Assignment 6: Kernels

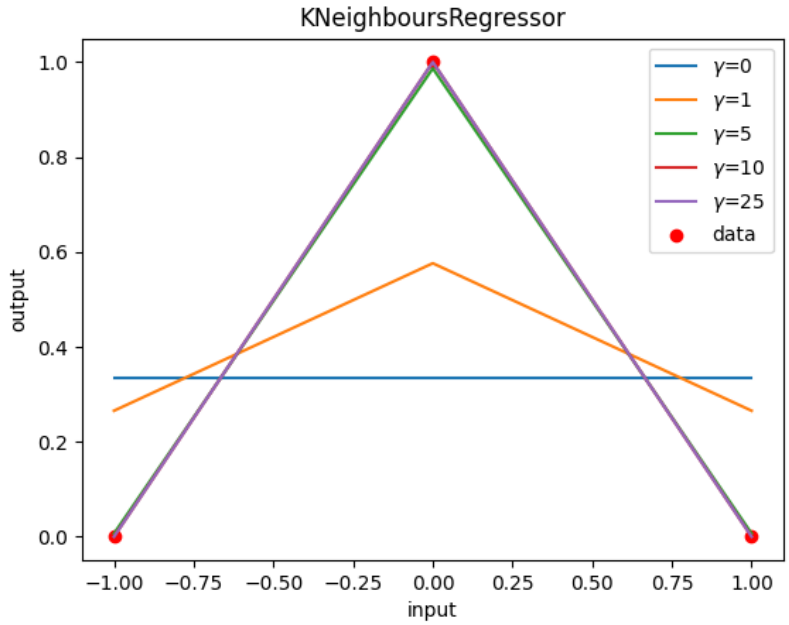
Ciarán Donegan
16322253

18 November 2020

Dataset # id:16–32-80

(i) (a) Training a kNN regression model to fit the three data points $[-1, 0], [0, 1], [1, 0]$. The value of k in the model is 3 i.e. $k =$ `# training points`.

Five separate knn models were trained using Gaussian kernels with kernel paramater $\gamma = \{0, 1, 5, 10, 25\}$. The model predictions and training data is shown below.



(b) A kernel $K(x_i, x)$ works by measuring the distance between an input x and a training point x_i and outputting a real value. By training a model using these kernalised features, you attach more weight to training data points that are close to the input x and less weight to far away training points.

A Gaussian kernel is defined as

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

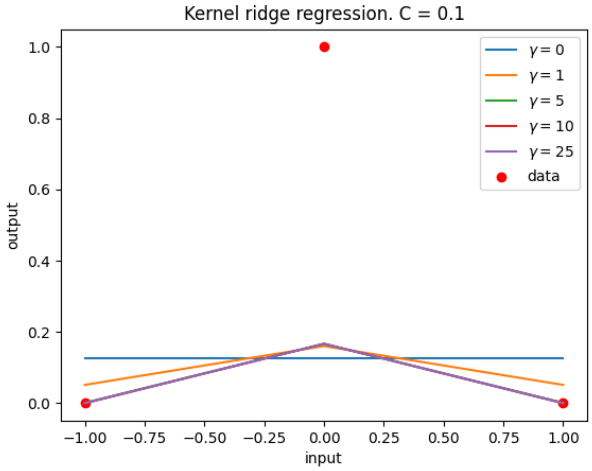
where γ controls how quickly $K(x_i, x)$ decreases as the distance between x_i and x grows.

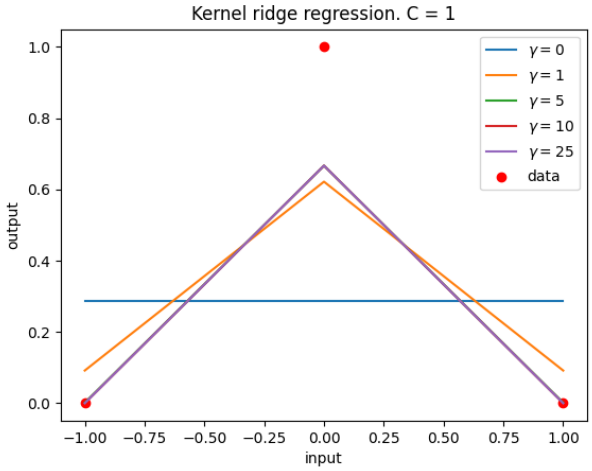
Looking at the plot in (i)(a) you see that when $\gamma = 0$ (i.e. you weight all points equally regardless of distance), you get a model that predicts a straight line in the middle of the data. It is closer to the bottom two points than the top one, because there are 2 points there versus only 1 at the top.

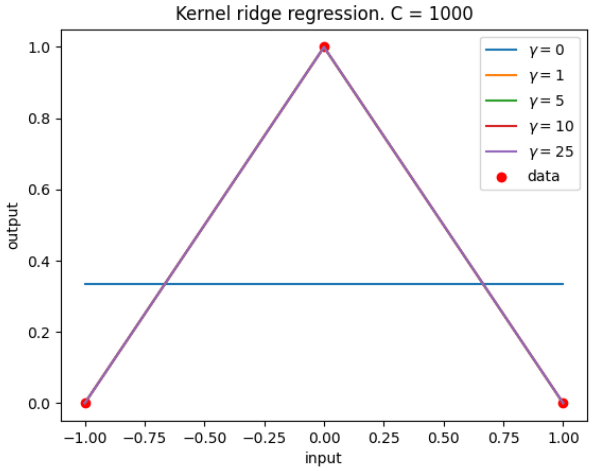
As you increase γ, you apply more weight to closer training points, so the model will track the training points more closely (having γ too high will overfit the data and lead to poor generalisation).

When $\gamma = 25$, the predictions are close to 1 for inputs between -0.5 and 0.5 because with such a large γ, you weight the closest training point very high. For these input ranges, the closest training point has label 1, and so this is what the model predicts.

(c)







	$C = 0.1$	$C = 1$	$C = 1000$
$\gamma = 0$	[[-0.025] [0.175] [-0.025]]	[[-0.57142857] [1.42857143] [-0.57142857]]	[[-666.55557407] [1333.44442593] [-666.55557407]]
$\gamma = 1$	[[-0.01026472] [0.16792539] [-0.01026472]]	[[-0.18331618] [0.75658434] [-0.18331618]]	[[-0.49138748] [1.36086227] [-0.49138748]]
$\gamma = 5$	[[-0.00018717] [0.16666709] [-0.00018717]]	[[-0.00299476] [0.66669357] [-0.00299476]]	[[-0.00673182] [0.99959092] [-0.00673182]]
$\gamma = 10$	[[-1.26110916e-06] [1.66666667e-01] [-1.26110916e-06]]	[[-2.01777466e-05] [6.66666668e-01] [-2.01777466e-05]]	[[-4.53545640e-05] [9.99500254e-01] [-4.53545640e-05]]
$\gamma = 25$	[[-3.85776218e-13] [1.66666667e-01] [-3.85776218e-13]]	[[-6.17241950e-12] [6.66666667e-01] [-6.17241950e-12]]	[[-1.38740663e-11] [9.99500250e-01] [-1.38740663e-11]]

Table 1: Trained θ paramaters (θ_1, θ_2, θ_3)for each model. These θ's indicate the weight each training point has in making a prediction.

(d) A kernalised ridge regression model predicts it outputs according to

$$\hat{y} = \theta_0 + \theta_1 y_1 K(x_i, x) + ... + \theta_m y_m K(x_m, x) + \frac{1}{2C}\theta^T\theta$$

where $\frac{1}{2C}\theta^T\theta$ is the L2 regularisation penalty.

As you would expect, increasing the value of the hyper-parameter C, increases the magnitude of the learned parameters because there is less of a regularisation penalty being applied. This explains why the learned parameters (θ's) increase as you move left to right across each column in the table above.

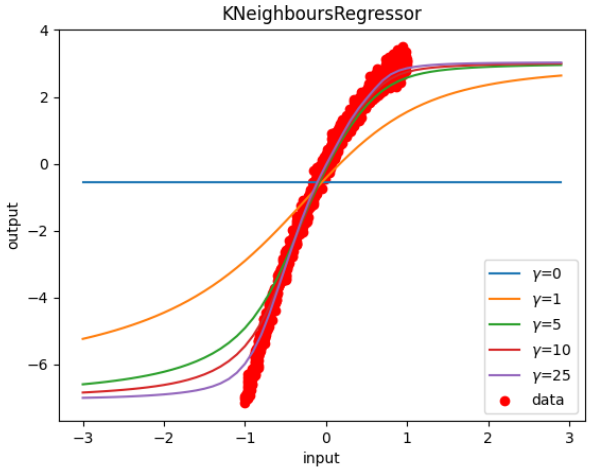
The plots from (c) show that for a strong regularisation penalty $C = 0.1$, the output predictions for every γ are small. This is because the strong regularisation diminishes the influence of the data point at $(0, 1)$. As you decrease the regularisation (by increasing C), the model does not diminish the effect of the single data point at $(0, 1)$ and so the predictions go through this point.

As you increase γ, far away data points have less influence on the predictions. Therefore, the learned θs are smaller because there is less weight being applied. Moving down any column in table 1, you will notice how the θs decrease.

Looking at the plots in (c), you see that for $\gamma = 0$, the predictions form a straight line. This is because all training points have equal influence in the predictions. Increasing γ results in a more pronounced triangle in the predictions as the model more closely tracks the training points. For a high γ, the closest training points will mostly determine the output which is why the predictions go through the three training data points.

Comparing kNN and kernalised ridge regression models, I see that both models require fine tuning of hpyerparameters but it seems the ridge regression model produces more nonsensical outputs with badly tuned hyperparameters.

(ii) (a)

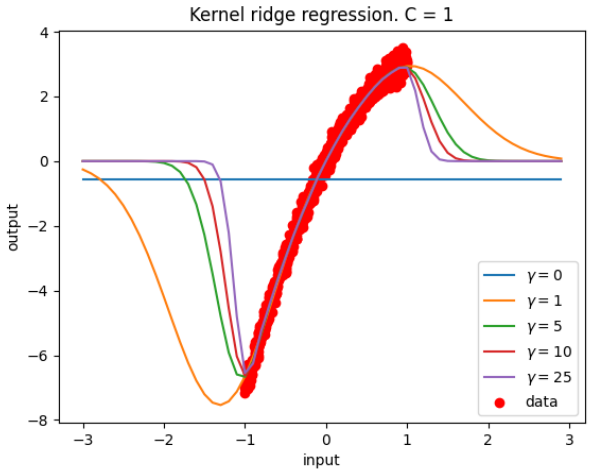


As seen before, with $\gamma = 0$, the predictions form a straight line because all the points are weighted equally regardless of distance. The straight line is closer to the top of the plot because there are more training points up here.

The more you increase γ, the more closely the predictions follow the training data because it weights closer points more highly.

The predictions for unseen inputs (inputs not seen in training data) show a poor generalisation ability of this kNN model. The predictions for unseen inputs are determined by the closest training points and so the curves taper toward flat lines beyond the training data range.

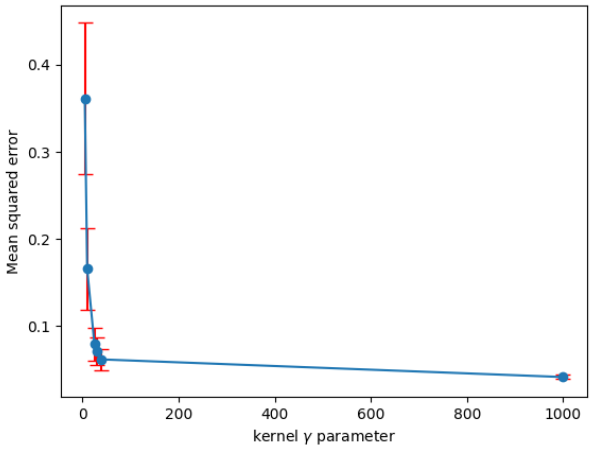
(b)



As always, setting $\gamma = 0$ results in straight line predictions.

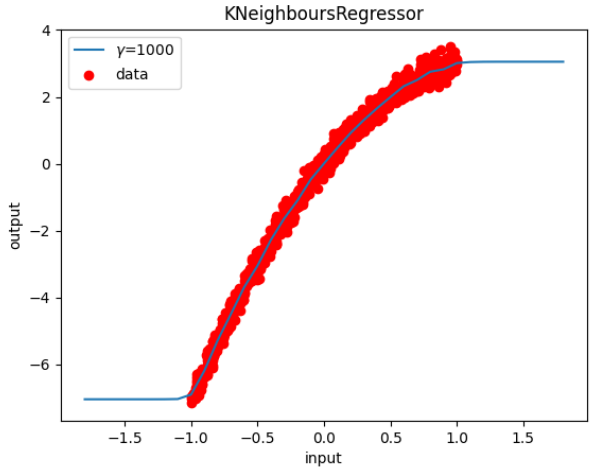
What's interesting is that the $\gamma = 1$ model seems to perform the best. This is because with higher γ values, the model doesn't consider training points that are far away and so the generalisation ability worsens.

(c)

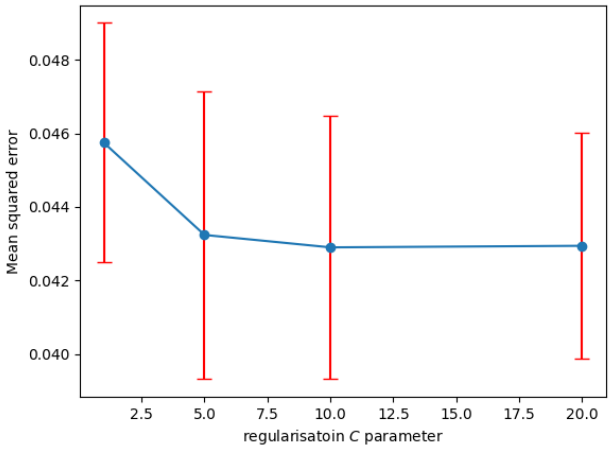


Performing cross-validation on the kNN model leads to peculiar results. It seems as though increasing γ will constantly improve the mean square error (i.e. lower MSE). This implies that the less weight you give to training data points, the better.

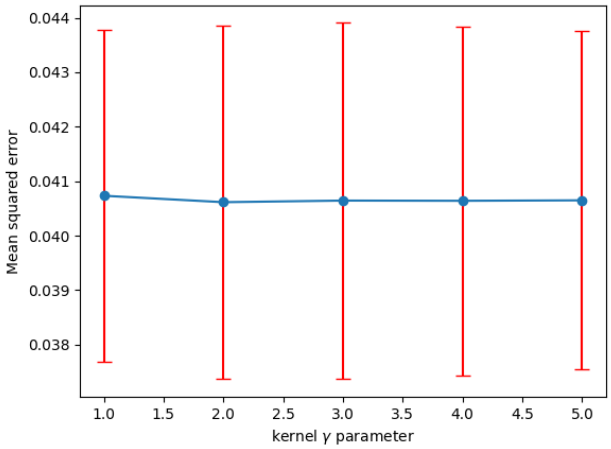
This is probably an attempt to overcome kNNs tendency to over fit the training data. The results of my cross-val show that in theory, an infinitely large γ would preform the best. For simplicity I'm just going to choose $\gamma = 1000$.



The figure above shows the predictions for the kNN model on this "optimized" hyperparametr value of $\gamma = 1000$.

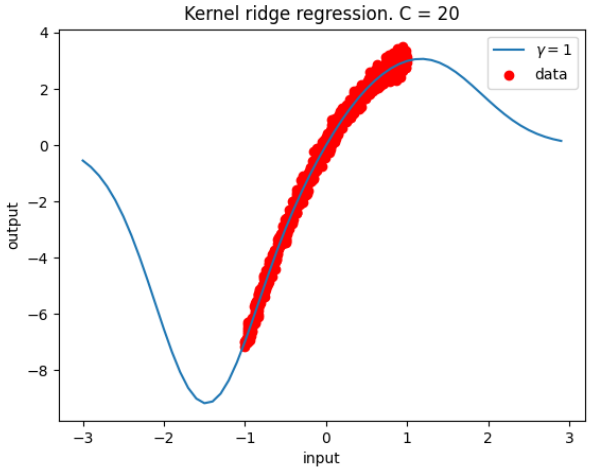


From my cross-val analysis on kernel ridge regression model, I found the best regularisation hyper-parameter to be $C = 20$. (Using 10 fold cross validation with $\gamma = 0$ for all.)



From my cross-val analysis on kernel ridge regression model, I found that kernel γ values between 1 and 5 all perform equally. (Using 10 fold cross validation with $C = 20$ for all.)

I therefore choose $\gamma = 1$, but other options perform equally well.



Predictions for "optimized" ridge regression model with $C = 20$ and $\gamma = 1$.

Both kNN and the ridge model have pretty bad generalisation ability. For the kNN model, the predictions beyond the training data range are a straight line. For the ridge regression model, the predictions beyond the training range are some complex polynomial.

It is clear from this analysis that kernel methods generalise very badly to unseen data.

A Code

```
import os
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import KFold
from sklearn.kernel_ridge import KernelRidge
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt



def read_csv(filename):
    df = pd.read_csv(filename, comment="#")
    # print(df.head())
    X = np.array(df.iloc[:,0])
    X = X.reshape(-1, 1)
    y = np.array(df.iloc[:,1])
    y = y.reshape(-1,1)
    return X, y


def knn_and_ridge_regression_kernels(X, y):

    X_test = np.arange(-3, 3, 0.1).reshape(-1,1)

    def gaussina_kernel(distances):
        weights = np.exp(-gamma * (distances**2))
        return weights / np.sum(weights)

    plt.figure()
    plt.scatter(X, y, color='red', label="data")
    for gamma in [1000]:
        model = KNeighborsRegressor(n_neighbors=len(X), weights=gaussina_kernel)
        model.fit(X, y)
        y_pred = model.predict(X_test)


        plt.plot(X_test, y_pred, label="$\gamma$="+str(gamma))
    plt.legend()
    plt.xlabel("input")
    plt.ylabel("output")
    plt.title("KNeighboursRegressor")
    plt.savefig(out_dir + "knn_optimized.png")

    for c in [20]:
        plt.figure()
        plt.scatter(X, y, color='red', label="data")
        for gamma in [1]:
            model = KernelRidge(alpha=1/(2*c), kernel='rbf', gamma=gamma)
            model.fit(X, y)
            y_pred = model.predict(X_test)
            # print("dual_coef", model.dual_coef_)
            plt.plot(X_test, y_pred, label="$\gamma=$"+str(gamma))
        plt.legend()
        plt.xlabel("input")
        plt.ylabel("output")
        plt.title("Kernel ridge regression. C = " + str(c) )
        plt.savefig(out_dir + "kernel_ridge_optimized" + str(c)+ ".png")


def do_kfold(X, y, method, gamma=0, C=1, kfolds=10):
    valid_methods = ["knn", "ridge_regression"]
    if method not in valid_methods:
        raise ValueError("Invalid type")

    def gaussina_kernel(distances):
```

```
        weights = np.exp(-gamma * (distances**2))
        return weights / np.sum(weights)

    kf = KFold(n_splits=kfolds)
    mean_error = []
    for train, test in kf.split(X):
        if method == "knn":
            model = KNeighborsRegressor(n_neighbors=len(X[train]), weights=gaussina_kernel)
        else:
            model = KernelRidge(alpha=1/(2*C), kernel='rbf', gamma=gamma)
        model.fit(X[train], y[train])
        y_pred = model.predict(X[test])
        mean_error.append(mean_squared_error(y[test], y_pred))
    mse = np.array(mean_error).mean()
    std = np.array(mean_error).std()
    # print("MSE = ", mse)
    # print("STD = ", std)
    return model, mse, std



if __name__ == "__main__":

    out_dir = "output_figs/"
    if not os.path.exists(out_dir):
        os.makedirs(out_dir)


    # For dummy data use these
    # data  = np.array([[-1,0], [0,1], [1,0]])
    # X = data[:,0].reshape(-1, 1)
    # y = data[:,1].reshape(-1,1)

    # Load week 6 dataset
    X, y =read_csv("week6.csv")

    # do k-fold for knn
    mse_list = []
    std_list = []
    gammas = [5, 10, 25, 30, 40, 1000]
    for gamma in gammas:
        model, mse, std = do_kfold(X, y, "knn", gamma)
        mse_list.append(mse)
        std_list.append(std)
        plt.figure()
    plt.xlabel("kernel $\gamma$ parameter")
    plt.ylabel("Mean squared error")
    plt.errorbar(gammas, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/knn_mse_vs_gamma.png")

    # do k-fold for ridge regression
    mse_list = []
    std_list = []
    c_vals = [ 1, 5, 10, 20]
    for C in c_vals:
        model, mse, std = do_kfold(X, y, "ridge_regression", C)
        mse_list.append(mse)
        std_list.append(std)
        plt.figure()
    plt.xlabel("regularisatoin $C$ parameter")
    plt.ylabel("Mean squared error")
    plt.errorbar(c_vals, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/ridge_mse_vs_C.png")

    mse_list = []
    std_list = []
    gammas = [ 1, 2, 3, 4, 5]
    for gamma in gammas:
```

```
        model, mse, std = do_kfold(X, y, "ridge_regression", gamma, C=20)
        mse_list.append(mse)
        std_list.append(std)
        plt.figure()
    plt.xlabel("kernel $\gamma$ parameter")
    plt.ylabel("Mean squared error")
    plt.errorbar(gammas, mse_list, yerr=std_list, fmt="-o", ecolor="r", capsize=5)
    plt.savefig(out_dir + "/ridge_mse_vs_gamma.png")

    knn_and_ridge_regression_kernels(X,y)
```