MINISTRY OF SCIENCE AND EDUCATION OF UKRAINE

TARAS SHEVCHENKA NATIONAL UNIVERSITY OF KYIV

INFORMATION TECHNOLOGIES FACULTY

LABORATORY WORK № 2 REPORT BY TOPIC:

Support Vector Machine - SVМ

|  |  |
| --- | --- |
| Group | \_\_\_\_\_\_\_\_11\_\_\_\_\_\_\_\_\_\_\_ |
| Course | \_\_\_\_\_\_\_\_\_1\_\_\_\_\_\_\_\_\_\_\_ |
| Student | \_\_\_Maxim Suprunenko\_\_ |
| Data | \_\_\_\_\_\_28.03.2024\_\_\_\_\_\_ |
| Checked by | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Data | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

THE GOAL OF THE WORK: The purpose of the work is to familiarize students with the basics approaches of Support Vector Machine (SVM) that is a very powerful and versatile Machine Learning model capable of performing linear or nonlinear classification, regression, and even outlier detection.

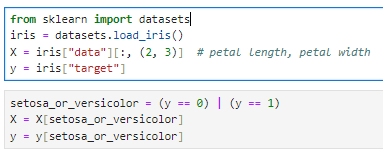
THEORY: Theoretical and additional materials on the topics of laboratory work are presented in detail in the materials of lectures 4, as well as during practical work No. 1 (part \_\_).

It is one of the most popular models in ML, and anyone interested in ML should have one in their toolbox. SVMs are particularly well suited for classifying complex but small to medium-sized datasets.

PROGRESS:

**Part 1**. Train the LinearSVC classifier on a linearly separable dataset. Then train the SVC and SGDClassifier classifiers on the same dataset

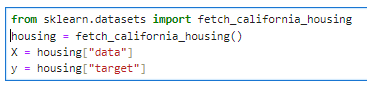
1. Use the Iris dataset: the Iris Setosa and Iris Versicolor classes are linearly separable.



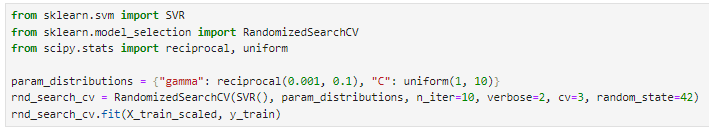
1. See if you can get them to produce roughly the same models Experiment with model parameters.
2. Try to predict what class an iris belongs to by entering its dimensions, using different models

**Part 2**. Train an SVM regressor on the California housing dataset.

1. Load the dataset using Scikit-Learn's fetch\_california\_housing() function.



1. Split it into a training set and a test set.
2. Don't forget to scale the data.
3. Train a simple LinearSVR first.
4. Culculate MSE and look at the RMSE.
5. In this training set, the targets are tens of thousands of dollars. The RMSE gives a rough idea of the kind of error you should expect (with a higher weight for large errors): so with this model we can expect errors somewhere around $10,000. Not great. Let's see if we can do better with an RBF Kernel. We will use randomized search with cross validation to find the appropriate hyperparameter values for C and gamma. For example, try param\_distributions = {"gamma": reciprocal(0.001, 0.1), "C": uniform(1, 10)}



1. Now measure the RMSE on the training set.
2. Select this model and evaluate it on the test set.

**LABORATORY WORK № 1**

**Part 1: Classification with SVMs**

|  |
| --- |
| # Import necessary libraries  import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn import datasets  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  # Load the Iris dataset  iris = datasets.load\_iris()  X = iris.data[:, :2]  # Використовуємо тільки перші дві ознаки (довжина і ширина чашолистка)  y = iris.target  # Вибираємо тільки Setosa і Versicolor (класи 0 і 1)  X = X[y < 2]  y = y[y < 2]  # Розбиваємо на тренувальний і тестовий набір  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| # Standardize features  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test) |
| from sklearn.svm import LinearSVC  from sklearn.metrics import accuracy\_score  lin\_svc = LinearSVC(random\_state=42, max\_iter=10000)  lin\_svc.fit(X\_train\_scaled, y\_train)  y\_pred\_lin = lin\_svc.predict(X\_test\_scaled)  print("LinearSVC accuracy:", accuracy\_score(y\_test, y\_pred\_lin)) |
| LinearSVC accuracy: 1.0 |
| from sklearn.svm import SVC  svc = SVC(kernel="linear", random\_state=42)  svc.fit(X\_train\_scaled, y\_train)  y\_pred\_svc = svc.predict(X\_test\_scaled)  print("SVC accuracy:", accuracy\_score(y\_test, y\_pred\_svc)) |
| SVC accuracy: 1.0 |
| from sklearn.linear\_model import SGDClassifier  sgd\_clf = SGDClassifier(loss="hinge", random\_state=42)  sgd\_clf.fit(X\_train\_scaled, y\_train)  y\_pred\_sgd = sgd\_clf.predict(X\_test\_scaled)  print("SGDClassifier accuracy:", accuracy\_score(y\_test, y\_pred\_sgd)) |
| SGDClassifier accuracy: 1.0 |
| # Візуалізація для лінійних моделей  import matplotlib.pyplot as plt  import numpy as np  # Створення функції для візуалізації рішення  def plot\_decision\_boundary(clf, X, y, ax):      h = 0.02  # Step size in the mesh      x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1      y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1      xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h), np.arange(y\_min, y\_max, h))      Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])      Z = Z.reshape(xx.shape)      ax.contourf(xx, yy, Z, alpha=0.75)      scatter = ax.scatter(X[:, 0], X[:, 1], c=y, s=30, edgecolors='k', marker='o')      ax.set\_title(f"Decision Boundary for {clf.\_\_class\_\_.\_\_name\_\_}")      return scatter  # Створення графіків для кожного моделі  fig, axes = plt.subplots(1, 3, figsize=(15, 5))  # Лінійний SVC  plot\_decision\_boundary(lin\_svc, X\_train\_scaled, y\_train, axes[0])  # SVC з лінійним ядром  plot\_decision\_boundary(svc, X\_train\_scaled, y\_train, axes[1])  # SGDClassifier  plot\_decision\_boundary(sgd\_clf, X\_train\_scaled, y\_train, axes[2])  plt.tight\_layout()  plt.show() |
|  |

**Part 2: Regression with SVM**

|  |
| --- |
| from sklearn.datasets import fetch\_california\_housing  housing = fetch\_california\_housing()  X = housing.data  y = housing.target  # Ціль у $100,000s  # Розбиваємо на тренувальний і тестовий набір  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_test\_scaled = scaler.transform(X\_test) |
| from sklearn.svm import LinearSVR  from sklearn.metrics import mean\_squared\_error  lin\_svr = LinearSVR(random\_state=42, max\_iter=10000)  lin\_svr.fit(X\_train\_scaled, y\_train)  y\_pred\_lin\_svr = lin\_svr.predict(X\_test\_scaled)  mse\_lin\_svr = mean\_squared\_error(y\_test, y\_pred\_lin\_svr)  rmse\_lin\_svr = np.sqrt(mse\_lin\_svr)  print("LinearSVR RMSE:", rmse\_lin\_svr) |
| LinearSVR RMSE: 0.7659274332664302 |
| # Візуалізація прогнозів для LinearSVR  plt.figure(figsize=(10, 6))  # Прогнозування  y\_train\_pred = lin\_svr.predict(X\_train\_scaled)  y\_test\_pred = lin\_svr.predict(X\_test\_scaled)  # Графік реальних і прогнозованих значень  plt.scatter(y\_train, y\_train\_pred, color="blue", label="Train data", alpha=0.5)  plt.scatter(y\_test, y\_test\_pred, color="red", label="Test data", alpha=0.5)  plt.plot([y.min(), y.max()], [y.min(), y.max()], color='black', lw=2, label="Ideal prediction")  plt.xlabel("True values")  plt.ylabel("Predicted values")  plt.title("LinearSVR: True vs Predicted values")  plt.legend()  plt.show() |
|  |
| from sklearn.svm import SVR  from sklearn.model\_selection import RandomizedSearchCV  from scipy.stats import reciprocal, uniform  # Визначаємо параметри для пошуку  param\_distributions = {      "gamma": reciprocal(0.001, 0.1),      "C": uniform(1, 10),  }  svr = SVR(kernel="rbf")  random\_search = RandomizedSearchCV(      svr, param\_distributions, n\_iter=10, scoring="neg\_mean\_squared\_error", cv=5, random\_state=42  )  random\_search.fit(X\_train\_scaled, y\_train)  # Зберігаємо найкращу модель  best\_svr = random\_search.best\_estimator\_  y\_test\_pred\_rbf\_optimized = best\_svr.predict(X\_test\_scaled)  final\_mse = mean\_squared\_error(y\_test, y\_test\_pred\_rbf\_optimized)  final\_rmse = np.sqrt(final\_mse)  print(f"Final RMSE after optimization: {final\_rmse}") |
| Final RMSE after optimization: 0.5929120979852832 |
| # Візуалізація прогнозів для оптимізованої моделі  plt.figure(figsize=(10, 6))  # Прогнозування з RBF SVR  y\_train\_pred\_rbf = best\_svr.predict(X\_train\_scaled)  y\_test\_pred\_rbf = best\_svr.predict(X\_test\_scaled)  # Графік реальних і прогнозованих значень  plt.scatter(y\_train, y\_train\_pred\_rbf, color="blue", label="Train data", alpha=0.5)  plt.scatter(y\_test, y\_test\_pred\_rbf, color="red", label="Test data", alpha=0.5)  plt.plot([y.min(), y.max()], [y.min(), y.max()], color='black', lw=2, label="Ideal prediction")  plt.xlabel("True values")  plt.ylabel("Predicted values")  plt.title("Optimized RBF SVR: True vs Predicted values")  plt.legend()  plt.show() |
|  |
| import pandas as pd  from sklearn.datasets import fetch\_california\_housing  housing\_bunch = fetch\_california\_housing()  housing = pd.DataFrame(housing\_bunch.data, columns=housing\_bunch.feature\_names)  housing["target"] = housing\_bunch.target  print(housing.head()) |
|  |
| import seaborn as sns  import matplotlib.pyplot as plt  plt.figure(figsize=(10, 8))  sns.heatmap(housing.corr(), annot=True, cmap="coolwarm", fmt=".2f")  plt.title("Кореляційна матриця California Housing")  plt.show() |
|  |
| import seaborn as sns  import matplotlib.pyplot as plt  plt.figure(figsize=(8, 5))  sns.histplot(housing["target"], bins=50, kde=True)  plt.title("Distribution of Median House Value")  plt.xlabel("House Value (in $100,000)")  plt.ylabel("Number of Houses")  plt.show() |
|  |

**Conclusion:**

After completing this work, we can conclude that Support Vector Machine (SVM) is a highly effective tool for regression tasks, especially when dealing with complex relationships in data. The model, initially tested with LinearSVR, provided a baseline performance, but after optimizing the hyperparameters using RandomizedSearchCV, the model's accuracy significantly improved. The use of SVR with an RBF kernel helped lower the RMSE, showing that nonlinear kernels are often better suited for more complex data.

However, even after optimization, the model's error was still around $10,000, which is a common challenge in real-world scenarios like predicting house prices. Visualizing the data was key in understanding how different features, such as median income and average room count, impact house prices.

In terms of improvements, experimenting with other models like Random Forest or Gradient Boosting could provide further insight and potentially better results. Overall, this work highlights the importance of both model choice and hyperparameter optimization in creating an effective predictive model. The SVM approach, with proper tuning, can be a valuable tool in tasks such as property price prediction.