Chapter Goals

- How to combine multiple models to solve a single problem, usually solved by a single model.

General Lab Guidlines

- Visualization.
- Modifiable code snippets.

```
# Loading a dataset
# dataset names: "airline", "breast-cancer", "contact-lenses", "cpu",
"cpu.with.vendor", "credit-g", "diabetes", "glass", "hypothyroid",
"ionosphere", "iris.2D", "iris", "labor", "segment-challenge",
"segment-test", "soybean", "supermarket", "unbalanced", "vote",
"weather.nominal", "weather.numeric"
# df = pd.read_csv("data/weather.numeric.csv")
# instances = loader.load_file("data/weather.numeric.arff")
```

Modules & Datasets Setup

```
# @title
!apt-get install default-jdk
!apt install libgraphviz-dev
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
default-jdk is already the newest version (2:1.11-72build2).
0 upgraded, 0 newly installed, 0 to remove and 15 not upgraded.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libgraphviz-dev is already the newest version (2.42.2-6).
0 upgraded, 0 newly installed, 0 to remove and 15 not upgraded.
```

```
# @title
!pip install pygraphviz
!pip install python-javabridge
!pip install python-weka-wrapper3
!pip install sklearn-weka-plugin
```

```
Requirement already satisfied: pygraphviz in
/usr/local/lib/python3.10/dist-packages (1.11)
```

```
Requirement already satisfied: python-javabridge in
/usr/local/lib/python3.10/dist-packages (4.0.3)
Requirement already satisfied: numpy>=1.20.1 in
/usr/local/lib/python3.10/dist-packages (from python-javabridge)
(1.23.5)
Requirement already satisfied: python-weka-wrapper3 in
/usr/local/lib/python3.10/dist-packages (0.2.14)
Requirement already satisfied: python-javabridge>=4.0.0 in
/usr/local/lib/python3.10/dist-packages (from python-weka-wrapper3)
(4.0.3)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from python-weka-wrapper3)
(1.23.5)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from python-weka-wrapper3)
(23.2)
Requirement already satisfied: configurable-objects in
/usr/local/lib/python3.10/dist-packages (from python-weka-wrapper3)
(0.0.1)
Requirement already satisfied: simple-data-flow in
/usr/local/lib/python3.10/dist-packages (from python-weka-wrapper3)
(0.0.1)
Collecting sklearn-weka-plugin
  Using cached sklearn-weka-plugin-0.0.7.tar.gz (69 kB)
  Preparing metadata (setup.py) ... ent already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from sklearn-weka-plugin)
(1.23.5)
Requirement already satisfied: python-weka-wrapper3>=0.2.5 in
/usr/local/lib/python3.10/dist-packages (from sklearn-weka-plugin)
(0.2.14)
Collecting sklearn (from sklearn-weka-plugin)
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a
problem with pip.
  Preparing metadata (setup.py) ... error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.

# @title
#Restart runtime after installing the dependencies
```

```python
# @title
import os
import glob
import numpy as np
import pandas as pd
import weka.core.jvm as jvm
from weka.core import converters
import matplotlib.pyplot as plt
```

```python
# @title
data_dir = 'data'
```

```python
# @title
#!rm -r weka
#!rm -r data
```

```python
# @title
#jvm.stop()
jvm.start(packages=True)
```

```
DEBUG:weka.core.jvm:Adding bundled jars
DEBUG:weka.core.jvm:Classpath=['/usr/local/lib/python3.10/dist-
packages/javabridge/jars/rhino-1.7R4.jar',
'/usr/local/lib/python3.10/dist-packages/javabridge/jars/runnablequeue
.jar',
'/usr/local/lib/python3.10/dist-packages/javabridge/jars/cpython.jar',
'/usr/local/lib/python3.10/dist-packages/weka/lib/core.jar',
'/usr/local/lib/python3.10/dist-packages/weka/lib/python-weka-
wrapper.jar',
'/usr/local/lib/python3.10/dist-packages/weka/lib/mtj.jar',
'/usr/local/lib/python3.10/dist-packages/weka/lib/weka.jar',
'/usr/local/lib/python3.10/dist-packages/weka/lib/arpack_combined.jar'
]
DEBUG:weka.core.jvm:MaxHeapSize=default
DEBUG:weka.core.jvm:Package support enabled
```

```python
# @title
# Preparing Datasets
if not os.path.exists(data_dir):
    !mkdir $data_dir
    for file in ['airline.arff', 'breast-cancer.arff', 'contact-
lenses.arff', 'cpu.arff', 'cpu.with.vendor.arff', 'credit-g.arff',
'diabetes.arff', 'glass.arff', 'hypothyroid.arff', 'ionosphere.arff',
'iris.2D.arff', 'iris.arff', 'labor.arff', 'segment-challenge.arff',
'segment-test.arff', 'soybean.arff', 'supermarket.arff',
'unbalanced.arff', 'vote.arff', 'weather.nominal.arff',
'weather.numeric.arff',]:
        url =
'https://git.cms.waikato.ac.nz/weka/weka/-/raw/main/trunk/wekadocs/
data/' + file
```

```
        !wget -P $data_dir $url
    loader =
converters.Loader(classname="weka.core.converters.ArffLoader")
    saver =
converters.Saver(classname="weka.core.converters.CSVSaver")
    for file in glob.glob(os.path.join(data_dir, '*.arff')):
        dataset = loader.load_file(file)
        filename, file_extension = os.path.splitext(file)
        saver.save_file(dataset, filename + '.csv')
    !wget -P $data_dir https://raw.githubusercontent.com/Rytuo/ITMO-
CT/master/Others/AdvancedML/data/OpenML/data/1438.arff
    !rm -r weka

# @title
import weka.core.packages as packages
packages.install_package("simpleEducationalLearningSchemes")
packages.install_package("generalizedSequentialPatterns")
packages.install_package("classAssociationRules")
packages.install_package("NNge")
packages.install_package("LibSVM")

from weka.core.converters import Loader
loader = Loader(classname="weka.core.converters.ArffLoader")
```

12.1 Combining Multiple Models

12.2 Bagging

```
# Bagging by voting
# for each iteration
# sample data
# train a model
# upon classification, call all trained models
# classify by voting

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Create a dummy dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=10, n_classes=2, random_state=42)

# Split the data into training and testing sets
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the base classifier
base_classifier = DecisionTreeClassifier(random_state=42)

# Alternatively, you can use a VotingClassifier for comparison
voting_classifier = VotingClassifier(estimators=[('base_classifier',
base_classifier)], voting='soft')
voting_classifier.fit(X_train, y_train)

# Make predictions using the VotingClassifier
y_pred_voting = voting_classifier.predict(X_test)

# Evaluate accuracy
accuracy_voting = accuracy_score(y_test, y_pred_voting)
print(f"Accuracy with VotingClassifier: {accuracy_voting:.2f}")
```

```
Accuracy with VotingClassifier: 0.83
```

```python
# Bagging by numeric weighted average
# for each iteration
# sample data
# train a model
# upon regression, call all trained models, and take weighted average

from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
import numpy as np

# Create a dummy regression dataset
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the base regressor
base_regressor = DecisionTreeRegressor(random_state=42)

# Define the BaggingRegressor with weighted average
bagging_regressor = BaggingRegressor(base_regressor, n_estimators=10,
random_state=42)

# Fit the BaggingRegressor on the training data
bagging_regressor.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = bagging_regressor.predict(X_test)

# Evaluate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 7484.15
```

12.2.1 Task Inject noise in some given data set and train a model only on it. Now apply the bagging method and observe how it guards against the noise.

12.3 Randomization

```
# Random subspace method
# For each iteration
# take a random subset of features, and subset of rows
# train a model
# upon classification, aggregate all models' answers

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score

# Create a dummy classification dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=10, n_classes=2, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the Random Forest Classifier with Random Subspace
random_subspace_classifier = RandomForestClassifier(n_estimators=10,
max_features='sqrt', random_state=42)

# Fit the Random Forest on the training data
random_subspace_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = random_subspace_classifier.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.87
```

Task 12.3.1 Modify the parameters of randomness, and compare corresponding models' performances. At which thresholds the model is benefited and harmed? Explain.

12.4 Boosting

```
# iterative unlike bagging where models are trained separately
# new models are focused on instances handled incorrectly by earlier
ones.
# incorrect instances are given more weights, so next models are more
biased towards them
# weighting models' contributions by their performance, rather than
giving equal weight to all models

from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Create a dummy classification dataset
X, y = make_classification(n_samples=1000, n_features=20,
n_informative=10, n_classes=2, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier(n_estimators=50,
random_state=42)

# Fit the AdaBoostClassifier on the training data
adaboost_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = adaboost_classifier.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.85
```

Task 12.4.1 Rather than giving slightly more weights to misclassified instances, give them 100% of weights, totally ignoring correctly classified instances. What do you observe? Explain.

12.5 Additive Regression

```python
# forward stagewise additive modeling.
# starts with an empty ensemble and incorporates new members sequentially.
# At each stage the model that maximizes the predictive performance of the ensemble as a whole is added,
# without altering those already in the ensemble.
# next model should focus on those training instances on which the ensemble performs poorly.

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np

# Create a dummy regression dataset
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the GradientBoostingRegressor
gradient_boosting_regressor = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Fit the GradientBoostingRegressor on the training data
gradient_boosting_regressor.fit(X_train, y_train)

# Make predictions on the test set
y_pred = gradient_boosting_regressor.predict(X_test)

# Evaluate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 3052.38
```

Task 12.5.1 Consider the misclassified instances at some stage. Compare the ensemble's performance among integrating different models into it.

12.6 Interpretable Ensembles

Task 12.6.1 Recall *model trees* covered in previous notebooks, and think how boosting algorithm can be applied to build them.

12.7 Stacking

Task 12.7.1 Build a Naive Bayes learner, and an instance-based learning scheme and combine them to form a classifier by voting.

The voting aggregation method you used does not conform to the *stacking* methodology. The idea is to build a meta-model, i.e a model above many models, learning how to use them.

Task 12.7.2 Take the outputs of both Naive Bayes and instance-based model, and re-interpret them as inputs to a new model. Do simple statistical analysis and manually combine them by a rule of your choice. Alternatively build a meta-model.