BU.330.775 Machine Learning: Design and Deployment

**Lab 4. Competition with breast cancer dataset**

Learning Goal: practice multiple supervised machine learning approaches on the Diagnostic Wisconsin Breast Cancer Database

Background: This example is curated from Muller and Guido (2016). The Wisconsin breast cancer dataset (<https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic>) includes features that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. The dataset was first introduced in the paper “Nuclear Feature Extraction for Breast Tumor Diagnosis” (<https://www.semanticscholar.org/paper/Nuclear-feature-extraction-for-breast-tumor-Street-Wolberg/53f0fbb425bc14468eb3bf96b2e1d41ba8087f36>) by W. Street, W. Wolberg, and O. Mangasarian in 1993.

1. We start by importing the necessary libraries, including numpy, matplotlib.pyplot, and some tools from scikit-learn for loading datasets and building classifiers. Using load\_breast\_cancer(), we load the breast cancer dataset and examine its descriptive statistics:
   1. There are 569 instances, each with 30 numeric features.



* 1. There are two classes: 'Malignant' and 'Benign'. Among all instances, 212 are malignant and 357 are benign.



This helps us understand the data distribution and prepares us for further modeling steps.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

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

from sklearn.neighbors import KNeighborsClassifier

cancer = load\_breast\_cancer()

print(cancer.DESCR)

1. We will now train and evaluate a K-Nearest Neighbors (KNN) classifier on the breast cancer dataset by testing different values of n\_neighbors (from 1 to 10) to find the optimal balance between training and test accuracy. First, we split the data into training and test sets. Then, for each value of n\_neighbors, we create a KNeighborsClassifier, fit it to the training data, and evaluate it on both the training and test sets, storing the accuracy scores in training\_accuracy and test\_accuracy. Finally, we generate a plot to show the training and test accuracy across different values of n\_neighbors, helping to visualize where the model best generalizes to unseen data.

Note that the scikit-learn implementation of KNN is sophisticated; for example, you can assign weights to neighbors based on their distance. The fit() method is used to preprocess the training set and store the necessary data structures, which helps speed up model evaluation on the test set. More details can be found at <https://scikit-learn.org/1.5/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>. We set a random\_state to ensure reproducible results.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, stratify=cancer.target, random\_state=46

)

training\_accuracy = []

test\_accuracy = []

# try n\_neighbors from 1 to 10

neighbors\_settings = range(1, 11)

for n\_neighbors in neighbors\_settings:

# build the model

knn = KNeighborsClassifier(n\_neighbors=n\_neighbors)

knn.fit(X\_train, y\_train)

# record training set accuracy

training\_accuracy.append(knn.score(X\_train, y\_train))

# record generalization accuracy

test\_accuracy.append(knn.score(X\_test, y\_test))

plt.plot(neighbors\_settings, training\_accuracy, label="training accuracy")

plt.plot(neighbors\_settings, test\_accuracy, label="test accuracy")

plt.ylabel("Accuracy")

plt.xlabel("n\_neighbors")

plt.legend()

plt.show()

1. Let's examine the confusion matrix using 6 neighbors. You are always welcome to use prompts to generate it with AI.

# prompt: generate confusion matrix when n\_neighbors is 6

from sklearn.metrics import confusion\_matrix

# Build the model with n\_neighbors=6

knn = KNeighborsClassifier(n\_neighbors=6)

knn.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = knn.predict(X\_test)

# Generate the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

cm

1. Now, we’ll use some code to train a Decision Tree Classifier on the breast cancer dataset with a maximum tree depth of 4 to prevent overfitting. After training, the code evaluates the classifier’s accuracy on both the training and test sets.

from sklearn.tree import DecisionTreeClassifier

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)))

1. Examine the confusion matrix again.

# prompt: generate confusion matrix on tree

# Make predictions on the test set for the decision tree

y\_pred\_tree = tree.predict(X\_test)

# Generate the confusion matrix for the decision tree

cm\_tree = confusion\_matrix(y\_test, y\_pred\_tree)

print("Confusion Matrix for Decision Tree:")

cm\_tree

1. Next, we’ll use some code to visualize the previously trained Decision Tree Classifier. It first uses export\_graphviz from sklearn.tree to export the tree structure to a file named tree.dot, including labels for the classes (malignant, benign) and feature names from the breast cancer dataset. The parameters impurity=False and filled=True are used to simplify the visualization by hiding impurity values and adding color to indicate class labels.

Next, it imports graphviz and reads the contents of tree.dot. The graphviz.Source function is then used to display the tree structure within the notebook, allowing a visual understanding of how the decision tree makes classifications based on feature values. This visualization helps interpret the model’s decision-making process at each node.

from sklearn.tree import export\_graphviz

export\_graphviz(tree, out\_file="tree.dot", class\_names=["malignant", "benign"],

feature\_names=cancer.feature\_names, impurity=False, filled=True)

import graphviz

with open("tree.dot") as f:

dot\_graph = f.read()

display(graphviz.Source(dot\_graph))

1. Lastly, we’ll use some code to train a Gradient Boosting Classifier on the breast cancer dataset to evaluate its performance. The model is trained with a max\_depth of 1 to prevent overfitting.

from sklearn.ensemble import GradientBoostingClassifier

gbrt = GradientBoostingClassifier(random\_state=0, max\_depth=1)

gbrt.fit(X\_train, y\_train)

print("Accuracy on training set: {:.3f}".format(gbrt.score(X\_train, y\_train)))

print("Accuracy on test set: {:.3f}".format(gbrt.score(X\_test, y\_test)))

1. Examine the confusion matrix.

# prompt: generate confusion matrix of gbrt

# Make predictions on the test set using the trained GBTR model

y\_pred\_gbrt = gbrt.predict(X\_test)

# Generate the confusion matrix for the GBTR model

cm\_gbrt = confusion\_matrix(y\_test, y\_pred\_gbrt)

print("Confusion Matrix for Gradient Boosted Regression Trees:")

cm\_gbrt

**Homework Question 1 (2pt):** In a text cell, compare the performance of the three models: (1) KNN with K=6; (2) Decision Tree; (3) Gradient Boosting.

**Competition:** Use the breast cancer dataset and any supervised machine learning models of your choice:

1. Linear models: <https://scikit-learn.org/1.5/api/sklearn.linear_model.html>
2. Ensemble: <https://scikit-learn.org/1.5/api/sklearn.ensemble.html>

You have 30 minutes to prepare your notebook and presentation addressing the following perspectives:

1. Pre-model thinking: Why you chose the models and why they are appropriate for the problem.
2. Model explanation: Explain your data preprocessing and modeling approach.
3. After-model interpretation: Evaluate your model's performance.

Each team (2-3 students) will have 3 minutes for their presentation, which will be scored by the audience, and awards will be announced.

**Criteria**: Both model performance (30%) and presentation quality (70%).

**Homework Question 2 (8 pts):** Your group code work and your individual work on the three perspectives.

**Submission:** Complete and submit on Canvas by the beginning of Class 5. Use homework4\_yourname.ipynb as the file name.