Lab Session 2

¶

«Decision tree, Random forest and logistic regression»

---

1. Introduction

In this lab session, you will model datasets into representitive decision trees, that will be used later to form a random forest. Next you will explore logistic regression. A set of exercises is proposed to test and compare the accuray of different methods.

2. Learning outcome

On successful completion of this lab session you will be able to :

- Handle mglearn datasets.
- represent your data set in Decision tree
- Build models Random forest
- Build models using Logistic regression

3. Ressources

Libraries Documentation

- Python : https://docs.python.org/3/ (https://docs.python.org/3/)
- NumPy : https://numpy.org/doc/ (https://numpy.org/doc/)
- SciPy : https://docs.scipy.org/doc/scipy/ (https://docs.scipy.org/doc/scipy/)
- Matplotlib : https://matplotlib.org/3.5.1/ (https://matplotlib.org/3.5.1/)
- Panda : https://pandas.pydata.org/docs/ (https://pandas.pydata.org/docs/)
- mglearn : https://libraries.io/pypi/mglearn (https://libraries.io/pypi/mglearn)
- sklearn : https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)

**to proceed with the lab session, refer to this section*

4. Decision tree

In [ ]:
```
import sys
sys.path
```

In [ ]:
```
mglearn.plots.plot_animal_tree()
```

Building decision trees

In [ ]:
```
mglearn.plots.plot_tree_progressive()
```

Controlling complexity of decision trees

In [ ]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stra
tify=cancer.target, random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

In [ ]:
```
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

Analyzing Decision Trees

In [ ]:
```
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["malignant", "benign"], feat
ure_names=cancer.feature_names, impurity=False, filled=True)
```

In [ ]:
```
import graphviz

with open("tree.dot") as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```

Feature Importance in trees

In [ ]:
```
print("Feature importances:")
print(tree.feature_importances_)
```

In [ ]:
```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(np.arange(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances_cancer(tree)
```

In [ ]:
```
tree = mglearn.plots.plot_tree_not_monotone()
display(tree)
```

Exercise 1

In this exercise you will make a decision three with two levels using using the penguins_classification dataset https://inria.github.io/scikit-learn-mooc/python_scripts/trees_dataset.html (https://inria.github.io/scikit-learn-mooc/python_scripts/trees_dataset.html)

1. explore the dataset and try to identify the features and targets 2.load your dataset from a cv file, Link :https://github.com/INRIA/scikit-learn-mooc/blob/main/datasets/penguins_classification.csv (https://github.com/INRIA/scikit-learn-mooc/blob/main/datasets/penguins_classification.csv)
2. split the dataset to training (80%) and testing (20%).
3. plot your two level tree.
4. compute the accuracy of the decision tree on the testing data.

Exercise 2

Using the iris dataset, make a decision tree and Knn model (5 neighbours) (80% training 20% testing), and compare their effectivness using cross validation

Find a better combination of parameters to get a higher cross validation accuracy

5. Random Forest

Building random forests (5 trees)

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=4
2)

forest = RandomForestClassifier(n_estimators=5, random_state=2)
forest.fit(X_train, y_train)
```

Random forest representation

In [ ]:
```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), forest.estimators_)):
    ax.set_title("Tree {}".format(i))
    mglearn.plots.plot_tree_partition(X_train, y_train, tree, ax=ax)

mglearn.plots.plot_2d_separator(forest, X_train, fill=True, ax=axes[-1, -1], alpha=.
4)
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
```

Building random forest (100 trees)

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
```

Importance of random forest features

In [ ]:
```
plot_feature_importances_cancer(forest)
```

Exercice

Using the iris dataset, make a random forest of 5 trees with a maximum depth of 3, then visualise each tree with its accuracy.

analyse the structure of the different trees

6. Logistic regression

Logistic regression for 2 classes

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
import mglearn

#Decision boundaries of a linear SVM and logistic regression on the forge dataset wi
th the default parameters
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))
for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
  clf = model.fit(X, y)
  mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
  ax=ax, alpha=.7)
  mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
  ax.set_title("{}".format(clf.__class__.__name__))
  ax.set_xlabel("Feature 0")
  ax.set_ylabel("Feature 1")
axes[0].legend()
```

In [ ]:
```
# Decision boundaries of a linear SVM on the forge dataset for different values of C
mglearn.plots.plot_linear_svc_regularization()
```

In [ ]:
```
#Let’s analyze LinearLogistic in more detail on the Breast Cancer dataset:
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, stratify=cancer.target, random_state=42)
logreg = LogisticRegression().fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

In [ ]:
```
#Let’s try to increase C to fit a more flexible model:
logreg100 = LogisticRegression(C=100).fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg100.score(X_test, y_test)))
```

Exercise 1

Create an even more regularized model by setting C as 0,01.

What are the results ? How do you interpret them ?

In [ ]:
```
# Coefficients learned by logistic regression on the Breast Cancer dataset for diffe
rent values of C
plt.plot(logreg.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.001")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.legend()
```

In [ ]:
```
# Coefficients learned by logistic regression with L1 penalty on the Breast Cancer d
ataset for different values of C
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
 lr_l1 = LogisticRegression(C=C, penalty="l1", solver="liblinear").fit(X_train, y_tr
ain)
 print("Training accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
 C, lr_l1.score(X_train, y_train)))
 print("Test accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
 C, lr_l1.score(X_test, y_test)))
 plt.plot(lr_l1.coef_.T, marker, label="C={:.3f}".format(C))
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.ylim(-5, 5)
plt.legend(loc=3)
```

Exercise 2

Use logistic regression on the iris dataset.

Logistic regression for 3 classes

In [ ]:
```
#Two-dimensional toy dataset containing three classes
from sklearn.datasets import make_blobs
import mglearn
X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(["Class 0", "Class 1", "Class 2"])
```

In [ ]:
```
#We train a LinearSVC classifier on the dataset
linear_svm = LinearSVC().fit(X, y)
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
```

In [ ]:
```
#Decision boundaries learned by the three one-vs.-rest classifiers
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
 ['b', 'r', 'g']):
 plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
 'Line class 2'], loc=(1.01, 0.3))
```

In [ ]:
```
#Multiclass decision boundaries derived from the three one-vs.-rest classifiers
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
 ['b', 'r', 'g']):
 plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
 'Line class 2'], loc=(1.01, 0.3))
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```