WeatherAus

February 9, 2023

1 Weather Aus

Data taken from https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package

[1]:

```
import sys
import pandas as pd
import numpy as np
import torch
from torch import distributions
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from scipy import optimize
from copy import deepcopy
import warnings
import pickle
# setting path
sys.path.append('..')
from functions.np_classifier_torch import cutoff_bin, power_alpha_calc  # noqa:␣
  ↪E402
from functions.estimators_torch import kliep_multi_dim_sep_wrap,␣
  ↪kliep_multi_dim_naive_sep_wrap, kliep_miss_wrap
from functions.estimators_torch import kliep_multi_dim_sep_imp_wrap,␣
  ↪kliep_multi_dim_imp_wrap
from functions.objective_funcs_torch import get_dat_vals_impute,␣
  ↪get_dat_vals_multidim
from functions.pipeline_funcs import missing_pipeline, full_pipeline, get_ci,␣
  ↪progress, create_standard_miss_func

plt.rcParams["figure.facecolor"] = "White"
plt.rcParams["savefig.facecolor"] = "White"
unif=distributions.Uniform(0,1)
page_width=9
width_height_ratio=3/4
font = {'family' : 'Helvetica',
        'weight' : 'normal',
        'size'   : 14}
```

```
plt.rc('font', **font)

def convert_factor(df,column):
    values=df[column].dropna().unique()
    for i in np.arange(1,len(values)):
        df[column+"_"+values[i]]=np.where(df[column]==values[i],1,0)
    df.drop(column,axis=1,inplace=True)


def trim_series(series,niqr=5):
    iqr = np.subtract(*np.nanpercentile(series, [75, 25]))
    med = np.nanmedian(series)
    out=series
    out = np.where(out>=med-niqr*iqr,out,med-niqr*iqr)
    out = np.where(out<=med+niqr*iqr,out,med+niqr*iqr)
    return out
```

[3]:

```
# Read in data
df = pd.read_csv("../real_world_data/weatherAUS.csv", header=0)
df = df.loc[df["RainTomorrow"].notnull()]
convert_factor(df, "WindGustDir")
convert_factor(df, "WindDir9am")
convert_factor(df, "WindDir3pm")
convert_factor(df, "RainToday")
convert_factor(df, "RainTomorrow")


df_fin = df.drop(["Date", "Location"], axis=1)

cols_to_trim = [
    "MinTemp", "MaxTemp", "Rainfall", "Evaporation", "Sunshine",
    "WindGustSpeed", "WindSpeed9am", "WindSpeed3pm", "Humidity9am",
    "Humidity3pm", "Pressure9am", "Pressure3pm", "Cloud9am", "Cloud3pm",
    "Temp9am", "Temp3pm"]
for column in cols_to_trim:
    df_fin[column] = trim_series(df_fin[column])

df_fin_nodrop = df_fin.drop("RainTomorrow_Yes", axis=1)
null_df = df_fin[df_fin["RainTomorrow_Yes"]
                 == 1].drop("RainTomorrow_Yes", axis=1)
alt_df = df_fin[df_fin["RainTomorrow_Yes"]
                == 0].drop("RainTomorrow_Yes", axis=1)


# Get normalisation terms from training tensor
std = np.nanstd(df_fin_nodrop.to_numpy(), axis=0)
m = np.nanmean(df_fin_nodrop.to_numpy(), axis=0)
```

[4]:

```
def our_f(x):
    return torch.concat([x,x**2],dim=1)
# def our_f(x):
#     return x

lr = 0.1*(0.7**(np.floor((np.arange(10000)+2)/100)))
maxiter = 1000
tol=1e-3
reg=0
new_seed=133478

torch.manual_seed(new_seed)
# ### Seed Setting ### #
split_seed=int(1e9*unif.sample([1])[0])
miss_seed=int(1e9*unif.sample([1])[0])
learn_seed=int(1e9*unif.sample([1])[0])
new_seed = int(1e9*unif.sample([1])[0])
# ### Chossing missing functions ### #
signs = torch.tensor([-1., 1.])
signs = signs[torch.multinomial(torch.zeros((2))+1, num_samples=df_fin_nodrop.
 ↪shape[1],
                                replacement=True)]
missing_funcs = [create_standard_miss_func(
    m[j], std[j], -signs[j]) for j in range(df_fin_nodrop.shape[1])]
# Test our method

temp_our = full_pipeline(
    null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
    est_miss=True, lr=lr, alpha=0.2, delta=0.05, nlearn=50,
    f=our_f,
    reg=reg, maxiter=maxiter, tol=tol,
    split_seed=split_seed, miss_seed=miss_seed, learn_seed=learn_seed)

# Test true method
temp_true = full_pipeline(
    null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
    est_miss=False, lr=lr,f=our_f,
    alpha=0.2, delta=0.05, reg=reg, maxiter=maxiter, tol=tol,
    split_seed=split_seed, miss_seed=miss_seed)

# Test naive method
temp_naive = full_pipeline(
    null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
    dr_proc=kliep_multi_dim_naive_sep_wrap,
    est_miss=False, lr=lr,f=our_f,
    alpha=0.2, delta=0.05, reg=reg, maxiter=maxiter, tol=tol,
    split_seed=split_seed, miss_seed=miss_seed)
```

```
# Test no missing ability
temp_nomiss_norm = full_pipeline(
    null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
    dr_proc=kliep_miss_wrap, dat_val_fun=get_dat_vals_impute,
    est_miss=False, lr=lr,f=our_f,
    alpha=0.2, delta=0.05, reg=reg, maxiter=maxiter, tol=tol,
    split_seed=split_seed)

# Test no missing with NB approach
temp_nomiss_nb = full_pipeline(
    null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
    est_miss=False, lr=lr, f=our_f,
    alpha=0.2, delta=0.05, maxiter=maxiter,
    split_seed=split_seed)
```

[6]:

```
print(temp_nomiss_norm["power_res"])
print(temp_nomiss_nb["power_res"])
print(temp_our["power_res"])
print(temp_naive["power_res"])
print(temp_true["power_res"])
```

```
[[tensor(0.7537), tensor(0.1949)]]
[[tensor(0.6926), tensor(0.1949)]]
[[tensor(0.6899), tensor(0.1949)]]
[[tensor(0.5050), tensor(0.1949)]]
[[tensor(0.6879), tensor(0.1949)]]
```

1.1 Varying α

[11]:

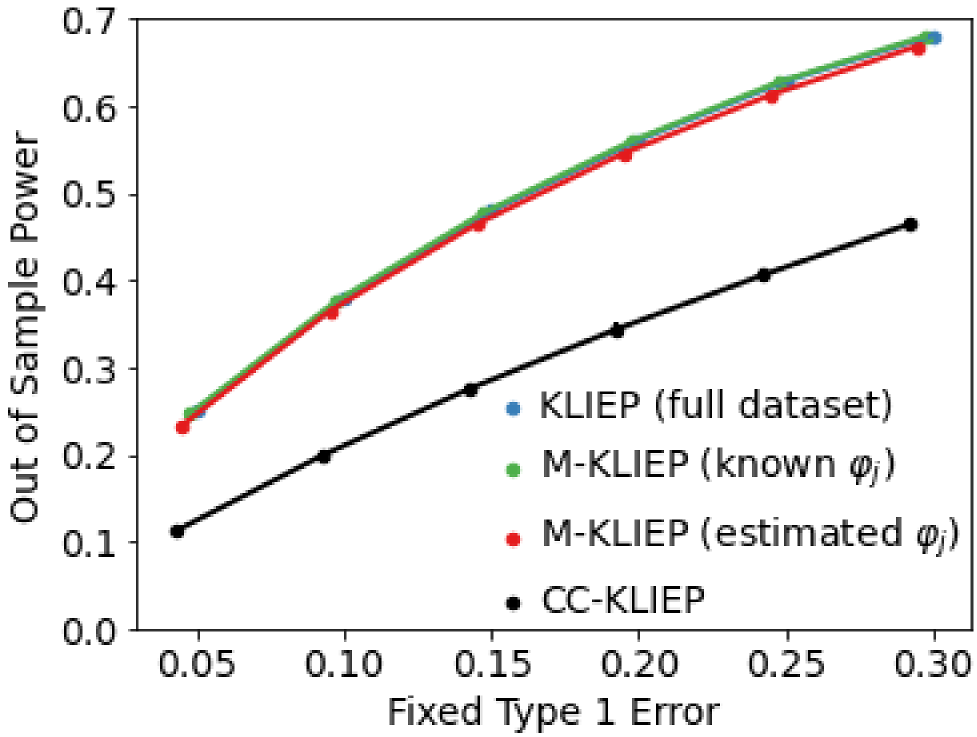
```
warnings.filterwarnings("ignore")
def our_f(x):
    return torch.concat([x,x**2],dim=1)
# def our_f(x):
#     return x

alphas = [0.3,0.25,0.2,0.15,0.1,0.05]
deltas = [0.05]*len(alphas)

lr = 0.1*(0.7**(np.floor((np.arange(10000)+2)/100)))
true_meth = []
best_meth = []
naive_meth = []
nomiss_nb_meth = []
nomiss_norm_meth = []
mice_meth = []
new_seed=12344
```

```
reg=0
tol=1e-3
maxiter=int(1e3)
mice_args = {"sample_posterior": False, "n_nearest_features": 9}
nsim=100
for i in range(nsim):
    torch.manual_seed(new_seed)
    # ### Seed Setting ### #
    split_seed=int(1e9*unif.sample([1])[0])
    miss_seed=int(1e9*unif.sample([1])[0])
    learn_seed=int(1e9*unif.sample([1])[0])
    new_seed = int(1e9*unif.sample([1])[0])
    # ### Chossing missing functions ### #
    signs = torch.tensor([-1., 1.])
    signs = signs[torch.multinomial(torch.zeros((2))+1,␣
 ↪num_samples=df_fin_nodrop.shape[1],
                                    replacement=True)]
    missing_funcs = [create_standard_miss_func(
        m[j], std[j], -signs[j]) for j in range(df_fin_nodrop.shape[1])]
    # Test our method
    temp_our = full_pipeline(
        null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
        est_miss=True, lr=lr, alpha=alphas, delta=deltas, nlearn=50,
        reg=reg, maxiter=maxiter, tol=tol,
        split_seed=split_seed, miss_seed=miss_seed, learn_seed=learn_seed)
    best_meth.append(temp_our["power_res"])

    # Test true method
    temp_true = full_pipeline(
        null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
        est_miss=False, lr=lr,
        alpha=alphas, delta=deltas, reg=reg, maxiter=maxiter, tol=tol,
        split_seed=split_seed, miss_seed=miss_seed)
    true_meth.append(temp_true["power_res"])

    # Test naive method
    temp_naive = full_pipeline(
        null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
        dr_proc=kliep_multi_dim_naive_sep_wrap,
        est_miss=False, lr=lr,
        alpha=alphas, delta=deltas, reg=reg, maxiter=maxiter, tol=tol,
        split_seed=split_seed, miss_seed=miss_seed)
    naive_meth.append(temp_naive["power_res"])

    # Test no missing ability
    temp_nomiss_norm = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        dr_proc=kliep_miss_wrap, dat_val_fun=get_dat_vals_impute,
        est_miss=False, lr=lr,
        alpha=alphas, delta=deltas, reg=reg, maxiter=maxiter, tol=tol,
        split_seed=split_seed)
    nomiss_norm_meth.append(temp_nomiss_norm["power_res"])

   # Test no missing with NB approach
    temp_nomiss_nb = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=maxiter,
        split_seed=split_seed)
    nomiss_nb_meth.append(temp_nomiss_nb["power_res"])

    # Test MICE
    temp_mice = full_pipeline(
        null_df, alt_df, missing_funcs=missing_funcs, n_altte=10000,␣
 ↪n_nulltr=15000,
        dr_proc=kliep_multi_dim_sep_imp_wrap, impute="MICE",
        mice_args=mice_args, est_miss=False, lr=lr, alpha=alphas,
        delta=deltas, maxiter=maxiter,
        split_seed=split_seed, miss_seed=miss_seed)
    mice_meth.append(temp_mice["power_res"])

    progress(int(100*(i+1)/nsim))

results = {"Learning Missingness Func": best_meth, "Naive": naive_meth,
 "Known Missingness Func": true_meth, "No Missing NB": nomiss_nb_meth,
 "No Missing Normal": nomiss_norm_meth,
 "MICE": mice_meth}

data = (results,{"alphas": alphas, "deltas": deltas})

with open('../results/real_world_results/
 ↪weatheraus_fullrand_'+str(nsim)+'sim_allmeth_varyalpha.pkl', 'wb') as handle:
    pickle.dump(data, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
[########################################]100%
```

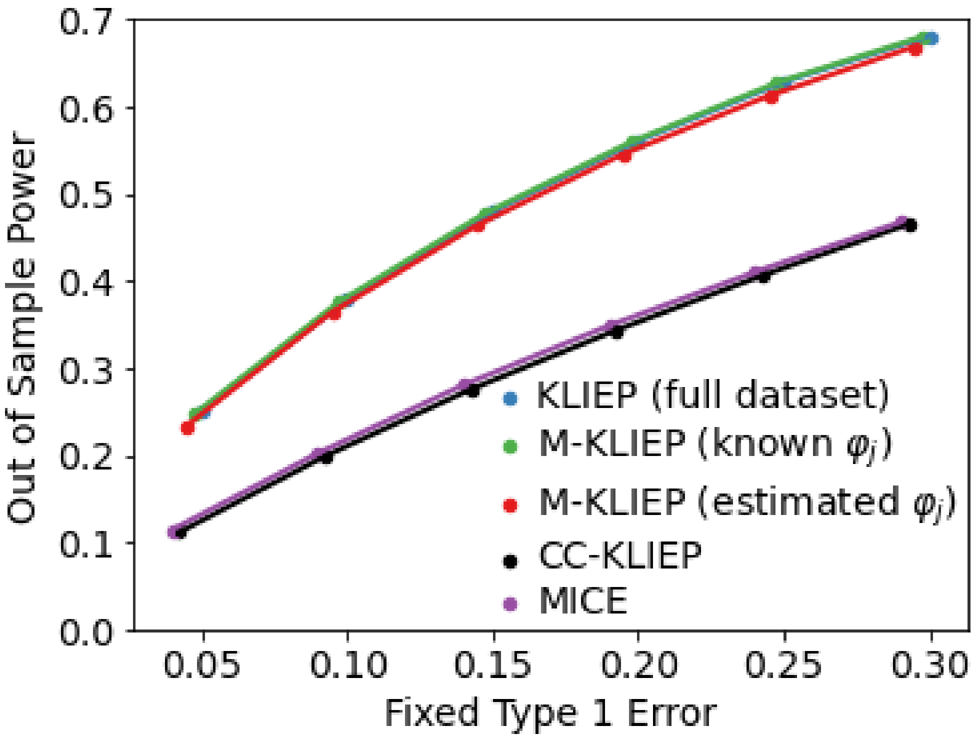
[27]:

```
with open('../results/real_world_results/
 ↪weatheraus_fullrand_100sim_allmeth_varyalpha.pkl', 'rb') as handle:
    results, params = pickle.load(handle)

names = {
    "No Missing NB": "KLIEP (full dataset)",
    "Known Missingness Func": r"M-KLIEP (known $\varphi_j$)",
    "Learning Missingness Func": r"M-KLIEP (estimated $\varphi_j$)",
    "Naive": "CC-KLIEP"
}
cs=[u'#377eb8',u'#4daf4a', u'#e41a1c',u'#000000', u'#984ea3']
fig, ax = plt.subplots(figsize=(0.7*8, 0.7*6))
for i, key in enumerate(names):
    x = np.array(params["alphas"])
    all_cis = get_ci(torch.tensor(results[key])[:, :, 0],
                     verbose=False)

    y1 = all_cis[1]
    y2 = all_cis[2]
    y = all_cis[0]
    error = y2-y

    diff=x[1]-x[0]
    x_jit=x+diff*0.05*i

    ax.scatter(x_jit, y, label=names[key], s=20, c=cs[i])
    ax.errorbar(x_jit, y, error, c=cs[i], linewidth=2)

ax.set(xlabel="Fixed Type 1 Error", ylabel="Out of Sample Power", ylim=(0,0.7))
ax.legend(handletextpad=-0.3,borderpad=0, borderaxespad=0.2,frameon=False)

plt.savefig("../plots/np_RWE_weather_varyalpha.pdf",bbox_inches="tight",␣
  ↪dpi=300)
```

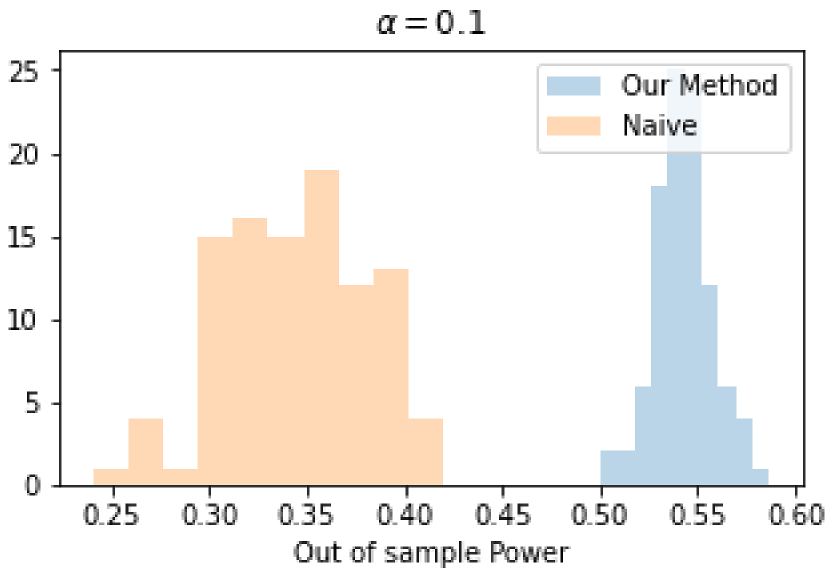


[24]:

```
with open('../results/real_world_results/
 ↪weatheraus_fullrand_100sim_allmeth_varyalpha.pkl', 'rb') as handle:
    results, params = pickle.load(handle)

names = {
    "No Missing NB": "KLIEP (full dataset)",
    "Known Missingness Func": r"M-KLIEP (known $\varphi_j$)",
    "Learning Missingness Func": r"M-KLIEP (estimated $\varphi_j$)",
    "Naive": "CC-KLIEP",
    "MICE": "MICE"
}
cs=[u'#377eb8',u'#4daf4a', u'#e41a1c',u'#000000', u'#984ea3']
fig, ax = plt.subplots(figsize=(0.7*8, 0.7*6))
for i, key in enumerate(names):
    x = np.array(params["alphas"])
    all_cis = get_ci(torch.tensor(results[key])[:, :, 0],
                     verbose=False)

    y1 = all_cis[1]
    y2 = all_cis[2]
    y = all_cis[0]
    error = y2-y

    diff=x[1]-x[0]
    x_jit=x+diff*0.05*i

    ax.scatter(x_jit, y, label=names[key], s=20, c=cs[i])
    ax.errorbar(x_jit, y, error, c=cs[i], linewidth=2)

ax.set(xlabel="Fixed Type 1 Error", ylabel="Out of Sample Power", ylim=(0,0.7))
ax.legend(handletextpad=-0.3,borderpad=0, borderaxespad=0.2,frameon=False,␣
 ↪labelspacing=0.2)

plt.savefig("../plots/np_RWE_weather_varyalpha_mice.pdf",bbox_inches="tight",␣
 ↪dpi=300)
```



[16]:
```
results_our = np.array(results["Learning Missingness Func"])[:,2,0]
results_naive = np.array(results["Naive"])[:,2,0]

fig, ax = plt.subplots(figsize=(5, 3))
ax.hist(results_our, alpha=0.3, label='Our Method')
ax.hist(results_naive, alpha=0.3, label='Naive')
ax.legend(loc='upper right')
ax.set(title=r"$\alpha=$"+str(params["alphas"][4]),xlabel="Out of sample Power")
plt.savefig("../plots/np_RWE_weather_power_dist.pdf",bbox_inches="tight")
```



2 Varying φ to vary proportion missing

We vary $a_{j,0}$ to vary the proportion of points missing. We now work out what $a_{j,0}$ ti choose to give our desired missing proportions

[13]:

```
def miss_prop_opt(loc,data, m, std, sign, target_prop):
    varphi = create_standard_miss_func(m,std, sign,loc)
    return (torch.nanmean(varphi(data))-target_prop)**2


df_tens = torch.tensor(
    (df_fin_nodrop).to_numpy().astype(np.float32))

locs = []
miss_props = np.arange(0.1,1,0.1)
for i, miss_prop in enumerate(miss_props):
    temp_locs = []
    for j in range(df_fin_nodrop.shape[1]):
        opt_plus = optimize.minimize_scalar(
            miss_prop_opt,args=(df_tens[:,j],m[j],std[j],1.,miss_prop))
        opt_minus = optimize.minimize_scalar(
            miss_prop_opt,args=(df_tens[:,j],m[j],std[j],-1.,miss_prop))

        temp_locs.append([opt_plus["x"], opt_minus["x"]])
    locs.append(temp_locs)
    progress(int(100*(i+1)/miss_props.shape[0]))
```

```
[########################################]100%
```

Now perform procedure for these various $a_{j,0}$

[14]:

```
warnings.filterwarnings("ignore")
new_seed = 12345
reg = 0
alphas = 0.1
deltas = 0.05
nsim = 30
miss_props = np.arange(0.1,1,0.1)
true_meth = {
    key: [[] for j in range(miss_props.shape[0])]
    for key in ["power_res", "prop_miss", "true_prop_miss"]
}
lr = 0.7**(np.floor((np.arange(1000))/100)+1)

best_meth = deepcopy(true_meth)
naive_meth = deepcopy(true_meth)
mice_meth = deepcopy(true_meth)

for j, miss_prop in enumerate(miss_props):
    for i in range(nsim):
        torch.manual_seed(new_seed)
        # ### Seed Setting ### #
        split_seed = int(1e9*unif.sample([1])[0])
        miss_seed = int(1e9*unif.sample([1])[0])
        learn_seed = int(1e9*unif.sample([1])[0])
        new_seed = int(1e9*unif.sample([1])[0])

        # ### Chossing missing functions ### #
        signs = torch.tensor([-1., 1.])
        signs = signs[torch.multinomial(torch.zeros((2))+1,␣
 ↪num_samples=df_fin_nodrop.shape[1],
                                        replacement=True)]

        sign_locs = ((signs+1)/2).int()
        missing_funcs = [create_standard_miss_func(
            m[l], std[l], -signs[l], shift=locs[j][l][sign_locs[l]])
            for l in range(df_fin_nodrop.shape[1])]

        # Test our method
        temp_our = full_pipeline(
            null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
            est_miss=True, lr=lr, alpha=alphas, delta=deltas, nlearn=10,␣
 ↪maxiter=1000,
            reg=reg,
            split_seed=split_seed, miss_seed=miss_seed, learn_seed=learn_seed)
        # Re-do with smaller LR if errors
        if any(torch.isnan(temp_our["dr"]["par"]).reshape(-1)):
            print(f"Fail on run {i}")
            print(
                f"Seeds are: Split - {split_seed}, Miss - {miss_seed}, Learn -␣
↪{learn_seed}")
            print(f"Signs are {signs}")
            temp_our = full_pipeline(
                null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
                est_miss=True, lr=0.1*lr, alpha=alphas, delta=deltas,␣
↪nlearn=10, maxiter=1000,
                reg=reg,
                split_seed=split_seed, miss_seed=miss_seed,␣
↪learn_seed=learn_seed)
            if any(torch.isnan(temp_our["dr"]["par"]).reshape(-1)):
                print(f"Still not Working on run {i}")
        best_meth["power_res"][j].append(temp_our["power_res"][0])
        best_meth["prop_miss"][j].append(temp_our["prop_miss"])
        best_meth["true_prop_miss"][j].append(miss_prop)

        # Test true method
        temp_true = full_pipeline(
            null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
            est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=1000,
            reg=reg,
            split_seed=split_seed, miss_seed=miss_seed)
        true_meth["power_res"][j].append(temp_true["power_res"][0])
        true_meth["prop_miss"][j].append(temp_true["prop_miss"])
        true_meth["true_prop_miss"][j].append(miss_prop)

        # Test naive method
        temp_naive = full_pipeline(
            null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
            dr_proc=kliep_multi_dim_naive_sep_wrap,
            est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=1000,
            reg=reg,
            split_seed=split_seed, miss_seed=miss_seed)
        naive_meth["power_res"][j].append(temp_naive["power_res"][0])
        naive_meth["prop_miss"][j].append(temp_naive["prop_miss"])
        naive_meth["true_prop_miss"][j].append(miss_prop)

        # Test MICE method
        temp_mice = full_pipeline(
            null_df, alt_df, missing_funcs, n_altte=10000, n_nulltr=15000,
            dr_proc=kliep_multi_dim_sep_imp_wrap, impute="Mice",␣
↪opt_type="scalar",
            est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=1000,
            reg=reg, split_seed=split_seed, miss_seed=miss_seed)
        mice_meth["power_res"][j].append(temp_mice["power_res"][0])
        mice_meth["prop_miss"][j].append(temp_mice["prop_miss"])
        mice_meth["true_prop_miss"][j].append(miss_prop)

        progress(int(100*(j*nsim+(i+1))/(nsim*miss_props.shape[0])))

results = {
    "Learning Missingness Func": best_meth,
    "Naive": naive_meth, "Known Missingness Func": true_meth,
    "MICE": mice_meth
}

data = (results, {"alphas": alphas, "deltas": deltas})
with open('../results/real_world_results/
 ↪weatheraus_fullrand_'+str(nsim)+'sim_allmeth_varymissfixed.pkl', 'wb') as␣
 ↪handle:
    pickle.dump(data, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
[########################################]100%
```

Finally perform the procedure for the case of no missingness

[17]:

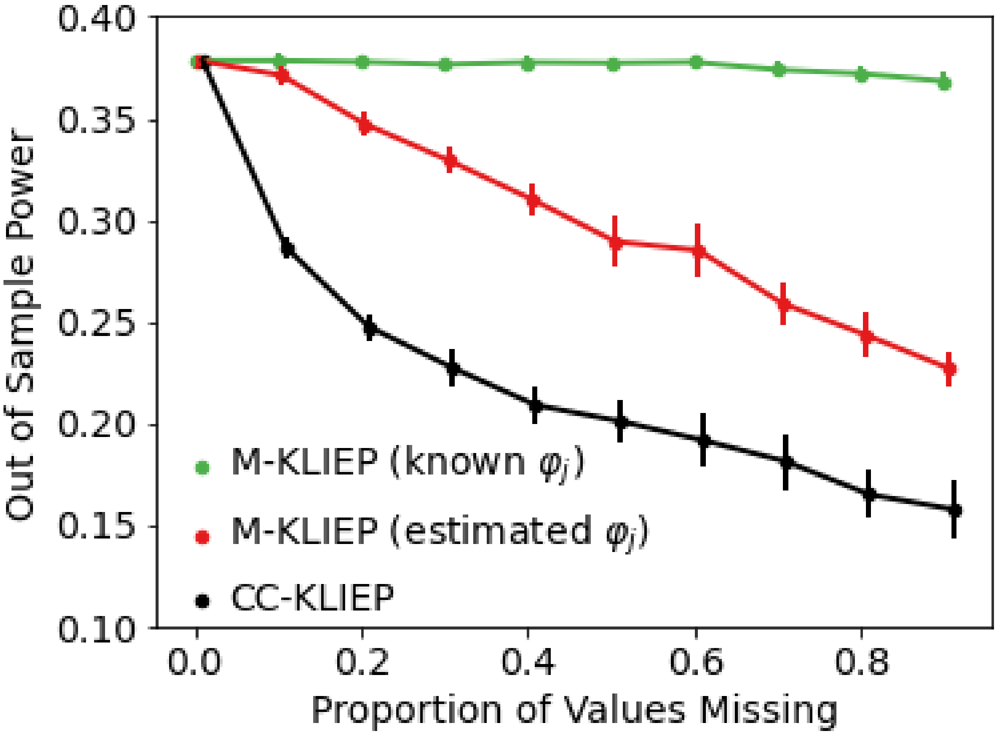
```
def our_f(x):
    return torch.concat([x,x**2],dim=1)
# def our_f(x):
#     return x
new_seed = 12345
reg = 0
alphas = 0.1
deltas = 0.05
nsim = 30
true_meth = {
    key: [[]]
    for key in ["power_res", "prop_miss", "true_prop_miss"]
}
lr = 0.7**(np.floor((np.arange(1000))/100)+1)

best_meth = deepcopy(true_meth)
naive_meth = deepcopy(true_meth)
mice_meth = deepcopy(true_meth)
miss_prop=0
for i in range(nsim):
    torch.manual_seed(new_seed)
    # ### Seed Setting ### #
    split_seed = int(1e9*unif.sample([1])[0])
    miss_seed = int(1e9*unif.sample([1])[0])
    learn_seed = int(1e9*unif.sample([1])[0])
    new_seed = int(1e9*unif.sample([1])[0])

    # ### Chossing missing functions ### #
    signs = torch.tensor([-1., 1.])
    signs = signs[torch.multinomial(torch.zeros((2))+1,␣
↪num_samples=df_fin_nodrop.shape[1],
                                    replacement=True)]

    sign_locs = ((signs+1)/2).int()
    missing_funcs = [create_standard_miss_func(
        m[l], std[l], -signs[l], shift=locs[j][l][sign_locs[l]])
        for l in range(df_fin_nodrop.shape[1])]

    # Test our method
    temp_our = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        est_miss=False, lr=lr, alpha=alphas, delta=deltas, nlearn=10,␣
↪maxiter=1000,
        reg=reg,
        split_seed=split_seed, miss_seed=miss_seed, learn_seed=learn_seed)
    # Re-do with smaller LR if errors
    if any(torch.isnan(temp_our["dr"]["par"]).reshape(-1)):
        print(f"Fail on run {i}")
        print(
            f"Seeds are: Split - {split_seed}, Miss - {miss_seed}, Learn -␣
↪{learn_seed}")
        print(f"Signs are {signs}")
        temp_our = full_pipeline(
            null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
            est_miss=False, lr=0.1*lr, alpha=alphas, delta=deltas, nlearn=10,␣
↪maxiter=1000,
            reg=reg,
            split_seed=split_seed, miss_seed=miss_seed, learn_seed=learn_seed)
        if any(torch.isnan(temp_our["dr"]["par"]).reshape(-1)):
            print(f"Still not Working on run {i}")
    best_meth["power_res"][0].append(temp_our["power_res"][0])
    best_meth["prop_miss"][0].append(temp_our["prop_miss"])
    best_meth["true_prop_miss"][0].append(miss_prop)

    # Test true method
    temp_true = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=1000,
        reg=reg,
        split_seed=split_seed, miss_seed=miss_seed)
    true_meth["power_res"][0].append(temp_true["power_res"][0])
    true_meth["prop_miss"][0].append(temp_true["prop_miss"])
    true_meth["true_prop_miss"][0].append(miss_prop)

    # Test naive method
    temp_naive = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        dr_proc=kliep_multi_dim_naive_sep_wrap,
        est_miss=False, lr=lr, alpha=alphas, delta=deltas, maxiter=1000,
        reg=reg,
        split_seed=split_seed, miss_seed=miss_seed)
    naive_meth["power_res"][0].append(temp_naive["power_res"][0])
    naive_meth["prop_miss"][0].append(temp_naive["prop_miss"])
    naive_meth["true_prop_miss"][0].append(miss_prop)

    # Test MICE method
    temp_mice = full_pipeline(
        null_df, alt_df, None, n_altte=10000, n_nulltr=15000,
        dr_proc=kliep_multi_dim_sep_imp_wrap, impute="MICE", opt_type="Scalar",
        est_miss=False, lr=lr, alpha=alphas,
        delta=deltas, maxiter=1000,
        split_seed=split_seed, miss_seed=miss_seed)
    mice_meth["power_res"][0].append(temp_mice["power_res"][0])
    mice_meth["prop_miss"][0].append(0)
    mice_meth["true_prop_miss"][0].append(miss_prop)

    progress(int(100*(i+1)/nsim))

results = {
    "Learning Missingness Func": best_meth,
    "Naive": naive_meth, "Known Missingness Func": true_meth,
    "MICE": mice_meth
}

data = (results, {"alphas": alphas, "deltas": deltas})
with open('../results/real_world_results/
 ↪weatheraus_fullrand_'+str(nsim)+'sim_allmeth_nomiss.pkl', 'wb') as handle:
    pickle.dump(data, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

```
[########################################]100%
```

[26]:

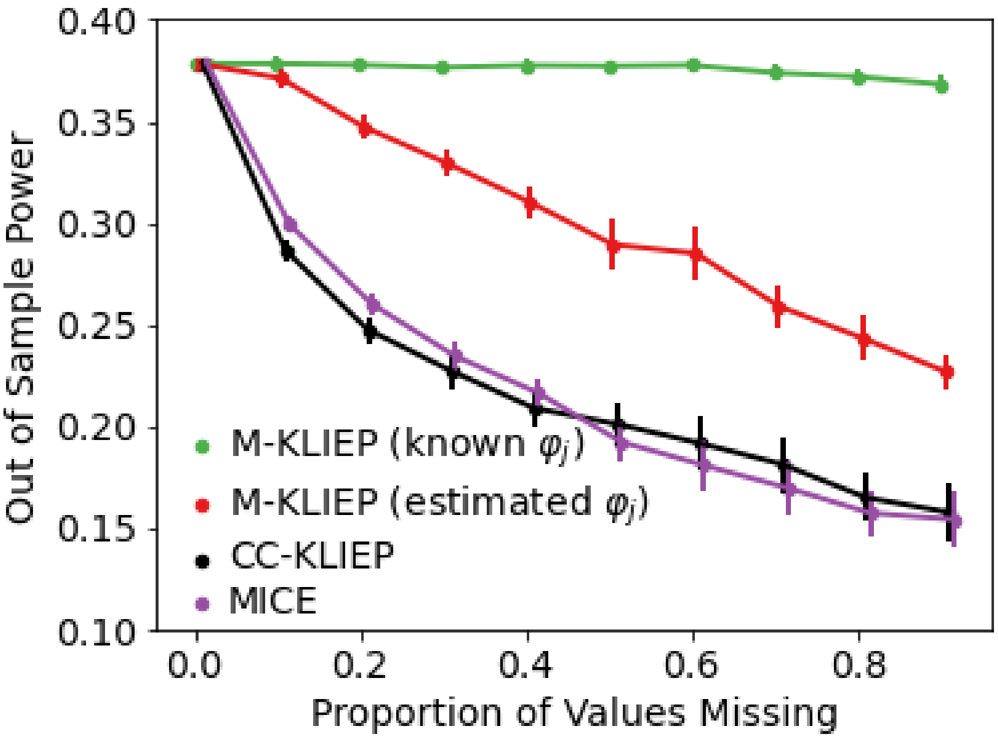
```
with open('../results/real_world_results/
 ↪weatheraus_fullrand_30sim_allmeth_nomiss.pkl', 'rb') as handle:
    no_miss_results, params = pickle.load(handle)

with open('../results/real_world_results/
 ↪weatheraus_fullrand_30sim_allmeth_varymissfixed.pkl', 'rb') as handle:
    results, params = pickle.load(handle)
```

```
names = {
    "Known Missingness Func": r"M-KLIEP (known $\varphi_j$)",
    "Learning Missingness Func": r"M-KLIEP (estimated $\varphi_j$) ",
    "Naive": "CC-KLIEP",
}

cs=[u'#377eb8',u'#4daf4a', u'#e41a1c',u'#000000', u'#984ea3']
fig, ax = plt.subplots(figsize=(0.7*8, 0.7*6))

for i, key in enumerate(names):
    x = torch.mean(torch.
 ↪tensor(no_miss_results[key]["prop_miss"]+results[key]["prop_miss"]),dim=1)
    all_cis = get_ci(torch.
 ↪tensor(no_miss_results[key]["power_res"]+results[key]["power_res"])[:, :, 0].
 ↪T,
                     verbose=False)

    y1 = all_cis[1]
    y2 = all_cis[2]
    y = all_cis[0]
    error = y2-y

    diff = x[1]-x[0]

    x_jit = x+diff*0.05*i

    ax.scatter(x_jit, y, label=names[key], s=20, c=cs[i+1])
    ax.errorbar(x_jit, y, error,c=cs[i+1], linewidth=2)

ax.set(xlabel="Proportion of Values Missing", ylabel="Out of Sample Power",␣
 ↪ylim=(0.1, 0.4))
ax.legend(handletextpad=-0.3,borderpad=0, borderaxespad=0.2,frameon=False)
plt.savefig("../plots/np_RWE_weather_varymissfixed.pdf",
            bbox_inches="tight", dpi=300)
```



[29]:
```
with open('../results/real_world_results/
 ↪weatheraus_fullrand_30sim_allmeth_nomiss.pkl', 'rb') as handle:
    no_miss_results, params = pickle.load(handle)

with open('../results/real_world_results/
 ↪weatheraus_fullrand_30sim_allmeth_varymissfixed.pkl', 'rb') as handle:
    results, params = pickle.load(handle)

names = {
    "Known Missingness Func": r"M-KLIEP (known $\varphi_j$)",
    "Learning Missingness Func": r"M-KLIEP (estimated $\varphi_j$) ",
    "Naive": "CC-KLIEP",
    "MICE": "MICE",
}

cs=[u'#377eb8',u'#4daf4a', u'#e41a1c',u'#000000', u'#984ea3']
fig, ax = plt.subplots(figsize=(0.7*8, 0.7*6))

for i, key in enumerate(names):
    x = torch.mean(torch.
 ↪tensor(no_miss_results[key]["prop_miss"]+results[key]["prop_miss"]),dim=1)
```

```
    all_cis = get_ci(torch.
 ↪tensor(no_miss_results[key]["power_res"]+results[key]["power_res"])[:, :, 0].
 ↪T,
                     verbose=False)

    y1 = all_cis[1]
    y2 = all_cis[2]
    y = all_cis[0]
    error = y2-y

    diff = x[1]-x[0]

    x_jit = x+diff*0.05*i

    ax.scatter(x_jit, y, label=names[key], s=20, c=cs[i+1])
    ax.errorbar(x_jit, y, error,c=cs[i+1], linewidth=2)

ax.set(xlabel="Proportion of Values Missing", ylabel="Out of Sample Power",␣
 ↪ylim=(0.1, 0.4))
ax.legend(handletextpad=-0.3,borderpad=0, borderaxespad=0.2,frameon=False,␣
 ↪labelspacing=0.2)
plt.savefig("../plots/np_RWE_weather_varymissfixed_mice.pdf",
            bbox_inches="tight", dpi=300)
```


Out of Sample Power
M-KLIEP (known φ_j)
M-KLIEP (estimated φ_j)
CC-KLIEP
MICE
Proportion of Values Missing