S7101_compare_models

1 Boilerplate

2 Imports

2.1 prod: NVM

```
from nvm import disp_df
from nvm import repr_df
from nvm import rdf
from nvm import ddf
from nvm import clean_str
from nvm.aux_str import CLEAN_STR_MAPPINGS_LARGE as maps0
from nvm.aux_str import REGEX_ABC_DASH_XYZ_ASTERISK as re0
from nvm.aux_pandas import fix_column_names
```

2.2 prod: Basics

```
import os
import pathlib
import numpy as np
import pandas as pd
import re
import json
import yaml
import srsly
import uuid
import random
import numbers
from collections import OrderedDict
from contextlib import ExitStack
import warnings
# warnings.warn("\nwarning")
from hashlib import md5
import humanfriendly as hf
import time
import datetime as dt
from pytz import timezone as tz
```

```
tz0 = tz("Europe/Berlin")
from glob import glob
from tqdm import tqdm
import logging
log0.info("DONE: basic imports")
```

```
I: DONE: basic imports
```

2.3 prod: Extra imports and settings

```
from contexttimer import Timer
import textwrap

HOME = pathlib.Path.home()

tqdm.pandas()

import matplotlib
from matplotlib import pyplot as plt
# import seaborn as sns
# import plotly.graph_objects as go
# import plotly.express as px

# get_ipython().run_line_magic("matplotlib", "qt")
# get_ipython().run_line_magic("matplotlib", "inline")

with Timer() as elapsed:
    time.sleep(0.001)

log0.info(hf.format_timespan(elapsed.elapsed))

log0.info("DONE: extra imports and settings")
```

```
I: 0 seconds
I: DONE: extra imports and settings
```

3 Extra Imports

3.1 prod: More extra imports and settings

```
log0.info("DONE: more extra imports and settings")
```

```
I: DONE: more extra imports and settings
```

4 Notes

5 Process

5.1 prod: Load data

```
dir0 = "../../data/d0088_trained_models/"
dir0 = pathlib.Path(dir0)
# dir0.mkdir(mode=0o700, parents=True, exist_ok=True)
assert dir0.exists(), f"The data directory dir0={str(dir0)} not found!"

glob0 = dir0.glob("*/metadata.yaml")
glob0 = sorted(list(glob0))
log0.info(f"{len(glob0)}")

data0 = []
for item0 in glob0:
    data0.append(srsly.read_yaml(item0))

log0.info(f"{len(data0)}")

df0 = pd.DataFrame.from_records(data0)

drop_cols = [
    "data_dir",
    "goldstd",
    "extn",
    "status",
    "testing",
    "init_metrics",
]
df0.drop(columns=drop_cols, inplace=True, errors="ignore")
df0["out_dir"] = df0.out_dir.apply(lambda x: x.split("/")[-1])

log0.info(f"{df0.shape = }")
disp_df(df0)
```

5.2 Check dictionary in columns

```
col0 = "train_metrics"
col2 = "test_metrics"

keys0 = df0[col0][0].keys()
keys2 = df0[col2][0].keys()

keys0 = [item for item in keys0]
keys2 = ["_".join(item.split("_")[1:]) for item in keys2]


print(keys0)
print(keys2)
```

```
print(srsly.json_dumps(df0[col0][0], indent=2))
print(srsly.json_dumps(df0[col2][0], indent=2))
```

5.3 Check Series

```
log0.info(f"{df0.shape = }")

col0 = "gold_metrics"
disp_df(df0[col0].apply(pd.Series).add_prefix("GOLD_"))
```

5.4 Check columns

```
for col0 in df0.columns:
    print(f"    \"{col0}\",")
```

5.5 Data wrangle

```
dict_cols = [
    "train_metrics",
    "eval_metrics",
    "test_metrics",
    "gold_metrics",
]
df2 = df0.copy()
for col0 in dict_cols:
    prefix = col0.split("_")[0] + "_"
    log0.info(f"{prefix}")
    se0 = df2[col0].apply(pd.Series).copy()
    se0 = se0.add_prefix(prefix.upper())
    df2 = pd.concat([df2, se0], axis=1).copy()
    df2.columns = df2.columns.str.replace(prefix.upper()+"test_", prefix.upper())
    df2.columns = df2.columns.str.replace(prefix.upper()+"eval_", prefix.upper())
    df2.drop(columns=[col0], inplace=True, errors="ignore")

df2 = df2.dropna(how="all", axis=1)

df2 = df2.loc[:, ~df2.columns.str.endswith("_runtime")]
df2 = df2.loc[:, ~df2.columns.str.endswith("_samples_per_second")]
df2 = df2.loc[:, ~df2.columns.str.endswith("_steps_per_second")]

df2.sort_values(by=["dataset", "out_dir"], inplace=True)

log0.info(f"{df2.shape = }")
disp_df(df2)
```

5.6 New columns

```
for col0 in df2.columns:
    print(f"    \"{col0}\",")
```

```
    "num_epochs",
    "batch_size",
    "random_state",
    "seed",
    "max_length",
    "dataset",
    "model_name",
    "date",
    "out_dir",
    "full_len",
    "train_len",
    "eval_len",
    "test_len",
    "gold_len",
    "finished",
    "TRAIN_loss",
    "TRAIN_rmse",
    "TRAIN_mse",
    "TRAIN_mae",
    "TRAIN_r2",
    "TRAIN_max_err",
    "TRAIN_exp_var",
    "TRAIN_epoch",
    "EVAL_loss",
    "EVAL_rmse",
    "EVAL_mse",
    "EVAL_mae",
    "EVAL_r2",
    "EVAL_max_err",
    "EVAL_exp_var",
    "EVAL_epoch",
    "TEST_loss",
    "TEST_rmse",
    "TEST_mse",
    "TEST_mae",
    "TEST_r2",
    "TEST_max_err",
    "TEST_exp_var",
    "GOLD_loss",
    "GOLD_rmse",
    "GOLD_mse",
    "GOLD_mae",
    "GOLD_r2",
    "GOLD_max_err",
```

```
    "GOLD_exp_var",
```

5.7 DF4

```
cols4 = [
    "seed",
    "dataset",
    "model_name",
    "date",
    "num_epochs",
    "batch_size",
    "out_dir",
    "random_state",
    # "full_len",
    # "train_len",
    # "eval_len",
    "test_len",
    "gold_len",
    # "finished",
    # "TRAIN_epoch",
    # "EVAL_epoch",
    # "TRAIN_loss",
    # "EVAL_loss",
    "TEST_loss",
    "GOLD_loss",
    # "TRAIN_rmse",
    # "EVAL_rmse",
    "TEST_rmse",
    "GOLD_rmse",
    # "TRAIN_mse",
    # "EVAL_mse",
    "TEST_mse",
    "GOLD_mse",
    # "TRAIN_mae",
    # "EVAL_mae",
    "TEST_mae",
    "GOLD_mae",
    # "TRAIN_r2",
    # "EVAL_r2",
    "TEST_r2",
    "GOLD_r2",
    # "TRAIN_max_err",
    # "EVAL_max_err",
    "TEST_max_err",
    "GOLD_max_err",
    # "TRAIN_exp_var",
    # "EVAL_exp_var",
    "TEST_exp_var",
    "GOLD_exp_var",
]
df4 = df2[cols4].copy()
log0.info(f"{df4.shape = }")
disp_df(df4)
```

5.8 Check models for removal

```
df_keep = df4[df4.batch_size==64]
log0.info(f"{df_keep.shape = }")
disp_df(df_keep.sort_values(by=["date"]))
```

5.9 Check seed values

```
disp_df(df4[df4.batch_size>=64].seed.value_counts())
# disp_df(df4[df4.batch_size>=64].testing.value_counts())
```

5.10 Filter

```
cols8 = dict(
    # finished="finished",
    # out_dir="out_dir",
    random_state="random_state",
    seed="seed",
    date="date",
    model_name="Base model",
    dataset="Fine-tuning dataset",
    num_epochs="Number of epochs",
    batch_size="Batch size",
    # full_len="Full fine-tuning",
    train_len="Traininig data",
    eval_len="Evaluation data",
    test_len="Test data",
    gold_len="Gold standard",
    # TRAIN_epoch="TRAIN_epoch",
    # EVAL_epoch="EVAL_epoch",
    # TRAIN_loss="TRAIN_loss",
    # EVAL_loss="EVAL_loss",
    # TEST_loss="TEST_loss",
    # GOLD_loss="GOLD_loss",
    TRAIN_rmse="RMSE train",
    EVAL_rmse="RMSE eval",
    TEST_rmse="RMSE test",
    GOLD_rmse="RMSE gold",
    # TRAIN_mse="TRAIN_mse",
    # EVAL_mse="EVAL_mse",
    # TEST_mse="TEST_mse",
    # GOLD_mse="GOLD_mse",
    # TRAIN_mae="TRAIN_mae",
    # EVAL_mae="EVAL_mae",
    # TEST_mae="TEST_mae",
    # GOLD_mae="GOLD_mae",
    # TRAIN_r2="TRAIN_r2",
    # EVAL_r2="EVAL_r2",
    # TEST_r2="TEST_r2",
```

```
    # GOLD_r2="GOLD_r2",
    # TRAIN_max_err="TRAIN_max_err",
    # EVAL_max_err="EVAL_max_err",
    # TEST_max_err="TEST_max_err",
    # GOLD_max_err="GOLD_max_err",
    # TRAIN_exp_var="TRAIN_exp_var",
    # EVAL_exp_var="EVAL_exp_var",
    # TEST_exp_var="TEST_exp_var",
    # GOLD_exp_var="GOLD_exp_var",
)

df8 = df2[cols8.keys()].copy()


# filter by fine-tuning data
datasets = ["ft0x", "ft1x", "ft2x", "ft3x", "ft4x"]
datasets = ["ft0x", "ft1x", "ft2x", "ft3x"]
df8 = df8[df8.dataset.isin(datasets)]


# filter by batch size
batch_sizes = [64]
df8 = df8[df8.batch_size.isin(batch_sizes)]


# sort rows
by=["TEST_rmse"]
by=[ "model_name", "dataset","date"]
by=["GOLD_rmse"]
ascending = True
df8 = df8.sort_values(by=by, ascending=ascending)

df8["dataset"] = df8.dataset.map({"ft0x": "FT0", "ft1x": "FT1", "ft2x": "FT2", "ft3x": "FT3", "ft4x"

df8.rename(
columns = cols8,
    inplace=True,
)

df8.round(4).to_excel("s7101_compare_models.xlsx")

log0.info(f"{df8.shape = }")
disp_df(df8)
```

```
random_state
42    8
Name: count, dtype: int64I: df8.shape = (8, 15)
```

	random_state	seed	date	Base model	Fine-tuning dataset	Number of
47	42	42	20230523T145242	roberta-base	FT3	
45	42	42	20230523T134342	roberta-base	FT2	
46	42	42	20230523T140848	bert-base-uncased	FT3	
43	42	42	20230523T125926	roberta-base	FT1	

42	42	42	20230523T124016	bert-base-uncased	FT1
44	42	42	20230523T131837	bert-base-uncased	FT2
41	42	42	20230523T123140	roberta-base	FT0
40	42	42	20230523T122304	bert-base-uncased	FT0