7 Problems and Solutions (Part 1 of 3)

Kernel Regression

Background: Consider applying linear regression to nonlinear data. We could use a polynomial like $y = w_1 x_i + w_2 x_i^2 + b$ and check graphically (or by comparing MSE\text{\_train} and MSE\text{\_test}) for a good fit. However, for $D > 3$ dimensions, finding the right polynomial could be hard.

Kernel regression is a non-parametric method. It extends weighted $k$-NN (another non-parametric method) to the case of $\hat{f}$.

- Its simplest form for 1D $x = x$ is $f(x) = \sum_{i=1}^{N} w_i y_i$, where $w_i = \frac{k\left(\frac{x-x_i}{b}\right)}{\sum_{j=1}^{N} k\left(\frac{x-x_j}{b}\right)}$.
- $f(x)$ is a weighted average of $\{y_i\}$ since $\sum w_i = 1$.
- $k()$ is a kernel that plays the role of a similarity function. Coefficients $w_i$ are higher when $x$ is similar to $x_i$ and lower otherwise.
- The most common kernel is the Gaussian kernel, $k(z) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right)$.

The bandwidth $b$ is a hyperparameter tuned using cross-validation.

- e.g. Draw $\{(x_i, y_i)\}$ in nonlinear pattern. Add bell curves (wide, narrow, just right).

Python

- from sklearn.gaussian_process.kernels import ConstantKernel, WhiteKernel, RBF
  - ConstantKernel(constant_value=1.0, constant_value_bounds=(1.0e-5, 1.0e5)) can be used in a product for scaling or in a sum for modifying the mean.
  - WhiteKernel(noise_level=1.0, noise_level_bounds=(1.0e-5, 1.0e5)) gives the white noise kernel with noise $\sim N(\mu=0, \sigma^2=\text{noise\_level})$.
  - RBF(length_scale=1.0, length_scale_bounds=(1.0e-05, 1.0e5)) gives the Radial Basis Function (RBF) kernel, also called the Gaussian kernel.
    * $b=\text{length\_scale}$ is the bandwidth
    * length\_scale\_bounds provides lower and upper bounds on length\_scale; or if set to 'fixed', then length\_scale cannot change

There are many other kernels.

\footnote{For vector input, $x_i - x$ is replaced by Euclidean distance $||x_i - x||$. Where Burkov uses $b$ for bandwidth in the arguments to $k()$, scikit-learn uses $l$ for length\_scale in the definition of $k()$. Burkov uses factors $\frac{1}{N}$ in front of his $f(x)$ sum and $N$ in his $w_i$ definition; I omitted them since $N \frac{1}{N} = 1$. Burkov uses $x_i - x$ and $x_j - x$ where I used their opposites to see that we’re standardizing $x$ using $\mu = x_i$ and $\sigma = b$; $k()$ is symmetric, so this matters only for clarity (and compatibility with his §9).

\footnote{A kernel function is also called a covariance function. Random variables $X$ and $Y$ have covariance $\sigma_{XY} = E[(X-\mu_X)(Y-\mu_Y)]$ and correlation $\rho_{XY} = \sigma_{XY}/(\sigma_X\sigma_Y)$. Note: $\sigma_{XX} = \sigma_X^2$.}}
from sklearn.gaussian_process import GaussianProcessRegressor

model = GaussianProcessRegressor(kernel=None,
        optimizer='fmin_l_bfgs_b',
        n_restarts_optimizer=0,
        random_state=0)
gives kernel regression:

* kernel = ConstantKernel() * RBF() gives Gaussian kernel regression
* kernel = ConstantKernel() * RBF() + WhiteKernel() gives smoother Gaussian kernel regression.
* optimizer='fmin_l_bfgs_b', the default, sets $b$ (and a scale factor $\sigma^2$ and $\sigma_\epsilon^2$) to optimal values; optimizer=None holds hyperparameters fixed.
* n_restarts_optimizer > 0 can help escape a local minimum.

model.fit(X, y), .predict(X), .score(X, y) are like OLS in §3
model.kernel_ gives hyperparameters chosen by .fit() and used for prediction.
e.g. After fitting RBF(), model.kernel_ gives bandwidth $b = \text{length\_scale}$, which determines the smoothness of $f$.
e.g. After fitting WhiteKernel(), model.kernel_ shows $\sigma_\epsilon^2 = \text{noise\_level}$.

To learn more:

- Choosing the kernel: [https://www.cs.toronto.edu/~duvenaud/cookbook](https://www.cs.toronto.edu/~duvenaud/cookbook)
- Gaussian Processes API: [https://scikit-learn.org/stable/modules/classes.html#module-sklearn.gaussian_process]

3 These scikit-learn kernels do not correspond exactly with the math formulas I presented.
4 This example is very helpful.