Script-1:

import math

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

from sklearn.model\_selection import KFold

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

%matplotlib inline

sns.set(font\_scale=1.1, style="white")

sns\_cmap = sns.color\_palette().as\_hex()

cpu\_col = 'Resource Req. [e.g., CPU time]'

thr\_col = 'Traffic Load [e.g., Mbit/s]'

def gen\_benchmark(cpu, coeff1=1, coeff2=1):

    return coeff1 \* math.log2(1 + coeff2 \* cpu)

def synthetic\_benchmark():

    cpu\_list = np.arange(0.1, 1, .01)

    data = []

    for cpu in cpu\_list:

        data.append([cpu, gen\_benchmark(cpu, coeff2=100)])

    return pd.DataFrame(data, columns=[cpu\_col, thr\_col])

def inverse(thru, coeff2=1):

    return (1/coeff2) \* (2\*\*thru - 1)

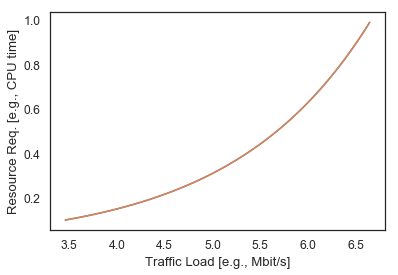
df = synthetic\_benchmark()

# sns.lineplot(df[cpu\_col], df[thr\_col])

sns.lineplot(df[thr\_col], df[cpu\_col])

cpu = [inverse(thr, coeff2=100) for thr in df[thr\_col]]

sns.lineplot(df[thr\_col], cpu)



X = df[[thr\_col]]

y = df[cpu\_col]

lin\_reg = LinearRegression()

lin\_reg.fit(X, y)

lin\_col = 'Linear'

df[lin\_col] = lin\_reg.predict(X)

joblib.dump(lin\_reg, 'ml\_models/synth\_data/linear.joblib')

boost = GradientBoostingRegressor()

boost.fit(X, y)

boost\_col = 'Boosting'

df[boost\_col] = boost.predict(X)

joblib.dump(boost, 'ml\_models/synth\_data/boosting.joblib')

# nice fit

mean\_squared\_error(df[cpu\_col], df[boost\_col])

boost.predict([[10.0]])

inverse(10, coeff2=100)

# plotting for cpu 0-100

def plot\_cpu\_0\_100():

    fig, ax = plt.subplots()

    fixed = 80

    # lines

    plt.plot(df[thr\_col], df[cpu\_col], label='True', color='black', linewidth=2)

    plt.plot(df[thr\_col], df[lin\_col], color='blue', linewidth=2)

    plt.axhline(y=fixed, label='Fixed', color='green', linewidth=2)

    # fill in between

    ax.fill\_between(df[thr\_col], df[cpu\_col], fixed, where=df[cpu\_col]<fixed, facecolor='white', edgecolor='grey', hatch='//')

    ax.fill\_between(df[thr\_col], df[lin\_col], df[cpu\_col], where=df[lin\_col]<df[cpu\_col], facecolor='lightgrey', edgecolor='grey', hatch='\\')

    # # text

    ax.text(10.5, 60, 'Over-allocation', bbox={'facecolor': 'white'})

    ax.annotate('Under-allocation', xy=(10.2, 5), xytext=(11, 5), arrowprops={'facecolor': 'black'}, verticalalignment='center')

    plt.xlabel(thr\_col)

    plt.ylabel(cpu\_col)

    plt.xlim(10)

    #plt.ylim(0, 1)

    plt.legend()

    fig.savefig('plots/example\_alloc.pdf', bbox\_inches='tight')

# plot\_cpu\_0\_100()

# plotting for cpu 0-1

fig, ax = plt.subplots()

fixed = 0.8

# lines

plt.plot(df[thr\_col], df[cpu\_col], label='True', color='black', linewidth=2)

plt.plot(df[thr\_col], df[lin\_col], color=sns\_cmap[0], linewidth=3, linestyle="dashed")

# plt.plot(df[thr\_col], df[boost\_col], color='red', linewidth=2)

plt.axhline(y=fixed, label='Fixed', color=sns\_cmap[3], linewidth=3, linestyle="dotted")

# fill in between

ax.fill\_between(df[thr\_col], df[cpu\_col], fixed, where=df[cpu\_col]<fixed, facecolor='white', edgecolor='lightgrey', hatch='//')

ax.fill\_between(df[thr\_col], df[lin\_col], df[cpu\_col], where=df[lin\_col]<df[cpu\_col], facecolor='lightgrey')

# # text

ax.text(4, 0.5, 'Over-Allocation', bbox={'facecolor': 'white'})

ax.annotate('Under-Allocation', xy=(3.7, 0.05), xytext=(4.5, 0.1), arrowprops={'facecolor': 'black'}, verticalalignment='center')

plt.xlabel(thr\_col)

plt.ylabel(cpu\_col)

plt.xlim(3.5)

plt.ylim(0, 1)

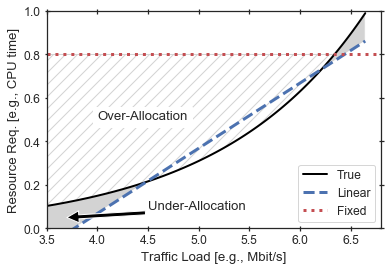
plt.tick\_params(axis='both', direction='inout', length=5, bottom=True, left=True, right=True, top=True)

plt.legend()

fig.savefig('plots/example\_alloc.pdf', bbox\_inches='tight')

# for presentation as png

fig.savefig('plots/example\_alloc\_lin.png', dpi=150, bbox\_inches='tight')



# 2nd: plot fixed resource alloc at 80%

fig, ax = plt.subplots()

fixed=0.8

# lines

plt.plot(df[thr\_col], df[cpu\_col], label='True', color='black', linewidth=2)

plt.axhline(y=fixed, label='Fixed', color='green', linewidth=2)

# fill in between

ax.fill\_between(df[thr\_col], df[cpu\_col], fixed, where=df[cpu\_col]<fixed, facecolor='white', edgecolor='lightgrey', hatch='//')

ax.fill\_between(df[thr\_col], df[cpu\_col], fixed, where=df[cpu\_col]>fixed, facecolor='lightgrey')

# # text

ax.text(4, 0.5, 'Over-allocation', bbox={'facecolor': 'white'})

ax.annotate('Under-allocation', xy=(6.5, 0.85), xytext=(6.25, 0.4), arrowprops={'facecolor': 'black'}, verticalalignment='center', horizontalalignment='center')

plt.xlabel(thr\_col)

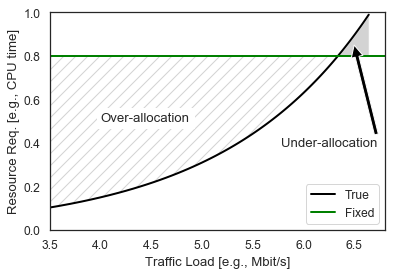
plt.ylabel(cpu\_col)

plt.xlim(3.5)

plt.ylim(0, 1)

plt.legend()

fig.savefig('plots/example\_alloc\_fixed.png', dpi=150, bbox\_inches='tight')



Script-2:

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

import os

import math

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

from sklearn.base import BaseEstimator

from sklearn.model\_selection import KFold, train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler, PolynomialFeatures

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

%matplotlib inline

sns.set(font\_scale=3, style="white")

class FixedModel(BaseEstimator):

    def \_\_init\_\_(self, fixed\_value):

        self.fixed\_value = fixed\_value

    def fit(self, X, y):

        return self

    def predict(self, X):

        n\_samples = X.shape[0]

        return [self.fixed\_value for \_ in range(n\_samples)]

# define column names

cpu\_col = 'Resource req. [e.g., CPU]'

thr\_col = 'Traffic load [e.g., Mbit/s]'

# generate synthetic data

def gen\_benchmark(cpu, coeff1=1, coeff2=1):

    return coeff1 \* math.log2(1 + coeff2 \* cpu)

def synthetic\_benchmark():

    cpu\_list = np.arange(0.1, 1, .01)

    data = []

    for cpu in cpu\_list:

        data.append([cpu, gen\_benchmark(cpu, coeff2=100)])

    return pd.DataFrame(data, columns=[cpu\_col, thr\_col])

# inverse function to calculate cpu given a throughput

def inverse(thru, coeff2=1):

    return (1/coeff2) \* (2\*\*thru - 1)

# prepare data

df = synthetic\_benchmark()

X = df[[thr\_col]]

y = df[cpu\_col]

def cross\_validation\_rmse(model, X, y, k=5, save\_model=False):

    scores = cross\_val\_score(model, X, y, scoring="neg\_mean\_squared\_error", cv=k)

    rmse = np.sqrt(-scores)

    name = type(model).\_\_name\_\_

    print(f"CV RMSE of {name}: {rmse.mean()} (+/-{rmse.std()})")

    if save\_model:

        model.fit(X, y)

        joblib.dump(model, f'ml\_models/synth\_data/{name}.joblib')

    return rmse

def barplot\_rmse(scores, labels, data\_name):

    assert len(scores) == len(labels)

    rmse\_mean = [s.mean() for s in scores]

    rmse\_std = [s.std() for s in scores]

    x = np.arange(len(labels))

    fig, ax = plt.subplots(figsize = (8, 6))

    plt.barh(x, rmse\_mean, color='grey', xerr=rmse\_std, capsize=5)

    ax.set\_xlabel('RMSE')

    ax.set\_yticks(x)

    ax.set\_yticklabels(labels)

    fig.savefig(f'plots/{data\_name}\_rmse.pdf', bbox\_inches='tight')

models = [LinearRegression(), Ridge(), SVR(), RandomForestRegressor(),

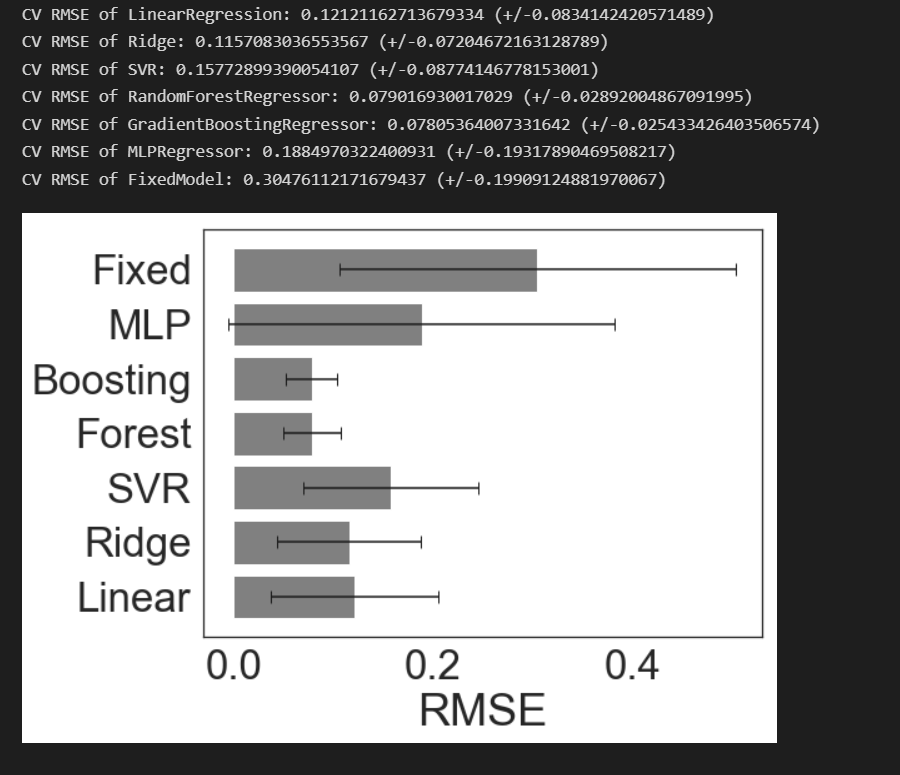
          GradientBoostingRegressor(), MLPRegressor(max\_iter=1500),

          FixedModel(fixed\_value=0.8)]

labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

rmse = [cross\_validation\_rmse(model, X, y) for model in models]

barplot\_rmse(rmse, labels, 'synth\_default')



scaler = MinMaxScaler()

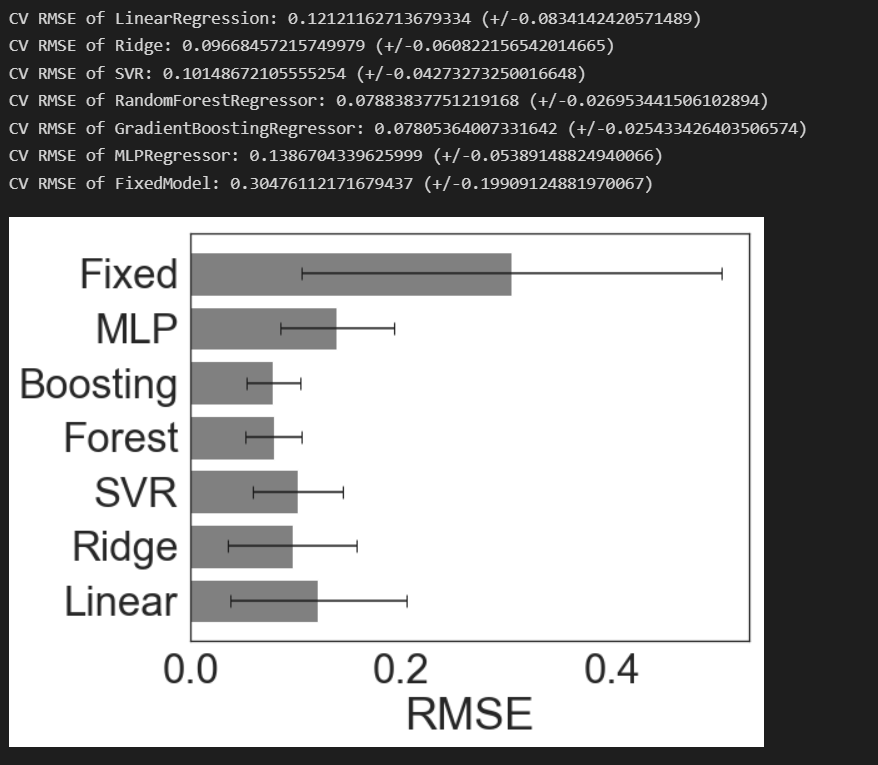
X\_scaled = scaler.fit\_transform(X)

os.makedirs(f'ml\_models/synth\_data', exist\_ok=True)

joblib.dump(scaler, f'ml\_models/synth\_data/scaler.joblib')

rmse = [cross\_validation\_rmse(model, X\_scaled, y) for model in models]

barplot\_rmse(rmse, labels, 'synth\_scaled')



def tune\_hyperparams(model, X, y, params):

    grid\_search = GridSearchCV(model, params, cv=5, scoring="neg\_mean\_squared\_error")

    grid\_search.fit(X, y)

    return grid\_search.best\_estimator\_

# hyperparam tuning

params\_ridge = {'alpha': [0.1, 1, 10]}

params\_svr = {'kernel': ['linear', 'poly', 'rbf'], 'C': [1, 10, 100],

              'epsilon': [0.001, 0.01, 0.1]}

params\_forest = {'n\_estimators': [10, 100, 200]}

params\_boosting = {'learning\_rate': [0.01, 0.1, 0.3], 'n\_estimators': [10, 100, 200]}

params\_mlp = {'hidden\_layer\_sizes': [(64,), (128,), (256)], 'alpha': [0.001, 0.0001, 0.00001],

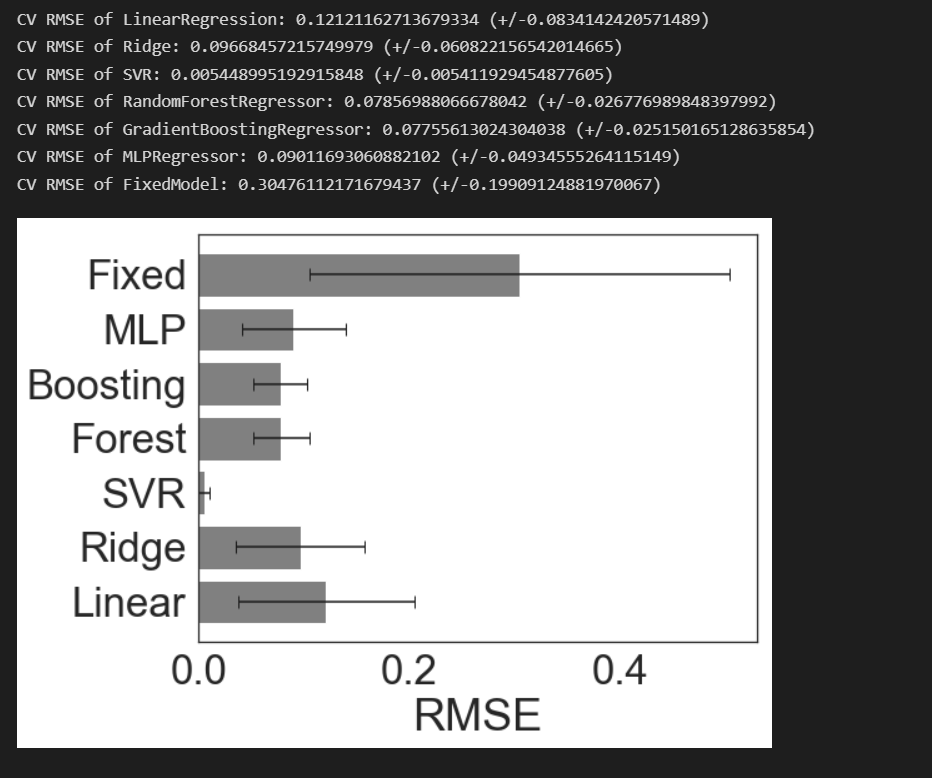
              'learning\_rate\_init': [0.01, 0.001, 0.0001]}

params = [{}, params\_ridge, params\_svr, params\_forest, params\_boosting, params\_mlp, {}]

models\_tuned = [tune\_hyperparams(models[i], X\_scaled, y, params[i]) for i in range(len(models))]

rmse\_tuned = [cross\_validation\_rmse(model, X\_scaled, y, save\_model=True) for model in models\_tuned]

barplot\_rmse(rmse\_tuned, labels, 'synth\_tuned')



# plot

sns.set(font\_scale=1, style="white")

def plot\_predictions(models, labels, X, y):

    assert len(models) == len(labels)

    fig, ax = plt.subplots()

    plt.plot(X[thr\_col], y, label='True', color='black', linewidth=2)

    for i in range(len(models)):

        models[i].fit(X, y)

        plt.plot(X[thr\_col], models[i].predict(X), label=labels[i])

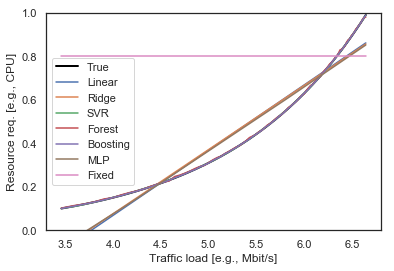
    plt.xlabel(thr\_col)

    plt.ylabel(cpu\_col)

    plt.ylim(0, 1)

    plt.legend()

plot\_predictions(models\_tuned, labels, X, y)



Script-3:

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

import time

from sklearn.base import BaseEstimator

from sklearn.model\_selection import KFold, train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler, PolynomialFeatures

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

sns.set(font\_scale=1.1, style='white')

class FixedModel(BaseEstimator):

    def \_\_init\_\_(self, fixed\_value):

        self.fixed\_value = fixed\_value

    def fit(self, X, y):

        return self

    def predict(self, X):

        n\_samples = X.shape[0]

        return [self.fixed\_value for \_ in range(n\_samples)]

def select\_and\_rename(df, mapping):

    dff = df[list(mapping.keys())]

    # rename

    for k, v in mapping.items():

        dff.rename(columns={k: v}, inplace=True)

    return dff

def replaceSize(df):

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -i http://20.0.0.254:8888/", "small")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k http://20.0.0.254:8888/bunny.mp4", "big")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -i -X 20.0.0.254:3128 http://40.0.0.254:80/", "small")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -X 20.0.0.254:3128 http://40.0.0.254:80/bunny.mp4", "big")

    return df

# Load data from path

web1 = pd.read\_csv("vnf\_data/csv\_experiments\_WEB1.csv")

web2 = pd.read\_csv("vnf\_data/csv\_experiments\_WEB2.csv")

web3 = pd.read\_csv("vnf\_data/csv\_experiments\_WEB3.csv")

# do processing, renaming and selection

mapping = {

    "param\_\_func\_\_mp.input\_\_cmd\_start": "size",

    "metric\_\_mp.input.vdu01.0\_\_ab\_transfer\_rate\_kbyte\_per\_second": "Max. throughput [kB/s]",

}

mapping01 = mapping.copy()

mapping01["param\_\_func\_\_de.upb.lb-nginx.0.1\_\_cpu\_bw"] = "CPU"

mapping01["param\_\_func\_\_de.upb.lb-nginx.0.1\_\_mem\_max"] = "Memory"

mapping02 = mapping.copy()

mapping02["param\_\_func\_\_de.upb.lb-haproxy.0.1\_\_cpu\_bw"] = "CPU"

mapping02["param\_\_func\_\_de.upb.lb-haproxy.0.1\_\_mem\_max"] = "Memory"

mapping03 = mapping.copy()

mapping03["param\_\_func\_\_de.upb.px-squid.0.1\_\_cpu\_bw"] = "CPU"

mapping03["param\_\_func\_\_de.upb.px-squid.0.1\_\_mem\_max"] = "Memory"

web1 = select\_and\_rename(web1, mapping01)

web2 = select\_and\_rename(web2, mapping02)

web3 = select\_and\_rename(web3, mapping03)

web1 = replaceSize(web1)

web2 = replaceSize(web2)

web3 = replaceSize(web3)

mem = 128

web1\_small = web1.loc[(web1["size"] == "small") & (web1["Memory"] == mem)]

web1\_small = web1\_small[["Max. throughput [kB/s]", "CPU"]]

web1\_big = web1.loc[(web1["size"] == "big")  & (web1["Memory"] == mem)]

web1\_big = web1\_big[["Max. throughput [kB/s]", "CPU"]]

web2\_small = web2.loc[(web2["size"] == "small")  & (web2["Memory"] == mem)]

web2\_small = web2\_small[["Max. throughput [kB/s]", "CPU"]]

web2\_big = web2.loc[(web2["size"] == "big")  & (web2["Memory"] == mem)]

web2\_big = web2\_big[["Max. throughput [kB/s]", "CPU"]]

web3\_small = web3.loc[(web3["size"] == "small")  & (web3["Memory"] == mem)]

web3\_small = web3\_small[["Max. throughput [kB/s]", "CPU"]]

web3\_big = web3.loc[(web3["size"] == "big")  & (web3["Memory"] == mem)]

web3\_big = web3\_big[["Max. throughput [kB/s]", "CPU"]]

num\_measures = 20

measures = [0 for \_ in range(num\_measures)]

web1\_small = web1\_small.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web1\_big = web1\_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web2\_small = web2\_small.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web2\_big = web2\_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web3\_small = web3\_small.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web3\_big = web3\_big.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

nginx = web1\_small

haproxy = web2\_small

squid = web3\_small

def plot\_vnf\_data():

    sns.set(font\_scale=1.1, style='white')

    fig, ax = plt.subplots()

    plt.scatter(nginx['Max. throughput [kB/s]'], nginx['CPU'], label='Nginx', marker='.')

    plt.scatter(haproxy['Max. throughput [kB/s]'], haproxy['CPU'], label='HAProxy', marker='+')

    plt.scatter(squid['Max. throughput [kB/s]'], squid['CPU'], label='Squid', marker='x')

    # labels

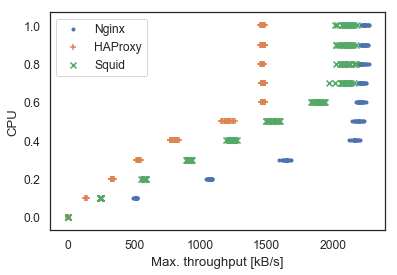
    ax.set\_xlabel('Max. throughput [kB/s]')

    ax.set\_ylabel('CPU')

    plt.legend()

    fig.savefig(f'plots/web\_vnf\_data.pdf', bbox\_inches='tight')

plot\_vnf\_data()



def cross\_validation\_rmse(model, X, y, vnf\_name, k=5, save\_model=False):

    scores = cross\_val\_score(model, X, y, scoring="neg\_mean\_squared\_error", cv=k)

    rmse = np.sqrt(-scores)

    name = type(model).\_\_name\_\_

    print(f"CV RMSE of {name}: {rmse.mean()} (+/-{rmse.std()})")

    if save\_model:

        model.fit(X, y)

        joblib.dump(model, f'ml\_models/{vnf\_name}/{name}.joblib')

    return rmse

def tune\_hyperparams(model, X, y, params):

    grid\_search = GridSearchCV(model, params, cv=5, scoring="neg\_mean\_squared\_error")

    grid\_search.fit(X, y)

    return grid\_search.best\_estimator\_

def barplot\_rmse(scores, labels, data\_name):

    sns.set(font\_scale=1.1, style='white')

    assert len(scores) == len(labels)

    # preparation

    rmse\_mean = [s.mean() for s in scores]

    rmse\_std = [s.std() for s in scores]

    x = np.arange(len(labels))

    # plot

    fig, ax = plt.subplots() #figsize = (8, 6))

    plt.barh(x, rmse\_mean, color='grey', xerr=rmse\_std, capsize=5)

    # labels

    ax.set\_xlabel('RMSE')

    ax.set\_yticks(x)

    ax.set\_yticklabels(labels)

    fig.savefig(f'plots/{data\_name}\_rmse.pdf', bbox\_inches='tight')

def train\_eval\_models(X, y, vnf\_name, tune\_params=False):

    # prepare models

    labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

    models = [LinearRegression(), Ridge(), SVR(), RandomForestRegressor(),

              GradientBoostingRegressor(), MLPRegressor(max\_iter=1500),

              FixedModel(fixed\_value=0.8)]

    # params for tuning

    params\_ridge = {'alpha': [0.1, 1, 10]}

    params\_svr = {'kernel': ['linear', 'poly', 'rbf'], 'C': [1, 10, 100],

                  'epsilon': [0.001, 0.01, 0.1]}

    params\_forest = {'n\_estimators': [10, 100, 200]}

    params\_boosting = {'learning\_rate': [0.01, 0.1, 0.3], 'n\_estimators': [10, 100, 200]}

    params\_mlp = {'hidden\_layer\_sizes': [(64,), (128,), (256)], 'alpha': [0.001, 0.0001, 0.00001],

                  'learning\_rate\_init': [0.01, 0.001, 0.0001]}

    params = [{}, params\_ridge, params\_svr, params\_forest, params\_boosting, params\_mlp, {}]

    # tune, train, eval

    if tune\_params:

        models = [tune\_hyperparams(models[i], X, y, params[i]) for i in range(len(models))]

    rmse = [cross\_validation\_rmse(model, X, y, vnf\_name, save\_model=True) for model in models]

    # plot

    if tune\_params:

        barplot\_rmse(rmse, labels, f'{vnf\_name}\_tuned')

    else:

        barplot\_rmse(rmse, labels, f'{vnf\_name}\_default')

    return models

def prepare\_data(data, vnf\_name):

    X = data[['Max. throughput [kB/s]']]

    y = data['CPU']

    X = X.fillna(X.median())

    scaler = MinMaxScaler()

    scaler.fit(X)

    os.makedirs(f'ml\_models/{vnf\_name}', exist\_ok=True)

    joblib.dump(scaler, f'ml\_models/{vnf\_name}/scaler.joblib')

    return X, y, scaler

def predict\_plot\_all(models, scaler, X, y, vnf\_name):

    sns.set(font\_scale=1.1, style='white')

    models = [models[0], models[2], models[3], models[6]]

    labels = ['Linear', 'SVR', 'Boosting', 'Fixed']

    markers = ['x', 'v', '^', '+']

    colors = ['blue', 'orange', 'red', 'green']

    fig, ax = plt.subplots()

    plt.scatter(X, y, label='True', marker='o', color='black', s=50)

    X = scaler.transform(X)

    times = []

    for i, model in enumerate(models):

        name = type(model).\_\_name\_\_

        model.fit(X, y)

        os.makedirs(f'ml\_models/{vnf\_name}', exist\_ok=True)

        X\_plot = pd.DataFrame({'Max. Throughput [kB/s]': np.arange(200, 2500, 50)})

        X\_plot\_scaled = scaler.transform(X\_plot)

        start = time.time()

        y\_pred = model.predict(X\_plot\_scaled)

        times.append(start - time.time())

        plt.scatter(X\_plot, y\_pred, label=labels[i], marker=markers[i])

    plt.xlabel('Traffic Load [kB/s]')

    plt.ylabel('CPU')

    plt.tick\_params(axis='both', direction='inout', length=5, bottom=True, left=True, right=True, top=True)

    plt.legend()

    # save, avoid cutting off labels

    plt.tight\_layout()

    fig.savefig(f'plots/{vnf\_name}\_model\_comparison.pdf')

    return times

def barplot\_compare\_rmse(scores\_default, scores\_tuned, labels, data\_name):

    sns.set(font\_scale=1.1, style='white')

    assert len(scores\_default) == len(scores\_tuned) == len(labels)

    x = np.arange(len(labels))

    width = 0.35

    # plot

    fig, ax = plt.subplots() # prev: figsize = (8, 5))

    # default

    rmse\_mean = [s.mean() for s in scores\_default]

    rmse\_std = [s.std() for s in scores\_default]

    ci95 = [1.96 \* std / np.sqrt(len(rmse\_std)) for std in rmse\_std]

    ax.bar(x - width/2, rmse\_mean, width, yerr=ci95, capsize=5, color='gray', label='Default')

    # same for tuned version

    rmse\_mean = [s.mean() for s in scores\_tuned]

    rmse\_std = [s.std() for s in scores\_tuned]

    ci95 = [1.96 \* std / np.sqrt(len(rmse\_std)) for std in rmse\_std]

    ax.bar(x + width/2, rmse\_mean, width, yerr=ci95, capsize=5, color='lightgray', label='Tuned')

    # labels

    ax.set\_ylabel('RMSE')

    ax.set\_xticks(x)

    ax.set\_xticklabels(labels)

    ax.tick\_params(axis='both', direction='inout', length=5, bottom=False, left=True, right=True, top=False)

    ax.legend()

    fig.savefig(f'plots/{data\_name}\_rmse.pdf', bbox\_inches='tight')

def train\_tune\_eval\_models(X, y, vnf\_name):

    # prepare models and rmse without tuning

    labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

    models\_default = [LinearRegression(), Ridge(), SVR(), RandomForestRegressor(),

              GradientBoostingRegressor(), MLPRegressor(max\_iter=1500),

              FixedModel(fixed\_value=0.8)]

    rmse\_default = [cross\_validation\_rmse(model, X, y, vnf\_name, save\_model=False) for model in models\_default]

    # hyperparam tuning

    params\_ridge = {'alpha': [0.1, 1, 10]}

    params\_svr = {'kernel': ['poly', 'rbf'], 'C': [1, 10, 100],

                  'epsilon': [0.001, 0.01, 0.1]}

    params\_forest = {'n\_estimators': [10, 100, 200]}

    params\_boosting = {'learning\_rate': [0.01, 0.1, 0.3], 'n\_estimators': [10, 100, 200]}

    params\_mlp = {'hidden\_layer\_sizes': [(64,), (128,), (256)], 'alpha': [0.001, 0.0001, 0.00001],

                  'learning\_rate\_init': [0.01, 0.001, 0.0001]}

    params = [{}, params\_ridge, params\_svr, params\_forest, params\_boosting, params\_mlp, {}]

    models\_tuned = [tune\_hyperparams(models\_default[i], X, y, params[i]) for i in range(len(labels))]

    rmse\_tuned = [cross\_validation\_rmse(model, X, y, vnf\_name, save\_model=True) for model in models\_tuned]

    barplot\_compare\_rmse(rmse\_default, rmse\_tuned, labels, f'{vnf\_name}\_default-tuned')

    return models\_tuned

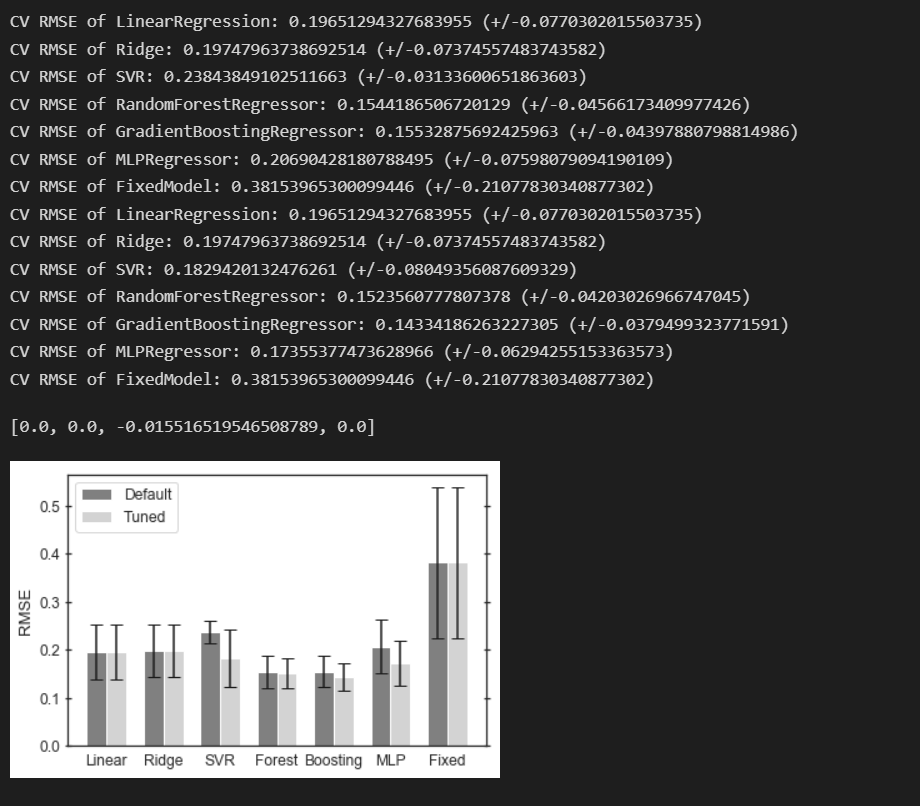
vnf\_name = 'nginx'

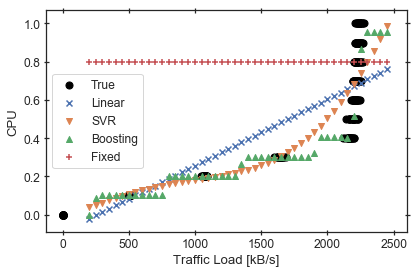
X, y, scaler = prepare\_data(nginx, vnf\_name)

X\_scaled = scaler.transform(X)

models = train\_tune\_eval\_models(X\_scaled, y, vnf\_name)

predict\_plot\_all(models, scaler, X, y, vnf\_name)





Script-4:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib as mpl

import seaborn as sns

import sklearn

import yaml

import sys

import glob

import os

%matplotlib inline

sns.set(font\_scale=1.1, style='white')

sns\_cmap = sns.color\_palette().as\_hex()

def sum\_cpu(node\_res):

    cpu = sum([v['cpu'] for v in node\_res])

    return cpu

def read\_placement(placement, df\_data, flow\_dr=250):

    df\_data['num\_flows'].append(placement['input']['num\_flows'])

    df\_data['num\_sources'].append(placement['input']['num\_sources'])

    df\_data['source\_dr'].append(placement['input']['num\_flows'] \* flow\_dr)

    df\_data['num\_instances'].append(placement['metrics']['num\_instances'])

    df\_data['max\_e2e\_delay'].append(placement['metrics']['max\_endToEnd\_delay'])

    df\_data['total\_delay'].append(placement['metrics']['total\_delay'])

    df\_data['runtime'].append(placement['metrics']['runtime'])

    df\_data['total\_cpu'].append(sum\_cpu(placement['placement']['alloc\_node\_res']))

    return df\_data

def read\_results(results):

    data = {'num\_sources' : [], 'num\_flows': [], 'source\_dr': [], 'num\_instances': [],

            'max\_e2e\_delay': [], 'total\_delay': [], 'runtime': [], 'total\_cpu': []}

    # iterate through result files

    for res in glob.glob(results):

        # open and save metrics of interest

        with open(res, 'r') as f:

            placement = yaml.load(f, Loader=yaml.SafeLoader)

            data = read\_placement(placement, data)

    return pd.DataFrame(data).sort\_values(by=['num\_flows'])

# read results

dataset = 'web\_data'

sources = 'three\_source\_dr250'

results = f'placement\_data/{dataset}/{sources}/'

df\_true = read\_results(results + 'true/\*.yaml')

df\_fixed = read\_results(results + 'fixed/\*.yaml')

df\_linear = read\_results(results + 'linear/\*.yaml')

df\_boost = read\_results(results + 'boosting/\*.yaml')

df\_svr = read\_results(results + 'svr/\*.yaml')

df\_ml = read\_results(results + 'ml/\*.yaml')

def plot(x\_col, x\_label, y\_col, y\_label, save\_plot=True, plot\_fixed=True):

    sns.set(font\_scale=1.1, style='white')

    fig, ax = plt.subplots()

    if plot\_fixed:

        plt.plot(df\_fixed[x\_col], df\_fixed[y\_col], label='Fixed', color=sns\_cmap[3], marker='+',

                 linewidth=3, linestyle="dotted", markersize=10, markeredgewidth=1.5)

    plt.plot(df\_linear[x\_col], df\_linear[y\_col], label='Linear', color=sns\_cmap[0], marker='x',

             linewidth=2, linestyle="dashed", markersize=7, markeredgewidth=1.5)

    plt.plot(df\_ml[x\_col], df\_ml[y\_col], label='SVR+Boosting', color=sns\_cmap[2], marker='s',

             linewidth=2)

    plt.xlabel(x\_label)

    plt.ylabel(y\_label)

    plt.ylim(bottom=0)

    plt.tick\_params(axis='both', direction='inout', length=5, bottom=True, left=True, right=True, top=True)

    plt.legend()

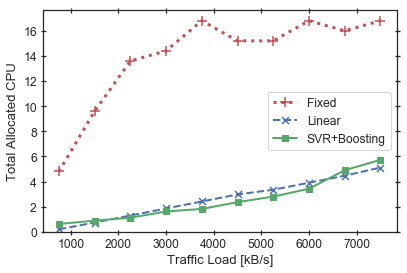
    plt.tight\_layout()

    if save\_plot:

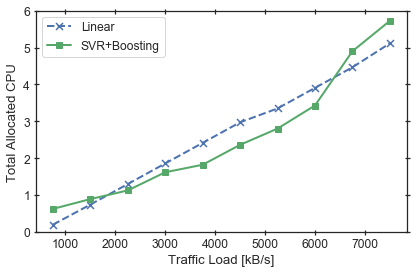
        fig.savefig(f'plots/{dataset}\_{y\_col}.pdf')

    return fig, ax

plot('source\_dr', 'Traffic Load [kB/s]', 'total\_cpu', 'Total Allocated CPU')



fig, ax = plot('source\_dr', 'Traffic Load [kB/s]', 'total\_cpu', 'Total Allocated CPU', plot\_fixed=False)



# plot both

def plot\_both(x\_col, x\_label, y\_col, y\_label):

    sns.set(font\_scale=1.1, style='white')

    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

    ax[0].plot(df\_fixed[x\_col], df\_fixed[y\_col], label='Fixed', color=sns\_cmap[3], marker='+',

               linewidth=3, linestyle="dotted", markersize=10, markeredgewidth=1.5)

    ax[0].plot(df\_linear[x\_col], df\_linear[y\_col], label='Linear', color=sns\_cmap[0], marker='x',

               linewidth=2, linestyle="dashed", markersize=7, markeredgewidth=1.5)

    ax[0].plot(df\_ml[x\_col], df\_ml[y\_col], label='SVR+Boosting', color=sns\_cmap[2], marker='s',

               linewidth=2)

    ax[0].set(xlabel=x\_label, ylabel=y\_label)

    ax[0].set\_ylim(bottom=0)

    ax[0].tick\_params(axis='both', direction='inout', length=5, bottom=True, left=True, right=True, top=True)

#     ax[0].legend()

    ax[1].plot([], [], label='Fixed', color=sns\_cmap[3], marker='+', linewidth=3,

               linestyle="dotted", markersize=10, markeredgewidth=1.5)

    ax[1].plot(df\_linear[x\_col], df\_linear[y\_col], label='Linear', color=sns\_cmap[0], marker='x',

               linewidth=2, linestyle="dashed", markersize=7, markeredgewidth=1.5)

    ax[1].plot(df\_ml[x\_col], df\_ml[y\_col], label='SVR+Boosting', color=sns\_cmap[2], marker='s',

               linewidth=2)

    ax[1].set(xlabel=x\_label)

    ax[1].set\_ylim(bottom=0)

    ax[1].tick\_params(axis='both', direction='inout', length=5, bottom=True, left=True, right=True, top=True)

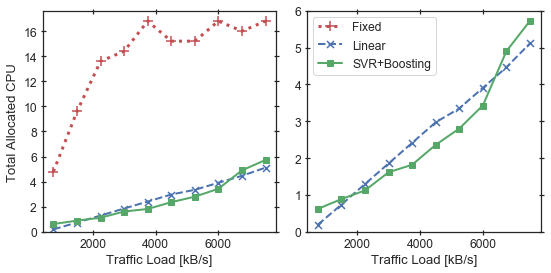
    ax[1].legend()

    # avoid cutting off figure labels

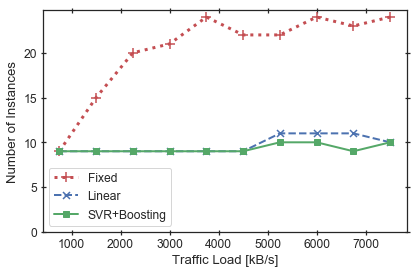
    plt.tight\_layout()

    fig.savefig(f'plots/{dataset}\_{y\_col}\_both.pdf')

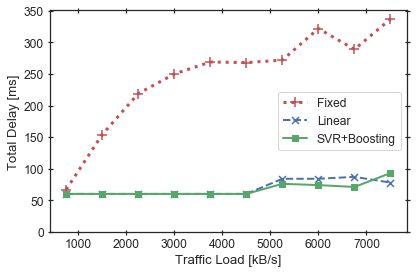
plot\_both('source\_dr', 'Traffic Load [kB/s]', 'total\_cpu', 'Total Allocated CPU')



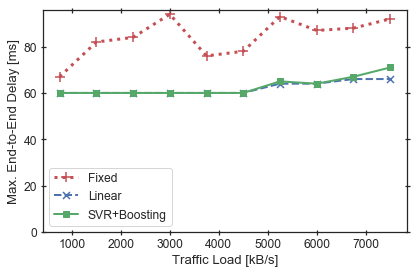
plot('source\_dr', 'Traffic Load [kB/s]', 'num\_instances', 'Number of Instances')



plot('source\_dr', 'Traffic Load [kB/s]', 'total\_delay', 'Total Delay [ms]')



plot('source\_dr', 'Traffic Load [kB/s]', 'max\_e2e\_delay', 'Max. End-to-End Delay [ms]')



Script-5:

import warnings

warnings.simplefilter(action='ignore', category=FutureWarning)

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

import random

import glob

import yaml

import timeit

import math

from sklearn.base import BaseEstimator

from sklearn.model\_selection import KFold, train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.linear\_model import LinearRegression, Ridge

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.preprocessing import LabelEncoder, StandardScaler, PolynomialFeatures

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error

sns.set(font\_scale=1.1, style='white')

class FixedModel(BaseEstimator):

    def \_\_init\_\_(self, fixed\_value):

        self.fixed\_value = fixed\_value

    def fit(self, X, y):

        return self

    def predict(self, X):

        n\_samples = X.shape[0]

        return [self.fixed\_value for \_ in range(n\_samples)]

# function for processing and simplifying the dataset

def select\_and\_rename(df, mapping):

    # select subset of columns

    dff = df[list(mapping.keys())]

    # rename

    for k, v in mapping.items():

        dff.rename(columns={k: v}, inplace=True)

    return dff

def replaceSize(df):

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -i http://20.0.0.254:8888/", "small")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k http://20.0.0.254:8888/bunny.mp4", "big")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -i -X 20.0.0.254:3128 http://40.0.0.254:80/", "small")

    df["size"] = df["size"].str.replace("ab -c 1 -t 60 -n 99999999 -e /tngbench\_share/ab\_dist.csv -s 60 -k -X 20.0.0.254:3128 http://40.0.0.254:80/bunny.mp4", "big")

    return df

web1 = pd.read\_csv("vnf\_data/csv\_experiments\_WEB1.csv")

web3 = pd.read\_csv("vnf\_data/csv\_experiments\_WEB3.csv")

mapping = {

    "param\_\_func\_\_mp.input\_\_cmd\_start": "size",

    "metric\_\_mp.input.vdu01.0\_\_ab\_transfer\_rate\_kbyte\_per\_second": "Max. throughput [kB/s]",

}

mapping01 = mapping.copy()

mapping01["param\_\_func\_\_de.upb.lb-nginx.0.1\_\_cpu\_bw"] = "CPU"

mapping01["param\_\_func\_\_de.upb.lb-nginx.0.1\_\_mem\_max"] = "Memory"

mapping03 = mapping.copy()

mapping03["param\_\_func\_\_de.upb.px-squid.0.1\_\_cpu\_bw"] = "CPU"

mapping03["param\_\_func\_\_de.upb.px-squid.0.1\_\_mem\_max"] = "Memory"

web1 = select\_and\_rename(web1, mapping01)

web3 = select\_and\_rename(web3, mapping03)

web1 = replaceSize(web1)

web3 = replaceSize(web3)

# select sub-datasets with small and large flows

# and with specific memory

mem = 128

web1\_small = web1.loc[(web1["size"] == "small") & (web1["Memory"] == mem)]

web1\_small = web1\_small[["Max. throughput [kB/s]", "CPU"]]

web3\_small = web3.loc[(web3["size"] == "small")  & (web3["Memory"] == mem)]

web3\_small = web3\_small[["Max. throughput [kB/s]", "CPU"]]

# add 20 "measurements" at 0 CPU and throuhgput

num\_measures = 20

measures = [0 for \_ in range(num\_measures)]

web1\_small = web1\_small.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

web3\_small = web3\_small.append(pd.DataFrame({'Max. throughput [kB/s]': measures, 'CPU': measures}), ignore\_index=True)

def prepare\_data(data):

    X = data[['Max. throughput [kB/s]']]

    y = data['CPU']

    X = X.fillna(X.median())

    scaler = MinMaxScaler()

    scaler.fit\_transform(X)

    return X, y

def barplot(times, labels, ylabel='Time [s]', filename=None):

    assert len(times) == len(labels)

    times\_mean = [np.array(t).mean() for t in times]

    print(times\_mean)

    times\_std = [np.array(t).std() for t in times]

    x = np.arange(len(labels))

    sns.set(font\_scale=1.1, style='white')

    fig, ax = plt.subplots(figsize=(8, 4))

    plt.bar(x, times\_mean, yerr=times\_std, capsize=5, color='gray')

    ax.set\_xticks(x)

    ax.set\_xticklabels(labels)

    ax.set\_ylabel(ylabel)

    if filename is not None:

        fig.savefig(f'plots/{filename}.pdf', bbox\_inches='tight')

# prepare data

X\_nginx, y\_nginx = prepare\_data(web1\_small)

X\_squid, y\_squid = prepare\_data(web3\_small)

X = X\_nginx

y = y\_nginx

vnf\_name = 'nginx'

# generate synthetic data

def gen\_benchmark(cpu, coeff1=1, coeff2=1):

    return coeff1 \* math.log2(1 + coeff2 \* cpu)

def synthetic\_benchmark(n):

    cpu\_list = [random.random() for \_ in range(n)]

    thr\_list = [gen\_benchmark(cpu, coeff2=100) for cpu in cpu\_list]

    X = pd.DataFrame(data={'Throughput': thr\_list})

    y = np.array(cpu\_list)

    return X, y

def training\_times(models, X, y, scaler):

    # measure training times

    times = []

    for model in models:

        print(type(model).\_\_name\_\_)

        X\_scaled = scaler.transform(X)

        t = %timeit -o model.fit(X, y)

        times.append(t)

    return times

labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

models = [LinearRegression(), Ridge(), SVR(verbose=True), RandomForestRegressor(n\_estimators=10),

          GradientBoostingRegressor(), MLPRegressor(max\_iter=1500),

          FixedModel(fixed\_value=0.8)]

model\_names = [type(model).\_\_name\_\_ for model in models]

models = [joblib.load(f'ml\_models/{vnf\_name}/{name}.joblib') for name in model\_names]

scaler = joblib.load(f'ml\_models/{vnf\_name}/scaler.joblib')

models = [joblib.load(f'ml\_models/{vnf\_name}/{name}.joblib') for name in model\_names]

scaler = joblib.load(f'ml\_models/{vnf\_name}/scaler.joblib')

X\_rand = pd.DataFrame(data={'Rand max. throughput': [random.randrange(0, 3000) for \_ in range(1)]})

times = []

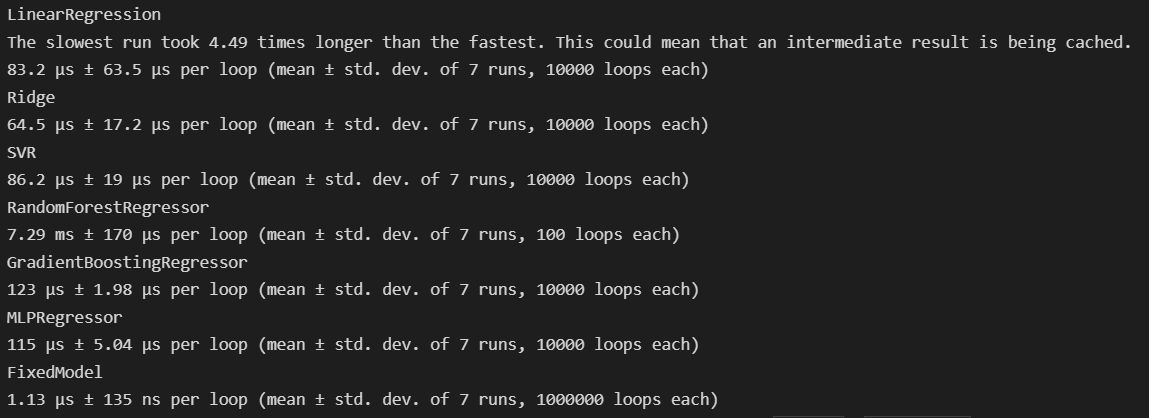
for model in models:

    print(type(model).\_\_name\_\_)

    X\_scaled = scaler.transform(X\_rand)

    t = %timeit -o model.predict(X\_scaled)

    times.append(t)



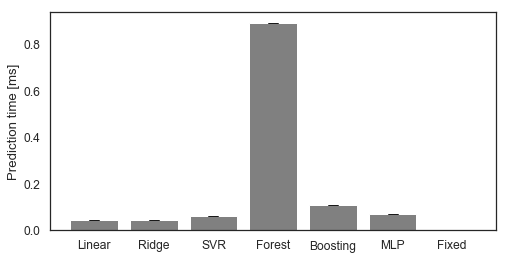
all\_times\_ms = [[i \* 1000.0 / t.loops for i in t.all\_runs] for t in times]

all\_times\_ms = [0.041868121428571416, 0.04201067571428731, 0.059522719999998634, 0.8912386999999821, 0.10723779999999741,

                0.06644656000000201, 0.0009462411285714519]

labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

barplot(all\_times\_ms, labels, ylabel="Prediction time [ms]", filename='prediction\_times')



def sum\_cpu(node\_res):

    cpu = sum([v['cpu'] for v in node\_res])

    return cpu

def read\_placement(placement, df\_data, flow\_dr=250):

    df\_data['num\_flows'].append(placement['input']['num\_flows'])

    df\_data['num\_sources'].append(placement['input']['num\_sources'])

    df\_data['source\_dr'].append(placement['input']['num\_flows'] \* flow\_dr)

    df\_data['num\_instances'].append(placement['metrics']['num\_instances'])

    df\_data['max\_e2e\_delay'].append(placement['metrics']['max\_endToEnd\_delay'])

    df\_data['total\_delay'].append(placement['metrics']['total\_delay'])

    df\_data['runtime'].append(placement['metrics']['runtime'])

    df\_data['total\_cpu'].append(sum\_cpu(placement['placement']['alloc\_node\_res']))

    return df\_data

def read\_results(results):

    data = {'num\_sources' : [], 'num\_flows': [], 'source\_dr': [], 'num\_instances': [],

            'max\_e2e\_delay': [], 'total\_delay': [], 'runtime': [], 'total\_cpu': []}

    # iterate through result files

    for res in glob.glob(results):

        # open and save metrics of interest

        with open(res, 'r') as f:

            placement = yaml.load(f, Loader=yaml.SafeLoader)

            data = read\_placement(placement, data)

    return pd.DataFrame(data).sort\_values(by=['num\_flows'])

# read results

dataset = 'web\_data'

sources = 'runtime'

results = f'placement\_data/{dataset}/{sources}/'

df\_results = []

df\_results.append(read\_results(results + 'linear/\*.yaml'))

df\_results.append(read\_results(results + 'ridge/\*.yaml'))

df\_results.append(read\_results(results + 'svr/\*.yaml'))

df\_results.append(read\_results(results + 'forest/\*.yaml'))

df\_results.append(read\_results(results + 'boosting/\*.yaml'))

df\_results.append(read\_results(results + 'mlp/\*.yaml'))

df\_results.append(read\_results(results + 'fixed/\*.yaml'))

# df\_results.append(read\_results(results + 'ml/\*.yaml'))

labels = ['Linear', 'Ridge', 'SVR', 'Forest', 'Boosting', 'MLP', 'Fixed']

placement\_times = [df['runtime'] for df in df\_results]

barplot(placement\_times, labels)

def comparison\_barplot\_one(pred\_times, place\_times, labels, filename=None):

    sns.set(font\_scale=1.1, style='white')

    assert len(pred\_times) == len(place\_times) == len(labels)

    # prepare data

    pred\_times\_mean = [np.array(t).mean() for t in pred\_times]

    pred\_times\_std = [np.array(t).std() for t in pred\_times]

    place\_times\_mean = [np.array(t).mean() for t in place\_times]

    place\_times\_std = [np.array(t).std() for t in place\_times]

    x = np.arange(len(labels))

    width = 0.35

    # plot: separate axes

    fig, ax1 = plt.subplots(figsize = (8, 5))

    ax1.bar(x - width/2, pred\_times\_mean, width, yerr=pred\_times\_std, capsize=5, color='gray', label='Prediction')

    ax1.set\_ylabel("Prediction time [ms]")

    ax1.set\_xticks(x)

    ax1.set\_xticklabels(labels)

    ax2 = ax1.twinx()

    ax2.bar(x + width/2, place\_times\_mean, width, yerr=place\_times\_std, capsize=5, color='lightgray', label='Placement')

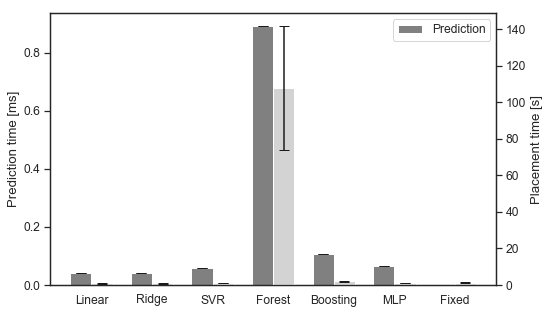
    ax2.set\_ylabel("Placement time [s]")

    # labels

    ax1.legend()

    if filename is not None:

        fig.savefig(f'plots/{filename}.pdf', bbox\_inches='tight')



from matplotlib.ticker import ScalarFormatter

def comparison\_barplot\_two(pred\_times, place\_times, labels):

    sns.set(font\_scale=1.1, style='white')

    assert len(pred\_times) == len(place\_times) == len(labels)

    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey='all', figsize=(8, 4))

    # prepare data

    pred\_times\_mean = [np.array(t).mean() for t in pred\_times]

    pred\_times\_std = [np.array(t).std() for t in pred\_times]

    # 95% CI (not bootstrapped like seaborn does): https://en.wikipedia.org/wiki/Confidence\_interval#Basic\_steps

    pred\_times\_ci95 = [1.96 \* std / np.sqrt(len(pred\_times\_std)) for std in pred\_times\_std]

    place\_times\_mean = [np.array(t).mean() for t in place\_times]

    place\_times\_std = [np.array(t).std() for t in place\_times]

    place\_times\_ci95 = [1.96 \* std / np.sqrt(len(place\_times\_std)) for std in place\_times\_std]

    x = np.arange(len(labels))

    # plot: separate axes

    ax1.barh(x, pred\_times\_mean, xerr=pred\_times\_ci95, capsize=5, color='gray', label='Prediction')

    ax1.set\_xlabel("Prediction Time [ms]")

#     ax1.set\_xscale('log')

    ax1.set\_yticks(x)

    ax1.set\_yticklabels(labels)

    ax1.tick\_params(axis='both', direction='inout', length=5, bottom=True, top=True)

    ax2.barh(x, place\_times\_mean, xerr=place\_times\_ci95, capsize=5, color='lightgray', label='Placement')

    ax2.set\_xlabel("Total Time [s]")

    # log scale; avoid "power 10" labels

    ax2.set\_xscale('log')

    ax2.tick\_params(axis='both', direction='inout', length=5, bottom=True, top=True)

    ax2.xaxis.set\_major\_formatter(ScalarFormatter())

    plt.tight\_layout()

    fig.savefig(f'plots/runtimes.pdf', bbox\_inches='tight')

comparison\_barplot\_two(list(reversed(all\_times\_ms)), list(reversed(placement\_times)), list(reversed(labels)))

