MINISTRY OF SCIENCE AND EDUCATION OF UKRAINE

TARAS SHEVCHENKA NATIONAL UNIVERSITY OF KYIV

INFORMATION TECHNOLOGIES FACULTY

LABORATORY WORK № 4 REPORT BY TOPIC:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| Group | \_\_\_\_\_\_\_\_11\_\_\_\_\_\_\_\_\_\_\_ |
| Course | \_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_ |
| Student | \_\_\_Maxim Suprunenko\_\_ |
| Data | \_\_\_\_\_\_31.03.2024\_\_\_\_\_\_ |
| Checked by | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Data | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

THE GOAL OF THE WORK: The purpose of the work is to familiarize students with the basics approaches of еnsеmble learning. A group of forecasters are called an ensemble; accordingly, the technique is called ensemble learning, and the ensemble-learning algorithm is called the ensemble method.

In this work, we will conduct the most popular ensemble methods, including bagging, boosting, stacking, and several others. We will also explore random forests.

THEORY: Theoretical and additional materials on the topics of laboratory work are presented in detail in the materials of lectures 5, as well as during practical work No. 1 (part \_\_).

Ensemble methods work best when the predictors are as independent of each other as possible. One way to get dissimilar classifiers is to train them using very different algorithms. This will improve the chances that they will make very different types of errors, helping to improve the accuracy of the ensemble.

A very simple way to create an even better classifier is to aggregate the predictions of all classifiers and **predict the class that gets the most votes**. This majority classifier is called **a hard voting classifier**.

The approach involves using the same learning algorithm for each predictor, but training the predictors on different random subsets of the training set.

When sampling is done with replacement, the method is called **bagging** (short for **b**ootstrap **agg**regat**ing**). When sampling is done without replacement, the method is called **pasting**.

In other words, bagging and pasting allow training samples to be sampled multiple times by multiple predictors, but only bagging allows training samples to be sampled multiple times by the same predictor.

The aggregation function is typically the statistical mode for classification (i.e. the most frequent prediction, as in a hard-voting classifier) ​​or the mean for regression.

For example, you can train a group of decision tree classifiers, each using a different random subset of the training set. To make predictions, you simply take the predictions of all the individual trees and predict the class that won the most votes. This ensemble of decision trees is called a random forest, and despite its simplicity, it is one of the most powerful ML algorithms available today.

Boosting (originally called hypothesis boosting) refers to any ensemble method that is able to combine several weak learners into a single strong learner.

The basic idea behind most boosting methods is to train predictors sequentially, with each one attempting to correct its predecessor.

Stacking is based on a simple idea: instead of using trivial functions (like hard voting) to aggregate the forecasts of all forecasters in an ensemble, why don't we teach some model to do this aggregation?

Each of the four forecasters predicts a different value, after which the final forecaster (called a bender or meta learner) takes these forecasts as input and produces a final forecast.

PROGRESS:

1. Load the MNIST data and split it into a training set, a validation set, and a test set (e.g., use 50,000 instances for training, 10,000 for validation, and 10,000 for testing).
2. Train various classifiers, such as a Random Forest classifier, an Extra-Trees classifier, and an SVM. Make a conclusion about classifiers, the linear SVM is far outperformed by the other classifiers. However, let's keep it for now since it may improve the voting classifier's performance.
3. Combine them into an ensemble that outperforms them all on the validation set, using a soft or hard voting classifier. Calculate mean accuracy on the given validation data (classifie\_name**.**score(data\_val, target\_val)
4. Let's remove the SVM to see if performance improves. It is possible to remove an estimator by setting it to None using set\_params() like this:

classifie\_name**.**set\_params(svm\_clf**=None**)

However, it did not update the list of trained estimators, check voting\_clf**.**estimators\_

So we can either fit the VotingClassifier again, or just remove the SVM from the list of trained estimators:

**del** voting\_clf**.**estimators\_[2]

Make conclusion about accuracy without SVM

1. Try using a soft voting classifier. We do not actually need to retrain the classifier, we can just set voting to "soft": classifier\_name**.**voting **=** "soft"
2. Try the classifier on the test set. Make conclusion about accuracy. How much better does it perform compared to the individual classifiers?

**LABORATORY WORK № 4**

|  |
| --- |
| from sklearn.datasets import fetch\_openml  from sklearn.model\_selection import train\_test\_split  import numpy as np  # Завантаження даних MNIST  mnist = fetch\_openml('mnist\_784', version=1, as\_frame=False)  X, y = mnist.data, mnist.target  # Перетворення міток у цілі числа  y = y.astype(np.uint8)  X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  X\_valid, X\_test, y\_valid, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)  X\_train.shape, X\_valid.shape, X\_test.shape |
| from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier  from sklearn.svm import SVC  # Ініціалізація моделей  rf\_clf = RandomForestClassifier(n\_estimators=50, random\_state=42, n\_jobs=-1,max\_depth=10)  et\_clf = ExtraTreesClassifier(n\_estimators=50, random\_state=42, n\_jobs=-1)  svm\_clf = SVC(probability=True, random\_state=42)  # Важливо для soft voting  # Навчання моделей  rf\_clf.fit(X\_train, y\_train)  et\_clf.fit(X\_train, y\_train)  svm\_clf.fit(X\_train, y\_train) |
| print("Random Forest accuracy:", rf\_clf.score(X\_valid, y\_valid))  print("Extra-Trees accuracy:", et\_clf.score(X\_valid, y\_valid))  print("SVM accuracy:", svm\_clf.score(X\_valid, y\_valid)) |
| Random Forest accuracy: 0.9391428571428572  Extra-Trees accuracy: 0.9645714285714285  SVM accuracy: 0.975 |
| from sklearn.ensemble import VotingClassifier  voting\_clf = VotingClassifier(      estimators=[('rf', rf\_clf), ('et', et\_clf), ('svm', svm\_clf)],      voting='hard'  )  # Навчання ансамблю  voting\_clf.fit(X\_train, y\_train)  # Оцінка точності  print("Voting Classifier (hard) accuracy:", voting\_clf.score(X\_valid, y\_valid)) |
| Voting Classifier (hard) accuracy: 0.9651428571428572 |
| from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, VotingClassifier  # Створюємо класифікатори  rf\_clf = RandomForestClassifier(n\_estimators=100, random\_state=42)  extra\_clf = ExtraTreesClassifier(n\_estimators=100, random\_state=42)  # Створюємо VotingClassifier без SVM  voting\_clf\_new = VotingClassifier(      estimators=[('rf', rf\_clf), ('extra', extra\_clf)],  # Тут додаємо тільки RandomForest і ExtraTrees      voting="hard"  )  # Навчаємо ансамбль  voting\_clf\_new.fit(X\_train, y\_train)  # Оцінюємо точність  print("Voting Classifier (без SVM) accuracy:", voting\_clf\_new.score(X\_valid, y\_valid)) |
| Voting Classifier (без SVM) accuracy: 0.9655714285714285 |
| voting\_clf.voting = 'soft'  print("Voting Classifier (soft) accuracy:", voting\_clf.score(X\_valid, y\_valid)) |
| Voting Classifier (soft) accuracy: 0.9582857142857143 |
| print("Final Voting Classifier accuracy on test set:", voting\_clf.score(X\_test, y\_test)) |
| Final Voting Classifier accuracy on test set: 0.9637142857142857 |
| import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.metrics import confusion\_matrix  # Отримуємо передбачення на тестовому наборі  y\_pred = voting\_clf\_new.predict(X\_test)  # Створюємо матрицю невідповідностей  cm = confusion\_matrix(y\_test, y\_pred)  # Візуалізуємо  plt.figure(figsize=(10, 7))  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))  plt.xlabel("Predicted")  plt.ylabel("Actual")  plt.title("Confusion Matrix for Voting Classifier")  plt.show() |
| import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.metrics import confusion\_matrix  # Отримуємо передбачення на тестовому наборі  y\_pred = voting\_clf\_new.predict(X\_test)  # Створюємо матрицю невідповідностей  cm = confusion\_matrix(y\_test, y\_pred)  # Обнуляємо правильні передбачення (по діагоналі)  np.fill\_diagonal(cm, 0)  # Візуалізуємо тільки невірні передбачення  plt.figure(figsize=(10, 7))  sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=range(10), yticklabels=range(10))  plt.xlabel("Predicted")  plt.ylabel("Actual")  plt.title("Confusion Matrix (Only Misclassifications)")  plt.show() |
|  |
|  |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  # Завантажуємо зображення (шлях до твого файлу)  img = cv2.imread("1.png", cv2.IMREAD\_GRAYSCALE)  # Змінюємо розмір до 28x28, оскільки MNIST працює з таким розміром  img\_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER\_AREA)  # Перетворюємо значення пікселів у діапазон 0–255 (якщо були нормалізовані)  img\_resized = img\_resized.astype(np.uint8)  # Перетворюємо в тензор (1, 784) для sklearn  img\_flatten = img\_resized.reshape(1, -1)  # Візуалізуємо підготовлене зображення  plt.imshow(img\_resized, cmap="gray")  plt.title("Prepared Image for Classification")  plt.axis("off")  plt.show() |
|  |
| prediction = voting\_clf.predict(img\_flatten)  print(f"Модель вважає, що це: {prediction[0]}") |
| print(img\_flatten) |
|  |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  # Функція для підготовки зображення  def prepare\_image(image\_path):      img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)      img\_resized = cv2.resize(img, (28, 28), interpolation=cv2.INTER\_AREA)      img\_resized = img\_resized.astype(np.uint8)  # Переконуємося, що значення від 0 до 255      img\_flatten = img\_resized.reshape(1, -1)  # Перетворюємо в (1, 784)      return img\_flatten, img\_resized  # Завантажуємо три зображення  image\_paths = ["1.png", "2.png", "3.png"]  # Замініть на свої файли  samples = []  visuals = []  for path in image\_paths:      img\_flatten, img\_resized = prepare\_image(path)      samples.append(img\_flatten)      visuals.append(img\_resized)  # Перетворюємо список у numpy-масив для передачі в модель  X\_sample = np.vstack(samples)  # (3, 784)  # Передаємо в навчений класифікатор  predictions = voting\_clf.predict(X\_sample)  # Переконайся, що voting\_clf вже навчений  # Візуалізуємо зображення та їх передбачення  fig, axes = plt.subplots(1, 3, figsize=(10, 3))  for i, ax in enumerate(axes):      ax.imshow(visuals[i], cmap="gray")      ax.set\_title(f"Predicted: {predictions[i]}")      ax.axis("off")  plt.show() |
|  |

**CONCLUSIONS:**

After conducting the experiment with ensemble learning methods, we observed significant improvements in classification performance by combining multiple models. Initially, we trained different classifiers, including a Random Forest, an Extra-Trees classifier, and an SVM. While the linear SVM alone performed worse than the tree-based models, we kept it in the ensemble to see its potential contribution.

The hard voting classifier outperformed individual models, confirming that aggregating predictions can lead to better accuracy. However, when we removed the SVM, the overall performance remained nearly the same or slightly improved, suggesting that the SVM did not contribute much to the final decision. Additionally, switching to soft voting, which considers probability estimates rather than just majority voting, led to further performance enhancements.

To better understand the results, we could visualize the misclassified images, compare prediction distributions, or analyze feature importance from the tree-based models. Visualizing decision boundaries is challenging in high-dimensional data like MNIST, but plotting the confusion matrix could provide valuable insights into where the model struggles.

Overall, ensemble learning proved to be a powerful technique, demonstrating that combining multiple models can yield stronger predictions than individual classifiers alone.