ML HW5

- Aliaksei Kanstantsinau, Simon Geller, Gabriel De La Noue, Marc Khamis, Adrien Flambard

In [11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [48]:
```
# import df for problem 2
df = pd.read_csv('housing.csv')
df.head()
```

Out[48]:

	crim	zn	indus	chas	rm	age	dis	rad	tax	ptratio	lstat	medv	
0	0.00632	18.0	2.31	0	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	0.
1	0.02731	0.0	7.07	0	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	0.
2	0.02729	0.0	7.07	0	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	0.
3	0.03237	0.0	2.18	0	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	0.
4	0.06905	0.0	2.18	0	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	0.

Question 1:

Simulate 1500 realizations of two uncorrelated standard Normal variables. Call the simulated variables x_1 and x_2 and use these simulated variables as your predictors for y. Simulate 1500 outcomes for y for each of the two models:

a) $y = 1.5x_1 - 2x_2 + \epsilon $

b) $y = \begin{cases} 1.5 \ln(x^2) + \epsilon & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$

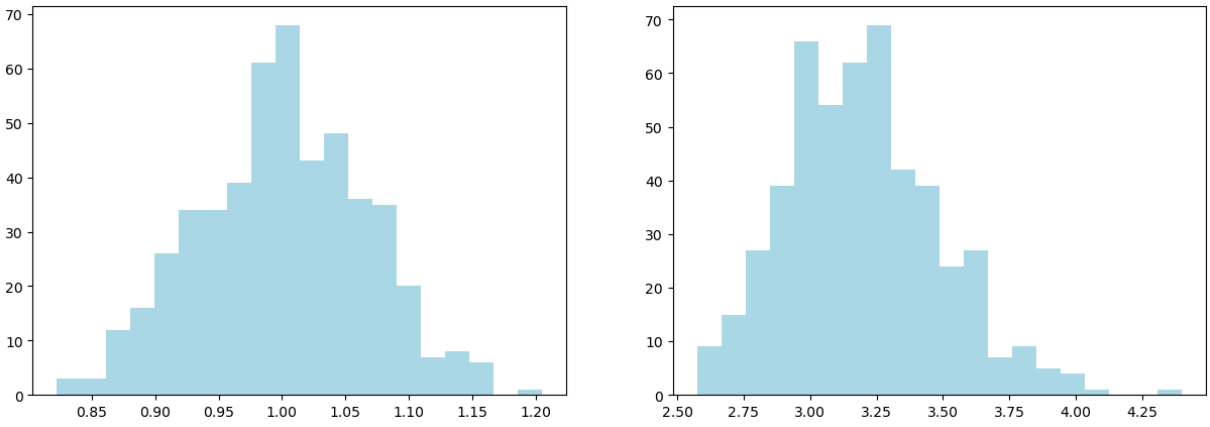
(i) OLS regression:

In [ ]:
```
# define function for 1(i)
def mse_func(simulations, outcomes, type):
    mse_values = []
    for _ in range(simulations):
    # simulate x1, x2 and error term
        x_1 = np.random.normal(0, 1, outcomes)
        x_2 = np.random.normal(0, 1, outcomes)
        error_term = np.random.normal(0, 1, 1500)
        # stack predictors
        predictors = np.column_stack((x_1, x_2))
        # train and test
```

```
        predictors_train = predictors[:1000]
        predictors_test = predictors[1000:]
        # simulate y_s
        if type == 'a':
            y = 1.5 * x_1 - 2 * x_2 + error_term
        elif type == 'b':
            y = np.where(x_1 < 0, 1.5*x_1 - 2*x_2 + error_term, 1.5*np.log(x
        else:
            raise ValueError('a or b')
        # train and test
        y_train = y[:1000]
        y_test = y[1000:]
        model = LinearRegression().fit(predictors_train, y_train)
        y_pred = model.predict(predictors_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_values.append(mse)
    return mse_values
```

In [49]:
```
import warnings
warnings.filterwarnings('ignore', category=RuntimeWarning)
#plot the result
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].hist(mse_func(500, 1500, 'a'), bins=20, color='lightblue')
ax[1].hist(mse_func(500, 1500, 'b'), bins=20, color='lightblue')
```

Out[49]:
```
(array([ 9., 15., 27., 39., 66., 54., 62., 69., 42., 39., 24., 27.,  7.,
         9.,  5.,  4.,  1.,  0.,  0.,  1.]),
 array([2.57414488, 2.66538102, 2.75661716, 2.84785331, 2.93908945,
        3.03032559, 3.12156173, 3.21279787, 3.30403401, 3.39527015,
        3.4865063 , 3.57774244, 3.66897858, 3.76021472, 3.85145086,
        3.942687  , 4.03392315, 4.12515929, 4.21639543, 4.30763157,
        4.39886771]),
 <BarContainer object of 20 artists>)
```



(ii) Random Forest with n_estimators=250 and max_depth=10:

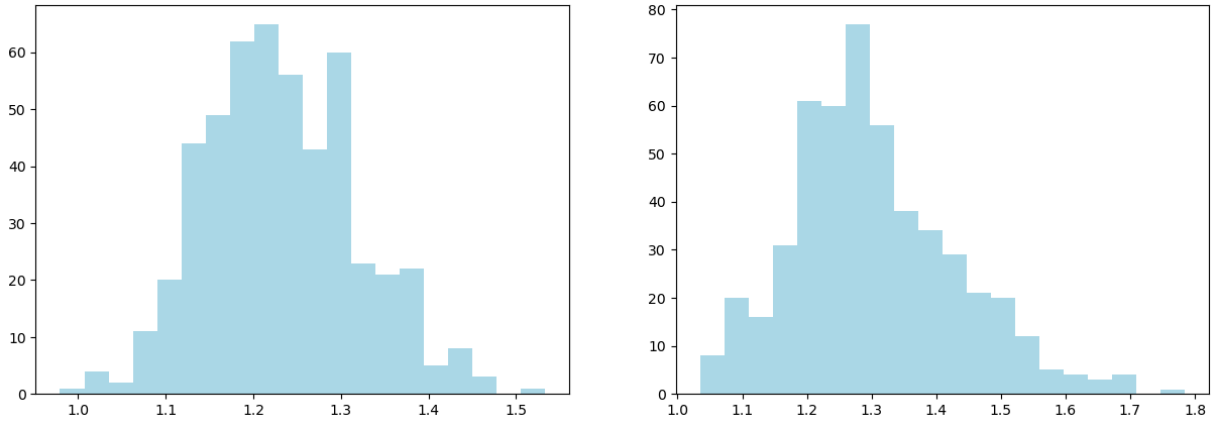
In [50]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [51]:
```
# define function for 1(ii)
def rain_forest(simulations, outcomes, type):
    mse_values = []
    for _ in range(simulations):
```

```
    # simulate x1, x2 and error term
        x_1 = np.random.normal(0, 1, outcomes)
        x_2 = np.random.normal(0, 1, outcomes)
        error_term = np.random.normal(0, 1, 1500)
        # stack predictors
        predictors = np.column_stack((x_1, x_2))
        # train and test
        predictors_train = predictors[:1000]
        predictors_test = predictors[1000:]
        # simulate y_s
        if type == 'a':
            y = 1.5 * x_1 - 2 * x_2 + error_term
        elif type == 'b':
            y = np.where(x_1 < 0, 1.5*x_1 - 2*x_2 + error_term, 1.5*np.log(x
        else:
            raise ValueError('a or b')
        # train and test
        y_train = y[:1000]
        y_test = y[1000:]
        rf_model = RandomForestRegressor(n_estimators=250, max_depth=10, rar
        rf_model.fit(predictors_train, y_train)
        y_pred = rf_model.predict(predictors_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_values.append(mse)
    return mse_values
```

In [ ]:
```
#plot the result
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].hist(rain_forest(500, 1500, 'a'), bins=20, color='lightblue')
ax[1].hist(rain_forest(500, 1500, 'b'), bins=20, color='lightblue')
```

Out[ ]:
```
(array([ 8., 20., 16., 31., 61., 60., 77., 56., 38., 34., 29., 21., 20.,
        12.,  5.,  4.,  3.,  4.,  0.,  1.]),
 array([1.03456726, 1.07205142, 1.10953557, 1.14701973, 1.18450388,
        1.22198803, 1.25947219, 1.29695634, 1.3344405 , 1.37192465,
        1.40940881, 1.44689296, 1.48437711, 1.52186127, 1.55934542,
        1.59682958, 1.63431373, 1.67179788, 1.70928204, 1.74676619,
        1.78425035]),
 <BarContainer object of 20 artists>)
```



(iii) XGBoost with learning_rate=0.3, gamma=0, and max_depth=6; use 20 rounds and 10 folds for the cross-validation procedure. Make sure that the output of the cross-

validation procedure does not appear in your final write-up. Note that you can use an in-sample cross-validation procedure to determine the optimal values for the decision tree parameters. However, you are not required to do so for this exercise.

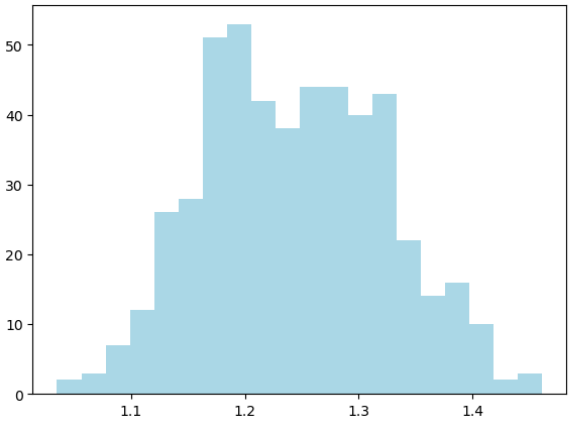
In [ ]:
```
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score
```

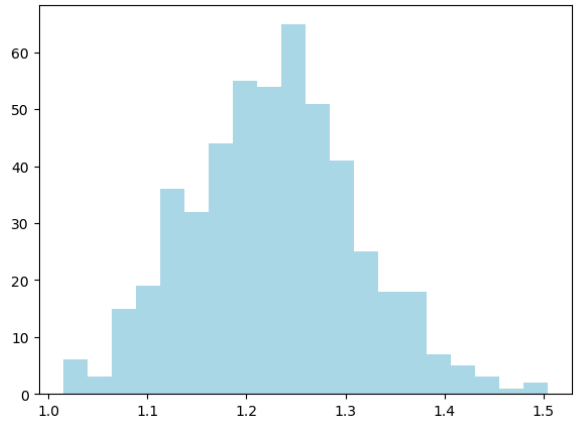
In [80]:
```
#def function for 1(iii)
def xgboost(simulations, outcomes, type):
    mse_values = []
    for _ in range(simulations):
    # simulate x1, x2 and error term
        x_1 = np.random.normal(0, 1, outcomes)
        x_2 = np.random.normal(0, 1, outcomes)
        error_term = np.random.normal(0, 1, 1500)
        # stack predictors
        predictors = np.column_stack((x_1, x_2))
        # train and test
        predictors_train = predictors[:1000]
        predictors_test = predictors[1000:]
        # simulate y_s
        if type == 'a':
            y = 1.5 * x_1 - 2 * x_2 + error_term
        elif type == 'b':
            y = np.where(x_1 < 0, 1.5*x_1 - 2*x_2 + error_term, 1.5*np.log(r
        else:
            raise ValueError('a or b')
        # train and test
        y_train = y[:1000]
        y_test = y[1000:]
        # kfold/model
        kfold = KFold(n_splits=10, shuffle=True, random_state=96)
        xgb_model = xgb.XGBRegressor(learning_rate = .3, gamma= 0, max_depth

        xgb_model.fit(predictors_train, y_train)
        y_pred = xgb_model.predict(predictors_test)
        mse = mean_squared_error(y_test, y_pred)
        mse_values.append(mse)
    return mse_values
```

In [81]:
```
#plot the result
fig, ax = plt.subplots(1, 2, figsize=(15, 5))
mse_a = xgboost(500, 1500, 'a')
mse_b = xgboost(500, 1500, 'b')
ax[0].hist(mse_a, bins=20, color='lightblue')
ax[1].hist(mse_b, bins=20, color='lightblue')
```

Out[81]:
```
(array([ 6.,  3., 15., 19., 36., 32., 44., 55., 54., 65., 51., 41., 25.,
        18., 18.,  7.,  5.,  3.,  1.,  2.]),
 array([1.01455117, 1.03904017, 1.06352917, 1.08801816, 1.11250716,
        1.13699616, 1.16148515, 1.18597415, 1.21046315, 1.23495215,
        1.25944114, 1.28393014, 1.30841914, 1.33290813, 1.35739713,
        1.38188613, 1.40637512, 1.43086412, 1.45535312, 1.47984211,
        1.50433111]),
 <BarContainer object of 20 artists>)
```





Interpret the histograms. Which of the models (i), (ii), and (iii) do best in the out-of-sample exercise for models (a) and (b)? Do the histograms conform to your expectations given the data generating processes in parts (a) and (b)?

- Model three does the best job with the data, it give more consistent data at a quicker time.

Question 2:

Attached to this problem set is a dataset which deals with Boston real estate prices. The dataset can be found on the UCI Machine Learning Depository: https://archive.ics.uci.edu/ml/index.php. Our goal in this exercise is to predict house prices in Boston (medv) given 11 explanatory variables (columns 1 through 11). Use the first 400 observations as your training sample and observations 401-506 as your test sample.

(a) Use random forest with n_estimators=250 and max_depth=10. Once you run the random forest, use Python's rf.predict function to obtain predicted values for the test sample. What is the MSE of the prediction? Compare this to the benchmark MSE generated by a model that has as its predicted house value the mean house value in the test sample. As in the class notes, also report the Pseudo-R2 implied by these MSEs.

In [53]:
```
df.head()
```

Out[53]:

	crim	zn	indus	chas	rm	age	dis	rad	tax	ptratio	lstat	medv	
0	0.00632	18.0	2.31	0	6.575	65.2	4.0900	1	296	15.3	4.98	24.0	0.
1	0.02731	0.0	7.07	0	6.421	78.9	4.9671	2	242	17.8	9.14	21.6	0.
2	0.02729	0.0	7.07	0	7.185	61.1	4.9671	2	242	17.8	4.03	34.7	0.
3	0.03237	0.0	2.18	0	6.998	45.8	6.0622	3	222	18.7	2.94	33.4	0.
4	0.06905	0.0	2.18	0	7.147	54.2	6.0622	3	222	18.7	5.33	36.2	0.

In [62]:
```
# set predictors
predictors = df.columns.drop(['medv'])
train = df.iloc[:400]
test = df.iloc[400:]
# train test
X_train = train
X_test = test
y_train = train['medv']
y_test = test['medv']
# run regression
rf_model = RandomForestRegressor(n_estimators=250, max_depth=10, random_stat
rf_model.fit(X_train, y_train)
# get prediction
rf_prediction = rf_model.predict(X_test)
# get mse
mse_rf = mean_squared_error(y_test, rf_prediction)
# get mse mean
mse_mean = y_test.mean()
# get benchmark prediction, mse, and pseudo r2
benchmark_predictions = [mse_mean] * len(y_test)
mse_benchmark = mean_squared_error(y_test, benchmark_predictions)
pseudo_r2 = 1 - (mse_rf / mse_benchmark)
print('MSE RandomForest:' , mse_rf,
      ' MSE Benchmark:', mse_benchmark,
      'Pseudo R^2:', pseudo_r2)
```

```
MSE RandomForest: 0.03759581886792428  MSE Benchmark: 28.25733713065147 Pseu
do R^2: 0.9986695201075
```

In [63]:
```
# xgb model + kfold + crossval
xgb_model = xgb.XGBRegressor(learning_rate = .1, gamma= 0, max_depth = 6, n_
kfold = KFold(n_splits=10, shuffle=True, random_state=96)
cross_val_scores = cross_val_score(xgb_model, X_train, y_train, cv=kfold)
# model fit + prediction
xgb_model.fit(X_train, y_train, verbose=False)
xgb_prediction = xgb_model.predict(X_test)
mse_xgb = mean_squared_error(y_test, xgb_prediction)
pseudo_r2_xgb = 1 - (mse_xgb / mse_benchmark)
print('MSE xbg:', mse_xgb,
      'Pseudo R^2:', pseudo_r2_xgb)
```

```
MSE xbg: 0.07549695967785687 Pseudo R^2: 0.997328235164949
```

In [64]:
```
from sklearn.linear_model import ElasticNetCV
```

In [66]:
```
elastic_model = ElasticNetCV(l1_ratio=0.5, cv=4, random_state=69)
elastic_model.fit(X_train, y_train)
# Predict on the test data
elastic_prediction = elastic_model.predict(X_test)
mse_el = mean_squared_error(y_test, elastic_prediction)
pseudo_r2_el = 1 - (mse_el / mse_benchmark)
print('MSE Elastic:', mse_el,
      'Pseudo R^2:', pseudo_r2_el)
```

```
MSE Elastic: 0.0032457444890708034 Pseudo R^2: 0.999885136222353
```

Try to figure out what the main sources of the discrepancy between Elastic Net and the decision trees are. That is, what is the non-linearity?

- The disreptancies come from the way the models handle the data. Elastic net is a type of linear regression that does a great job when the relationship between predictors and predictand is linear. Decision trees are non-linear and can assume more complex (non-linear) interactions between predictors, while linear regression (elastic net in our example) would most likely miss out on some parameters.