Artificial Intelligence for Cybersecurity
Experiment 4
Regression and hyperparameter tuning

Regression

Regression is a type of supervised machine-learning technique used for predicting continuous numerical values. In regression, the aim is to establish a relationship between input variables (also known as predictors or independent variables) and a continuous output variable (also known as the dependent variable or response variable).

In the context of cybersecurity, regression techniques can be applied to a variety of tasks, such as predicting the number of cyberattacks on a system, estimating the time it takes to detect and respond to a security breach, or forecasting the likelihood of a successful phishing attack.

Another example is using regression to predict the severity of a security incident based on various input features, such as the type of attack, the affected system, and the time of day. This information can help security teams respond quickly and appropriately to mitigate the impact of the incident.

Overall, regression techniques are an important tool for analysing and predicting cybersecurity threats and can help organisations improve their overall security posture.

Hyperparameter tuning

In machine learning, hyperparameters are the settings or parameters of a model that are not learned from the training data but rather are set before training the model. These settings are set by the user or developer and can significantly affect the performance of the model.

Examples of hyperparameters include regularisation parameters, learning rate, the number of hidden layers in a neural network, the number of trees in a random forest, and the kernel function in a support vector machine.

Tuning hyperparameters is an essential step in building a machine learning model as it helps to optimise the model's performance on a particular task. There are several techniques available for hyperparameter tuning, such as Grid Search, Random Search, Bayesian Optimization, and Evolutionary Algorithms.

GridSearchCV performs an exhaustive search over a specified hyperparameter grid to determine the best set of hyperparameters for a machine-learning model. It works by fitting the model on all possible combinations of hyperparameters provided in a grid and selecting the best hyperparameters based on the evaluation score provided.

GridSearchCV uses k-fold cross-validation to evaluate each combination of hyperparameters. This technique helps to prevent overfitting and provides a more accurate estimate of the model's performance on unseen data.

Cross-validation is a technique used to evaluate the performance of a machine-learning model by using multiple splits of the dataset. In k-fold cross-validation, the dataset is divided into k equally sized folds. The model is trained on k-1 folds and evaluated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The results of each fold are then averaged to obtain a single estimate of the model's performance.

Example Implementations

Linear regression

Linear regression is a statistical method used to model the relationship between two variables by fitting a linear equation to the observed data. The goal of linear regression is to find the best-fitting line through the data, which can then be used to predict the value of one variable based on the value of the other. It is widely used in fields such as economics, engineering, and social sciences to study and make predictions about the relationships between variables.

The California Housing dataset is a popular dataset provided by the Scikit-learn library for machine learning. It contains information on the median house value, as well as other features such as the total number of rooms, bedrooms, population, households, and geographic coordinates for various locations in California. It contains 20,640 instances. The dataset is often used in regression tasks, where the goal is to predict the median house value based on the other available features.

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Load the dataset
data = fetch_california_housing()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,
test_size=0.2, random_state=42)

# Create a linear regression model and train it on the training data
model = LinearRegression()
model.fit(X_train, y_train)
```

```
# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the performance of the model using mean squared error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error: ", mse)

# Printing the learned parameters
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Plot the actual and predicted values
chosen_X = X_test[:, 0]
plt.scatter(chosen_X, y_test, label="Actual")
plt.scatter(chosen_X, y_pred, label="Predicted")
plt.xlabel("X[0]")
plt.ylabel("y")
plt.legend()
```

In this program, we first load the California Housing dataset using the *fetch_california_housing* function from Scikit-learn. We then split the data into training and testing sets using the *train_test_split* function.

Next, we create a *LinearRegression* model and train it on the training data using the fit method. We then use the trained model to make predictions on the testing data using the *predict* method.

After that, we evaluate the performance of the model using the mean squared error (MSE) metric, which measures the average squared difference between the predicted and actual values. We use the *mean_squared_error* function from Scikit-learn to calculate the MSE and print it out.

Finally, we create a scatter plot of the actual and predicted values. If the predicted values follow the actual data closely, it indicates that the linear regression model is a good fit for the data.

Support vector regression with GridSearchCV

Support Vector Machine (SVM), a powerful and popular supervised machine learning algorithm, works by finding the best possible line or hyperplane that can separate the data points into different classes while also maximizing the margin or distance between the two closest data points of different classes. SVM is particularly useful for dealing with complex and non-linear datasets, as it can handle both linear and non-linear classification problems by using different kernel functions.

First, let's import the necessary libraries and load the dataset:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

california = fetch_california_housing()
X = california.data
y = california.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Next, we'll define our model and the hyperparameters we want to tune. We'll tune the hyperparameters using GridSearchCV. GridSearchCV maximises its score. However, MSE is an error metric to be minimised, so *neg_mean_squared_error* is used as scoring.

```
model = SVR()
params = {'kernel': ['rbf', 'sigmoid'], 'C': [0.1, 1]}
grid = GridSearchCV(estimator=model, param_grid=params, cv=5,
scoring='neg_mean_squared_error', verbose=3)
```

Now we'll fit the model to the training data using GridSearchCV:

```
grid.fit(X_train, y_train)
```

We can then print out the best hyperparameters found by GridSearchCV:

```
print("Best hyperparameters: ", grid.best_params_)
```

Next, we'll use the best hyperparameters to make predictions on the test data and evaluate the performance of the model using multiple metrics:

```
y_pred = grid.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("MAE:", mae)
print("R^2:", r2)
```

This will output the mean squared error, root mean squared error, mean absolute error, and R-squared value for the model. MSE, RMSE, and MAE are used to measure the difference between the predicted and actual values, while R-squared is used to evaluate the goodness of fit of a regression model. R-squared value ranges between 0 and 1, where 1 represents a perfect fit.

Practice task

The diabetes dataset can be loaded using load_diabetes() function from sklearn.datasets. This dataset contains 442 samples and 10 features, including age, sex, body mass index, average blood pressure, and six blood serum measurements. The target variable in this dataset is a quantitative measure of disease progression one year after baseline.

With necessary hyperparameter tuning, train an SVM algorithm using 75% of the dataset and then predict diabetes progression on the rest 25% of the dataset. Report the performance using at least four evaluation metrics.