Kernel Trick and SVM

Cheryl KOUADIO

2024-10-04

Activity 1 : Kernel Ridge Regression (KRR)

In regression and classification, we often use linear models to predict the target variable. However, in many cases, the relationship between the target variable and the explanatory variables is non-linear. In such cases, we can use the kernel trick whenever there is a scalar product between the explanatory variables. The kernel trick allows us to transform the data into a higher-dimensional space where the relationship is linear.

In this first activity, we will explore the kernel trick to transform the data and then use a linear model to predict the target variable. In particular, we will use Kernel ridge regression (KRR) which is a combination of ridge regression and the kernel trick. The optimization problem of KRR is given by:

$$\hat{\theta} = \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \theta) + \lambda \sum_{j=1}^{d} \theta_j$$

where x_i is the i-th row of the matrix X and y_i is the i-th element of the vector y. The parameter λ is the regularization parameter. The solution of the optimization problem is given by:

$$\hat{\theta} = (X^T X + \lambda I_d)^{-1} X^T y = X^T (X X^T + \lambda I_n)^{-1} y$$

where I_d and I_n are the identity matrix.

In prediction, the target variable is given by:

$$\hat{y}(x^*) = X^T \hat{\theta} = \langle x^*, \hat{\theta} \rangle = \left\langle x^*, \sum_{i=1}^{n} \alpha_i x_i \right\rangle = \sum_{i=1}^{n} \alpha_i \langle x_i, x^* \rangle$$

where $\alpha_i = \sum_{j=1}^{n} \theta_j x_{ij}$. We easily see that the prediction is a linear combination of the scalar product between the test point x^* and the training points x_i, we can use the kernel trick to transform the data into a higher-dimensional space where the relationship is linear. The prediction becomes:

$$\hat{y}(x^*) = \sum_{i=1}^{n} \alpha_i K(x_i, x^*)$$

where $K(x_i, x^*)$ is the kernel function.

I. Fit the Kernel Ridge Regression (KRR)

In problem which involves a distance between point, it is a common practice to normalize the data. In this notebook, we are going to normalize the data by dividing the time by 60 to convert it from minutes to hours (and have values between 0 and 1). We are going to use the `KernelRidge` class from the `sklearn` library to fit the KRR model. We will start by using the Gaussian/RBF kernel which is defined by:

$$K(x, x') = \exp\left(-\gamma ||x - x'||^2\right)$$

where γ is an hyperparameter.

```
# Libraries
import pandas as pd
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
#!pip install umap-learn
import umap.umap_ as umap
import numpy as np
import math
import warnings
warnings.filterwarnings("ignore")
```

```
/Users/cherylkouadio/Library/Python/3.9/lib/python/site-packages/tqdm/auto.py:21: TqdmWarnin

IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs
```

```
# Load the data
mcycle_data = pd.read_csv("Data/mcycle.csv")
mcycle_data["times"] = mcycle_data["times"]/60
mcycle_data.describe()
```

	times	accel
count	133.000000	133.000000
mean	0.419649	-25.545865
std	0.218868	48.322050
min	0.040000	-134.000000
25%	0.260000	-54.900000
50%	0.390000	-13.300000
75%	0.580000	0.000000
max	0.960000	75.000000

```
# Split the data into train and test sample
X = mcycle_data[["times"]]
y = mcycle_data[["accel"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1. Training and testing RBF kernel ridge regression

Since we have two hyperparameter (γ and λ) to tune, we can use the GridSearchCV function from scikit-learn to find the best hyperparameter.

```
rbf_krr_model = KernelRidge(kernel="rbf")
grid_eval = np.logspace(-2, 4, 50)
param_grid = {"alpha": grid_eval, "gamma": grid_eval}
rbf_krr_model_cv = GridSearchCV(rbf_krr_model, param_grid).fit(X_train,y_train)
print(f"Best parameters by CV : {rbf_krr_model_cv.best_params_}")
```

```
Best parameters by CV : {'alpha': np.float64(0.07196856730011521), 'gamma': np.float64(35.56
```

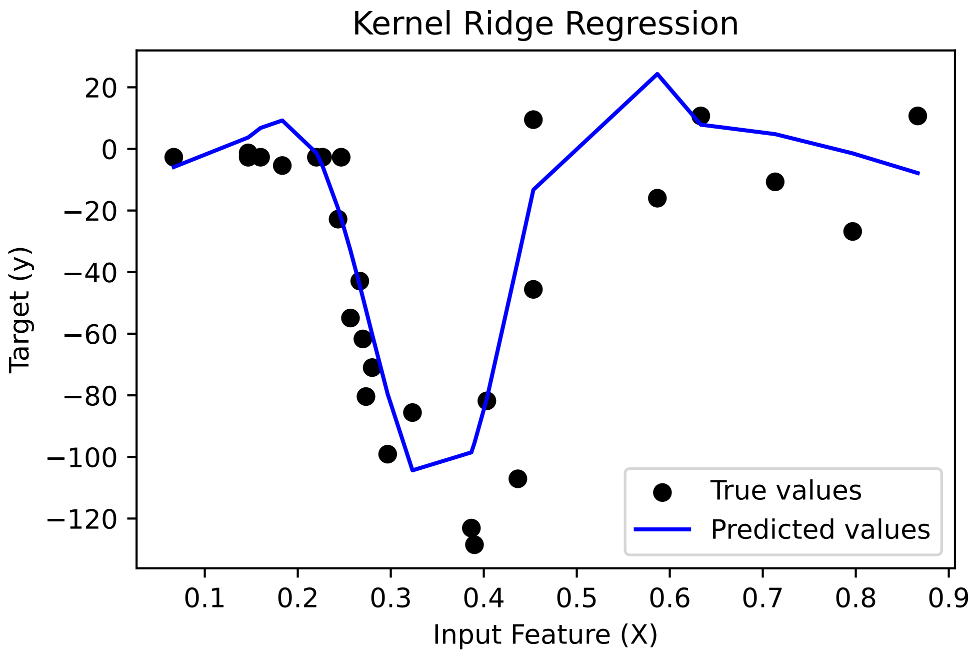
```
best_model = KernelRidge(kernel="rbf", alpha=rbf_krr_model_cv.best_params_["alpha"], gamma=r
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

print(f"Root mean square error: {math.sqrt(mean_squared_error(y_test, y_pred)): .2f}")
```

```
Root mean square error:  22.98
```

```
# Sort X_test and corresponding y_pred values
sorted_indices = np.argsort(X_test.values.flatten())
X_test_sorted = X_test.values.flatten()[sorted_indices]
y_pred_sorted = y_pred[sorted_indices]

plt.scatter(X_test, y_test, color="black", label="True values")
plt.plot(X_test_sorted, y_pred_sorted, color="blue", label="Predicted values")
plt.title("Kernel Ridge Regression")
plt.xlabel("Input Feature (X)")
plt.ylabel("Target (y)")
plt.legend(loc="best")
plt.show()
```



II. Ridge regression

Now we are going to use the basic ridge regression to predict the target variable. There is only one hyperparameter to tune which is the regularization parameter λ.