BU.330.775 Machine Learning: Design and Deployment

**Lab 3. Model evaluation on MNIST dataset**

Learning Goal: evaluate multiple supervised machine learning approaches on the MNIST database

Background: This example is curated from Geron (2022). The MNIST dataset (<http://yann.lecun.com/exdb/mnist/>), downloaded from openml.org, consists of handwritten digits with 784 features. This famous dataset was contributed by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. In the AI Essentials for Business course, you will use deep learning models to classify this dataset. The purpose of the exercise in this course is to practice evaluating simplified machine learning tasks.



1. First, we set up default font sizes for various components in plots to improve clarity and aesthetics.

import matplotlib.pyplot as plt

plt.rc('font', size=14)

plt.rc('axes', labelsize=14, titlesize=14)

plt.rc('legend', fontsize=14)

plt.rc('xtick', labelsize=10)

plt.rc('ytick', labelsize=10)

1. We load and explore the MNIST dataset, a well-known dataset of handwritten digits commonly used for machine learning tasks, particularly in classification. Here as\_frame indicates whether the data is NumPy arrays or pandas DataFrame.



from sklearn.datasets import fetch\_openml



mnist = fetch\_openml('mnist\_784', as\_frame=False)

X, y = mnist.data, mnist.target

print(X.shape)

print(y.shape)

y

1. We define a function to display individual digit images and then use it to create a grid of the first 100 images in the dataset. Here’s how it works:

def plot\_digit(image\_data):

image = image\_data.reshape(28, 28)

plt.imshow(image, cmap="binary")

plt.axis("off")

plt.figure(figsize=(9, 9))

for idx, image\_data in enumerate(X[:100]):

plt.subplot(10, 10, idx + 1)



plot\_digit(image\_data)

plt.subplots\_adjust(wspace=0, hspace=0)

plt.show()



1. We will use the first 60,000 images as the training set and the remaining 10,000 as the testing set.

X\_train, X\_test, y\_train, y\_test = X[:60000], X[60000:], y[:60000], y[60000:]

1. First, let’s train a binary classifier to distinguish whether the image is the digit 2 or not. We will use stochastic gradient descent classifier <https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.SGDClassifier.html>. We control the random\_state here so we will get the same results.



y\_train\_2 = (y\_train == '2')

y\_test\_2 = (y\_test == '2')

from sklearn.linear\_model import SGDClassifier



sgd\_clf = SGDClassifier(random\_state=46)



sgd\_clf.fit(X\_train, y\_train\_2)



1. Now, we are evaluating the classifier's performance using 3-fold cross-validation to measure accuracy.

from sklearn.model\_selection import cross\_val\_score

cross\_val\_score(sgd\_clf, X\_train, y\_train\_2, cv=3, scoring="accuracy")

1. Next, we generate predictions on test set and evaluate them using a confusion matrix to understand the performance of the binary classifier on the testing data.

from sklearn.metrics import confusion\_matrix

y\_test\_2\_pred = sgd\_clf.predict(X\_test)

cm = confusion\_matrix(y\_test\_2, y\_test\_2\_pred)

cm

**Homework Question 1 (3pt):** Write Python code using sklearn.metrics to obtain (1) precision, (2) recall, and (3) f1-score. Then validate the value **using the formulas** from the lecture notes. You can do the calculation either with Python code or by using mathematics in a Text cell.

1. Now, we explore the precision-recall trade-off using decision scores from the classifier and plot the curve.

y\_scores = cross\_val\_predict(sgd\_clf, X\_train, y\_train\_2, cv=3,

method="decision\_function")

from sklearn.metrics import precision\_recall\_curve

precisions, recalls, thresholds = precision\_recall\_curve(y\_train\_2, y\_scores)

plt.figure(figsize=(8, 4)) # comment: it's just formatting

plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)

plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)

plt.axis([-20000, 20000, 0, 1])

plt.grid()

plt.xlabel("Threshold")

plt.legend(loc="center right")

plt.show()



1. Next, we create an ROC curve to visualize the performance of the classifier.



from sklearn.metrics import roc\_curve

fpr, tpr, thresholds = roc\_curve(y\_train\_2, y\_scores)

plt.figure(figsize=(6, 5))

plt.plot(fpr, tpr, linewidth=2, label="ROC curve")

plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")

plt.xlabel('false positive rate')

plt.ylabel('true positive rate')

plt.grid()

plt.axis([0, 1, 0, 1])

plt.legend(loc="lower right", fontsize=13)

plt.show()

1. Calculate the AUC score.

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score(y\_train\_2, y\_scores)

1. Now, Let’s try multiclass classification. First, scale the features using MinMaxScaler. This will help speed up our machine learning algorithm. Remember to apply the same processing to both the training and testing sets.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train.astype("float64"))

X\_test\_scaled = scaler.fit\_transform(X\_test.astype("float64"))

1. Perform error analysis on the multiclass classification model by generating a confusion matrix, which provides a detailed view of how the model’s predictions compare with the actual labels.

from sklearn.metrics import ConfusionMatrixDisplay

sgd\_clf.fit(X\_train\_scaled, y\_train)

y\_test\_pred = sgd\_clf.predict(X\_test\_scaled)

plt.rc('font', size=9)

ConfusionMatrixDisplay.from\_predictions(y\_test, y\_test\_pred)

plt.show()

1. Next, display a normalized confusion matrix to show the classification performance for each class as percentages.

plt.rc('font', size=10)

ConfusionMatrixDisplay.from\_predictions(y\_test, y\_test\_pred,

normalize="true", values\_format=".0%")

plt.show()

1. Lastly, we visualize a weighted confusion matrix for error analysis, focusing only on the misclassified instances, normalized by row.

sample\_weight = (y\_test\_pred != y\_test)

plt.rc('font', size=10)

ConfusionMatrixDisplay.from\_predictions(y\_test, y\_test\_pred,

sample\_weight=sample\_weight,

normalize="true", values\_format=".0%")

plt.show()

**Homework Question 2 (7pt):** Use RidgeClassifier (<https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.RidgeClassifier.html>) instead of SGDClassifier for multiclass classification. Feel free to adjust the hyperparameters.

Generate a confusion matrix in any of the three styles of your choice. **Evaluate** the performance of the Ridge classifier compared to the SGD classifier as thoroughly as possible. Your submission will be graded based on comprehensiveness.

**Submission**: Complete all the lab steps and homework questions. Save your file as homework3\_yourname.ipynb and submit on Canvas by the beginning of class 4.