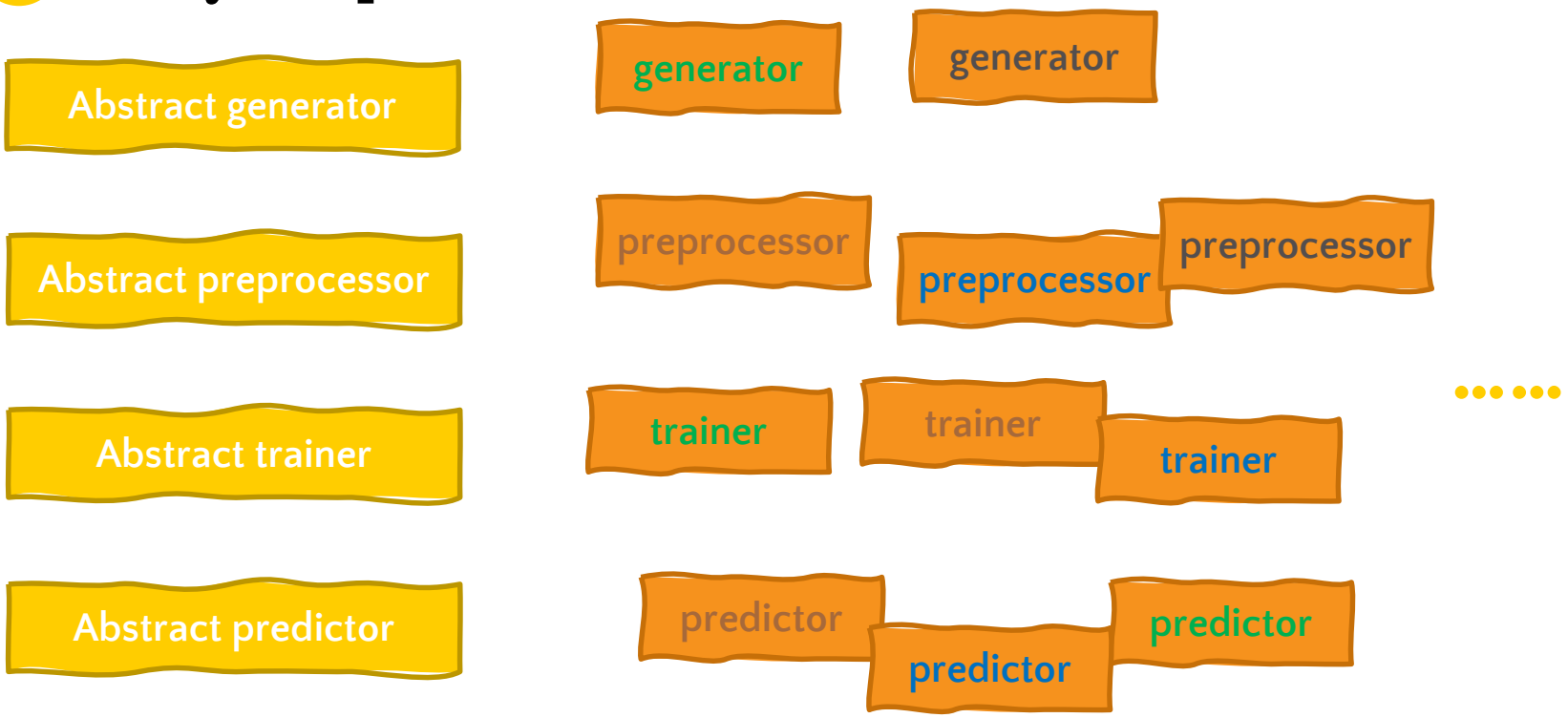
Layer3 Flags

Polymorphism & flags

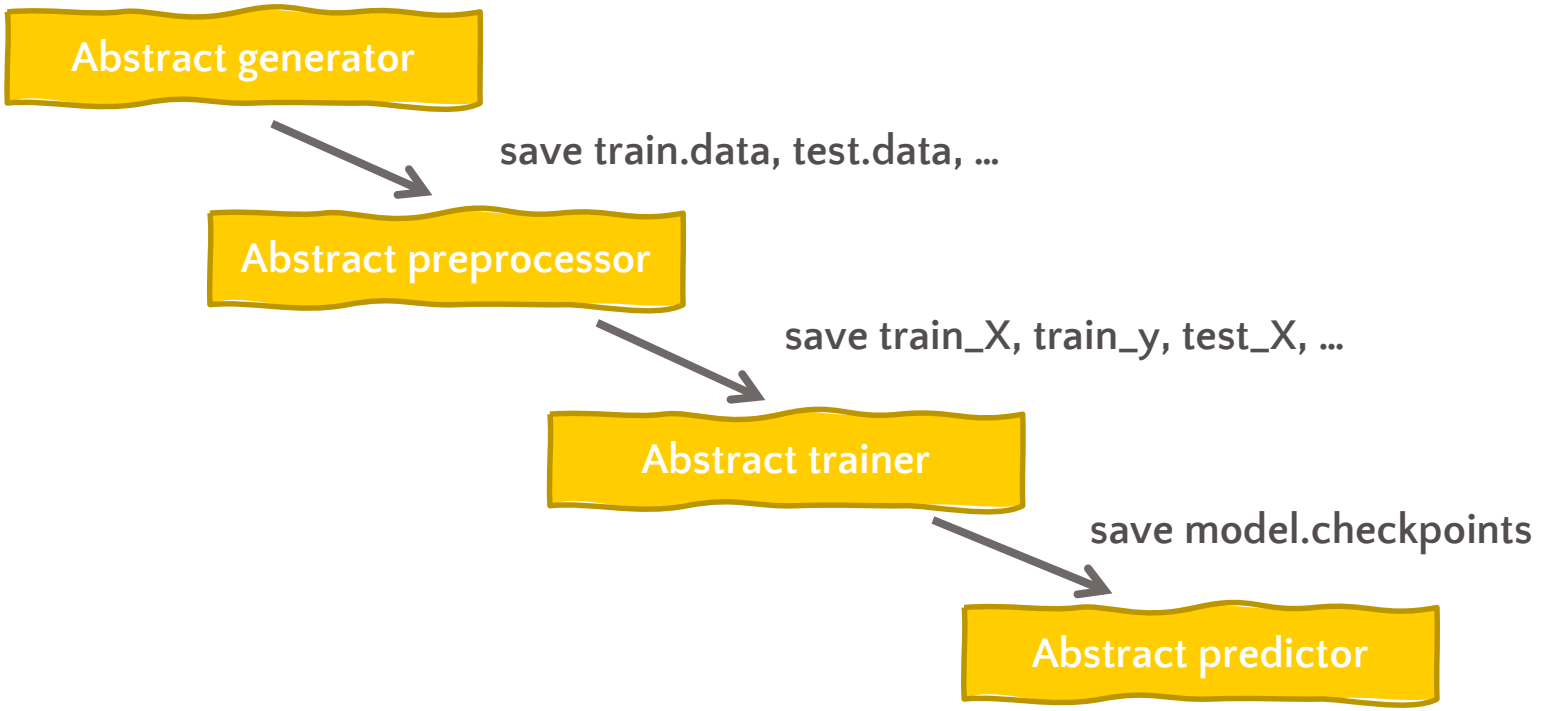
- Split each notebook into *four parts*.
- *Save data* for the next pipeline.
- A *central control room* to operate all actions.

Polymorphism illustration

Polymorphism illustration



Polymorphism illustration



Polymorphism

```python
class DataGenerator(ABC):
    @abstractmethod
    def outputTrainData(self, raw_train, output_train):
        ...

    @abstractmethod
    def outputTestData(self, raw_test, output_test):
        ...
```

```python
class DataPreprocessor(ABC):
    def __init__(self, train_data_path, test_data_path):
        self.train_data_path = train_data_path
        self.test_data_path = test_data_path

    @abstractmethod
    def outputTrainArrays(self, train_X_path, train_y_path):
        ...

    @abstractmethod
    def outputTestArray(self, test_X_path, test_mapping_path):
        ...
```

```python
@dataclass
class NerPredictor(ABC):
    model_data_path: str
    checkpoint_path: str
    output_path: str
    embedding_size: int
    hidden_nums: int
    learning_rate: float

    @abstractmethod
    def predict(self):
        ...

    @abstractmethod
    def output(self):
        ...

    def run(self):
        print("Start predicting ... ")
        self.predict()

        print("Start outputing")
        self.output()
```

```python
@dataclass
class NerTrainer(ABC):
    train_data_path: str
    model_data_path: str
    checkpoint_path: str
    checkpoint_keep: int
    max_sentence_length: int
    batch_size: int
    embedding_size: int
    hidden_nums: int
    epochs: int
    learning_rate: float
    isVisualize: bool

    @abstractmethod
    def tokenize(self):
        ...

    @abstractmethod
    def train(self):
        ...

    @abstractmethod
    def visualize(self):
        ...

    def run(self):
        print("Start tokenization ... ")
        self.tokenize()
        print("Start training ... ")
        self.train()

        if self.isVisualize:
            self.visualize()
```

Flags

A "design pattern" well suited for use with AI

absl flags module

```
python test.py --model=crf --epoch=15 --train=True
```

```python
from absl import app, flags
FLAGS = flags.FLAGS

flags.DEFINE_string("model", "bert", "model to run")
flags.DEFINE_integer("epoch", 20, "epoch count")
flags.DEFINE_bool("train", False, "model trainable")

def main(argv):
    print("model:", FLAGS.model)
    print("epochs:", FLAGS.epoch)
    print("trainable:", FLAGS.train)

if __name__ == "__main__":
    app.run(main)
```

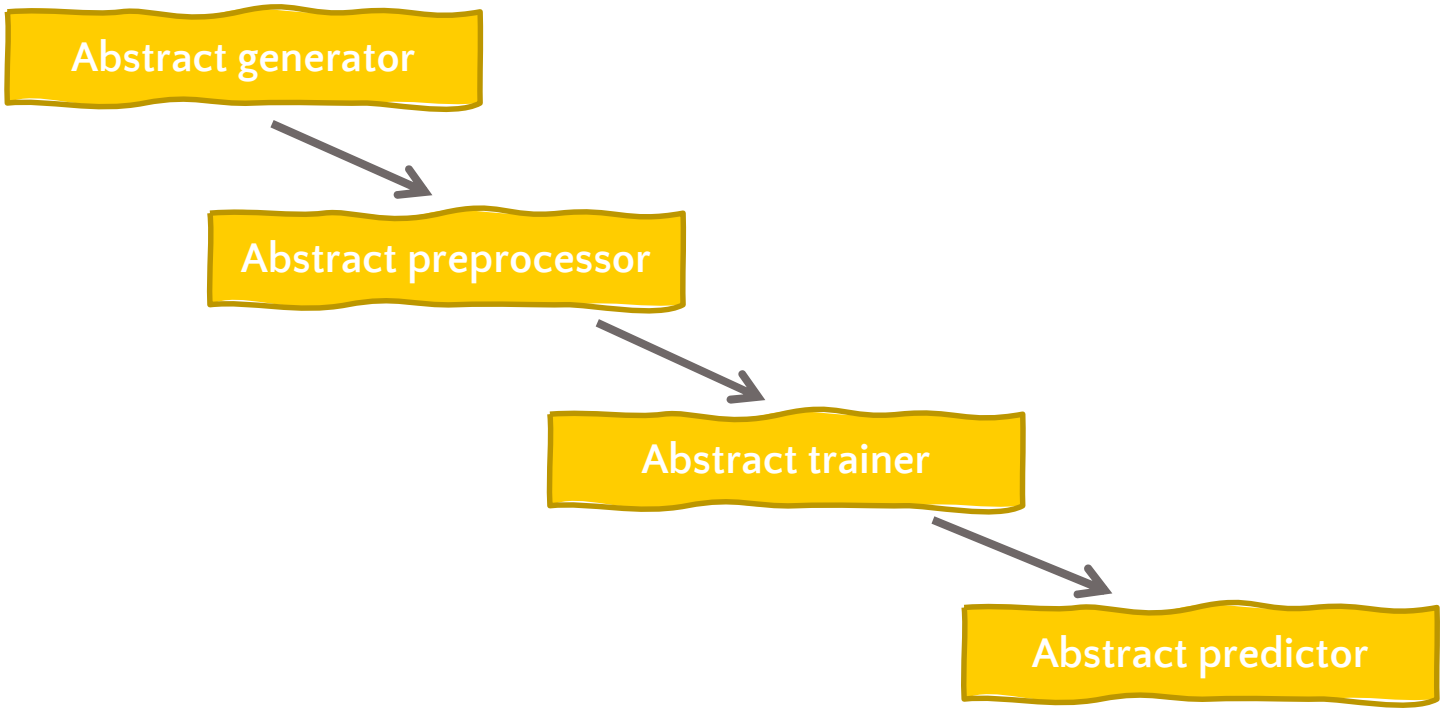
Without arguments

model: bert
epochs: 20
trainable: False

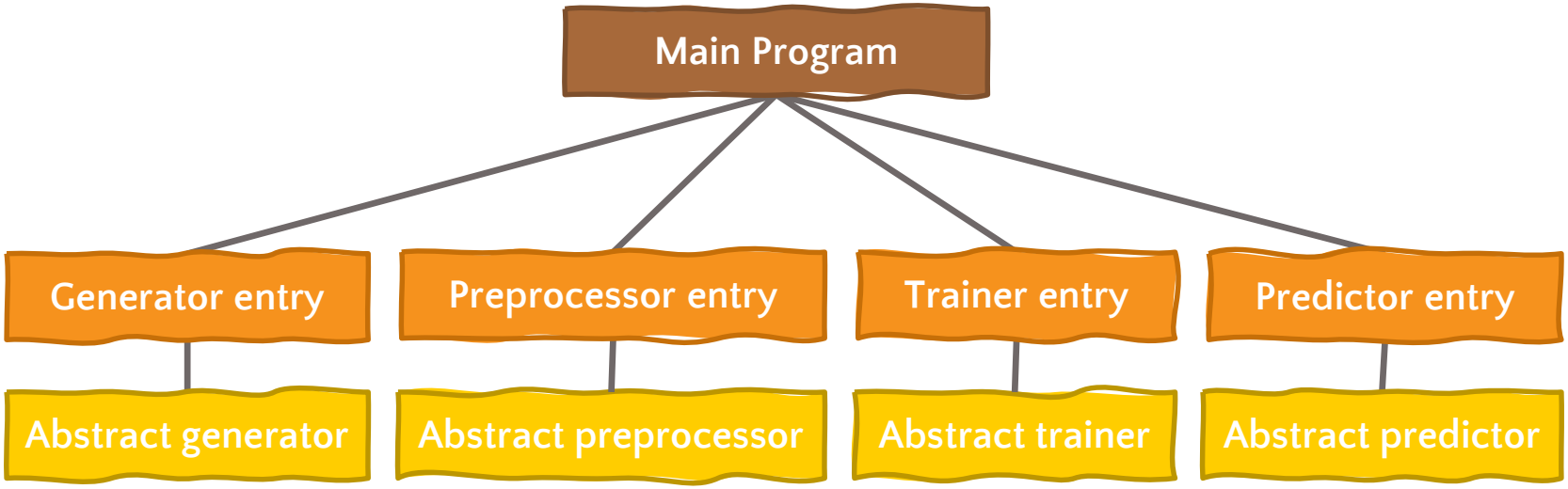
With arguments

model: crf
epochs: 15
trainable: True

Flags illustration



Flags illustration



Flags control center

In [ ]:
```
from pathlib import Path

ROOT_PATH = Path.cwd().parent.parent

RAW_TRAIN_DATA_PATH = "dataset/raw_data/train.txt"
RAW_TEST_DATA_PATH = "dataset/raw_data/test.txt"

TRAIN_DATA_PATH = "dataset/ner_data/train.data"
TEST_DATA_PATH = "dataset/ner_data/test.data"

TRAIN_GRAINED_DATA_PATH = "dataset/ner_data/train_grained.data"
TEST_GRAINED_DATA_PATH = "dataset/ner_data/test_grained.data"

MODEL = [
    "CRF",
    "SVM",
    "PYTORCH_CRF",
    "BILSTM_CRF",
    "BERT_CRF",
    "BERT_BILSTM_CRF"
]

MODEL_SELECT = 3

%set_env PYTHONPATH=$ROOT_PATH
```

In [ ]:
```
# Generate train, test NER format Data

!python data_generator.py \
            --RAW_TRAIN_DATA_PATH=$ROOT_PATH/$TRAIN_DATA_PATH \
            --RAW_TEST_DATA_PATH=$ROOT_PATH/$TEST_DATA_PATH \
            --TRAIN_DATA_PATH=$ROOT_PATH/$TRAIN_GRAINED_DATA_PATH \
            --TEST_DATA_PATH=$ROOT_PATH/$TEST_GRAINED_DATA_PATH \
            --OUTPUT_TYPE=split
```

In [ ]:
```
# Preprocess and generate trainable datasets

!python data_preprocessor.py \
            --TRAIN_DATA_PATH=$ROOT_PATH/$TRAIN_GRAINED_DATA_PATH \
            --TEST_DATA_PATH=$ROOT_PATH/$TEST_GRAINED_DATA_PATH \
            --RAW_TEST_DATA_PATH=$ROOT_PATH/$RAW_TEST_DATA_PATH \
            --MODEL_DATA_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/data/
```

In [ ]:
```
# Tokenize and training process, use the dataset pickled from data_preprocessor

!python ner_trainer.py \
            --MODEL={MODEL[MODEL_SELECT]} \
            --TRAIN_DATA_PATH=$ROOT_PATH/$TRAIN_GRAINED_DATA_PATH \
            --MODEL_DATA_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/data/ \
            --MODEL_CHECKPOINT_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/checkpoint/ \
            --CHECKPOINT_KEEP=3 \
            --SENTENCE_MAX_LENGTH=32 \
            --BATCH_SIZE=16 \
            --EMBEDDING_SIZE=300 \
            --HIIDEN_NUMS=512 \
            --EPOCHS=1 \
            --LEARNING_RATE=1e-3
```

In [ ]:
```
# Predicting process and export the results, use the model generated from training checkpoints

!python ner_predictor.py \
            --MODEL={MODEL[MODEL_SELECT]} \
            --MODEL_DATA_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/data/ \
            --MODEL_CHECKPOINT_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/checkpoint/ \
            --MODEL_OUTPUT_PATH=$ROOT_PATH/model/{MODEL[MODEL_SELECT]}/output/ \
            --EMBEDDING_SIZE=300 \
            --HIIDEN_NUMS=512 \
            --LEARNING_RATE=1e-3
```

Flags entry example

```python
FLAGS = flags.FLAGS

flags.DEFINE_string("MODEL", "CRF", "model")
flags.DEFINE_string("TRAIN_DATA_PATH", "dataset/ner_data/train.data", "
train data path")
flags.DEFINE_string("MODEL_DATA_PATH", "model/CRF/data/", "model train data path"
)
flags.DEFINE_string("MODEL_CHECKPOINT_PATH", "model/CRF/checkpoint/", "
checkpoint path")
flags.DEFINE_integer("CHECKPOINT_KEEP", 3, "checkpoint max-to-keep")
flags.DEFINE_integer("SENTENCE_MAX_LENGTH", 32, "sentence max length")
flags.DEFINE_integer("BATCH_SIZE", 128, "batch size")
flags.DEFINE_integer("EMBEDDING_SIZE", 512, "embedding size")
flags.DEFINE_integer("HIIDEN_NUMS", 512, "hidden nums")
flags.DEFINE_integer("EPOCHS", 20, "epochs")
flags.DEFINE_float("LEARNING_RATE", 1e-3, "learning rate")
flags.DEFINE_bool("VISUALIZE", True, "visualize or not")

physical_devices = tf.config.list_physical_devices("GPU")
tf.config.experimental.set_memory_growth(physical_devices[0], enable=True)


def buildBiLstmCrfTrainer():
    if FLAGS.MODEL == "BILSTM_CRF":
        return BiLstmCrfTrainer(
            FLAGS.TRAIN_DATA_PATH,
            FLAGS.MODEL_DATA_PATH,
            FLAGS.MODEL_CHECKPOINT_PATH,
            FLAGS.CHECKPOINT_KEEP,
            FLAGS.SENTENCE_MAX_LENGTH,
            FLAGS.BATCH_SIZE,
            FLAGS.EMBEDDING_SIZE,
            FLAGS.HIIDEN_NUMS,
            FLAGS.EPOCHS,
            FLAGS.LEARNING_RATE,
            FLAGS.VISUALIZE,
        )
```

```python
def main(_):
    if not Path(FLAGS.MODEL_DATA_PATH).exists():
        Path(FLAGS.MODEL_DATA_PATH).mkdir(parents=True)

    if not Path(FLAGS.MODEL_CHECKPOINT_PATH).exists():
        Path(FLAGS.MODEL_CHECKPOINT_PATH).mkdir(parents=True)

    trainer_list = {
        "CRF": buildCrfTrainer(),
        "SVM": buildSvmTrainer(),
        "PYTORCH_CRF": buildPytorchCrfTrainer(),
        "BILSTM_CRF": buildBiLstmCrfTrainer(),
        "BERT_CRF": buildBertCrfTrainer(),
        "BERT_BILSTM_CRF": buildBertBilstmCrfTrainer(),
    }

    trainer = trainer_list[FLAGS.MODEL]
    trainer.run()


if __name__ == "__main__":
    app.run(main)
```

Summary

- Hardware required, otherwise still use colab.
- Prepare in advance. Specify the data format.