The top gun of non parametric regressors:

KERNEL REGRESSION

By Bessi Marco and Tanasa Dragos

```python
def montecarlo1():

    montecarslo_simulation = 1000
    error_knn = []
    error_reg = []


    #MONTECARLO SIMULATION
    for i in tqdm (range (montecarslo_simulation), desc="Loading..."):

        #generate data for every simulation
        X, Y = data_generating_process(dimensions = 600, feature = 1)

        #split data into train and test
        x_train, x_test, y_train, y_test = train_test_split(Y, X, test_size = 0.2)
        x_train = np.array(x_train).reshape(-1,1)
        x_test = np.array(x_test).reshape(-1,1)

        #grid search for best parameters for both KNN and Kernel Regression
        knn = KNN()
        param_grid_knn = { "n_neighbors": np.arange(1, 100) }
        knn_gscv = GridSearchCV(knn, param_grid_knn, cv=5, scoring="neg_mean_squared_error")
        knn_gscv.fit(x_train, y_train)

        ker2 = KernelRegression()
        param_grid = { "bandwidth": np.arange(0.02, 0.4, 0.02) }
        ker_gscv = GridSearchCV(ker2, param_grid, cv=5, scoring="neg_mean_squared_error")
        ker_gscv.fit(x_train, y_train)

        #get best estimator for both KNN and Kernel Regression
        knn_best = knn_gscv.best_estimator_
        ker_best = ker_gscv.best_estimator_

        #predict values for both KNN and Kernel Regression
        knn_pred = knn_best.predict(x_test)
        ker_pred = ker_best.predict(x_test)

        error_knn.append(mean_squared_error(y_test, knn_pred))
        error_reg.append(mean_squared_error(y_test, ker_pred))

    error_knn = np.asarray(error_knn)
    error_reg = np.asarray(error_reg)
    np.savetxt("error_knn_1000_600.csv", error_knn, delimiter=",")
    np.savetxt("error_reg_1000_600.csv", error_reg, delimiter=",")
```

Generate data and split into train and test

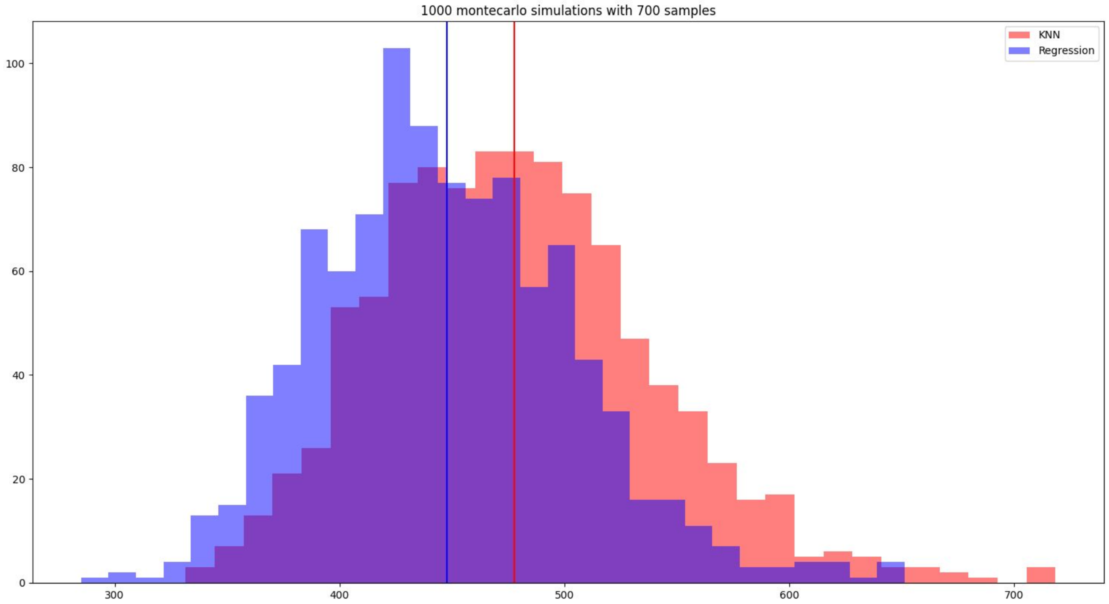
The best hyperparameter is chosen using grid search

Regressor are evaluated on test data, mean squared error is used.

KNN vs Regression

Result of the (really long) Montecarlo simulation are displayed as the histogram of the MSE.

As we can see Kernel Regression performs on average better than KNN regression

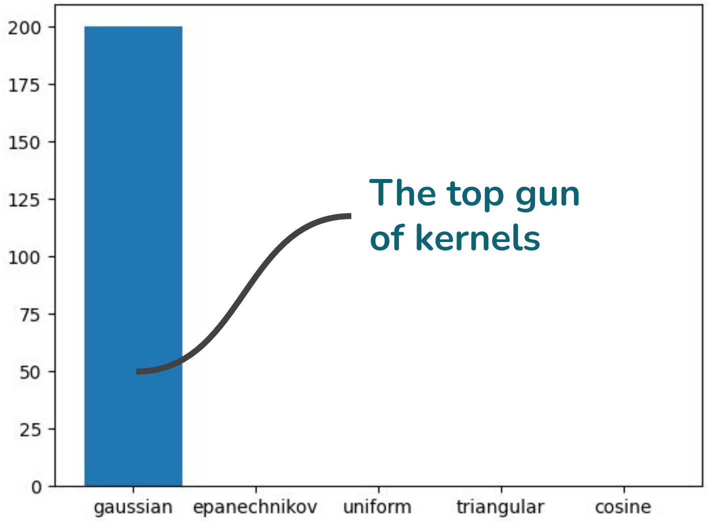


What about kernels?

Our code implements 5 different types of kernels:

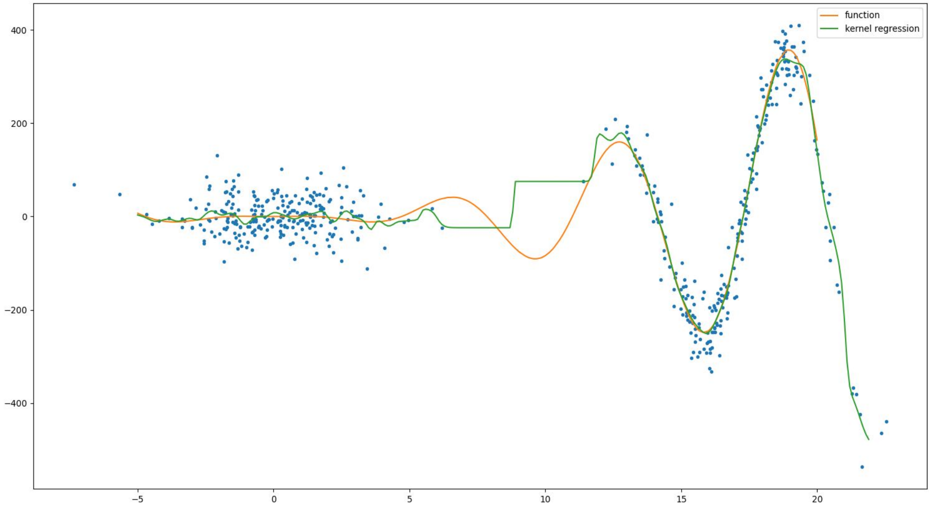
```
class Kernel:
    def __init__(self, kernel_type = "gaussian"):
        self.kernel_type = kernel_type

    def __call__(self, value):
        if self.kernel_type == "gaussian":
                return np.exp(-value**2/(2))/(np.sqrt(2*np.pi))
        if self.kernel_type == "uniform":
            return np.where(np.abs(value) <= 1, 1/2, 0)
        if self.kernel_type == "triangular":
            return np.where(np.abs(value) <= 1, 1 - np.abs(value), 0)
        if self.kernel_type == "epanechnikov":
            return np.where(np.abs(value) <= 1, 3/(4)*(1 - (value)**2), 0)
        if self.kernel_type == "cosine":
            return np.where(np.abs(value) <= 1, np.pi/(4)*np.cos(np.pi*value), 0)
```

Results of a montecarlo simulation.



Non uniform sample

```python
def data_generating_process(dimensions = 100):

    mu, sigma = 0, 2
    mu2, sigma2 = 17, 2
    X1 = np.random.normal(mu, sigma, int(dimensions/2))
    X2 = np.random.normal(mu2, sigma2, int(dimensions/2))
    X = np.concatenate([X1, X2])
    X = np.sort(X)
    Y = [(x ** 2)*np.cos(x) + np.random.normal(0, 40) for x in X]

    return(Y, X)
```

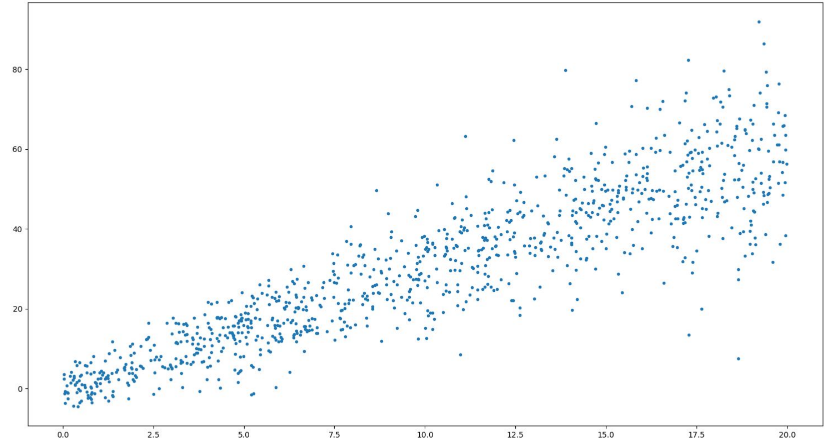


When data is not uniformly sampled kernel regression struggles to estimate the real function.

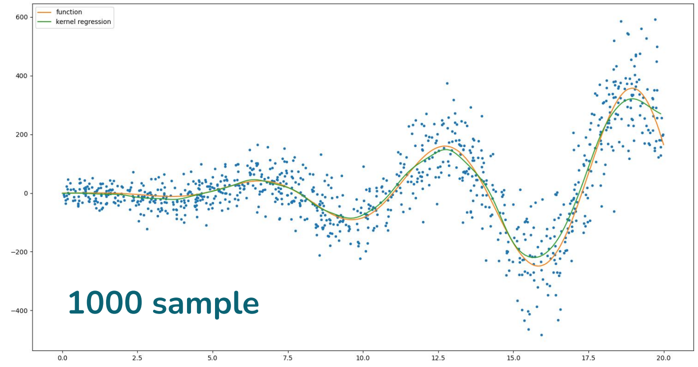
What about heteroscedasticity?

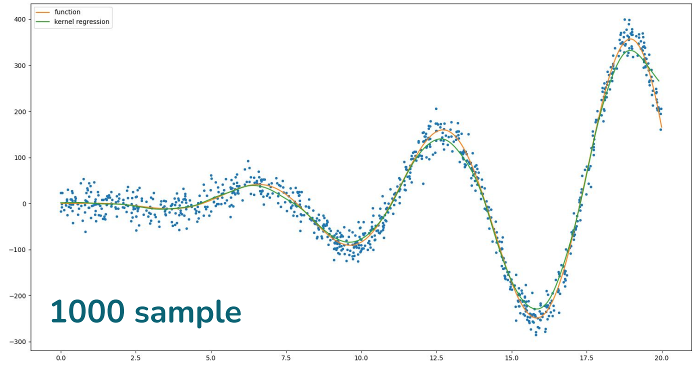
Some empirical observation about what we removing homoscedasticity assumption in the data

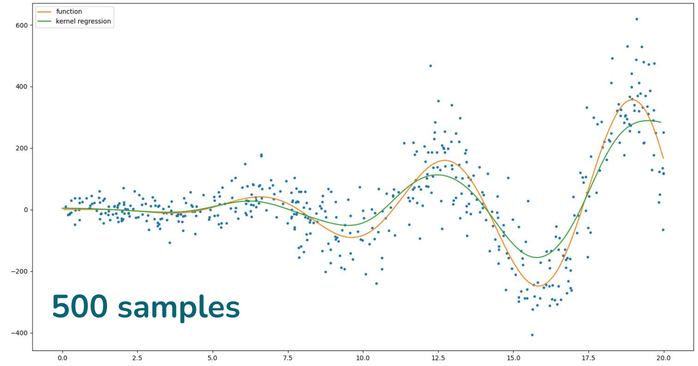
```
def data_generating_process(dimensions = 100):

        X = np.random.uniform(0, 20, dimensions)
        X = np.sort(X)
        Y = [(x ** 2)*np.cos(x) + np.random.normal(0, 20 + 5*x) for x in X]

        return(Y, X)
```

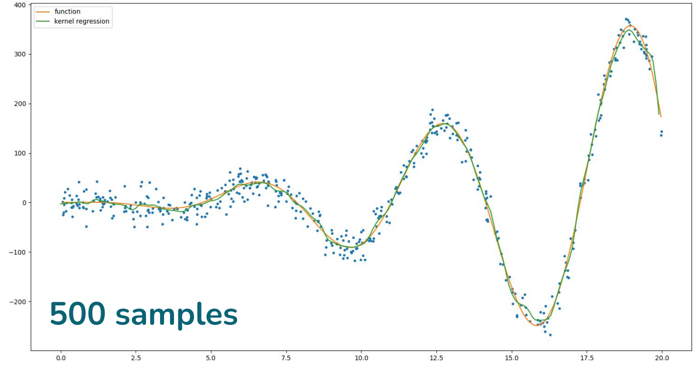


Heteroscedasticity vs Homoscedasticity









Reference:

- Nonparametric regression, Wikipedia.
- Kernel regression, Wikipedia.
- Notes from the course of Predictive Modelling of the University of Madrid

Code used:

- https://github.com/DragosTana/ML_Homework