Chapter 2. Overview of Supervised Learning

ESL2 review

오영민

Variable Types and Terminology

- Quantitative measurement
  - Continuous
  - Ex. Heights, price, …
- Qualitative measurement
  - Discrete or Categorical; factors
  - Ex. Iris discrimination, gender, …
- Ordered categorical
  - Ex. Small, Medium, Large

Variable Types and Terminology

- Input variable: $\mathbf{X} = (X_1, ..., X_p) \in \mathbb{R}^{N \times p}$
  - Covariates, predictors, features, independent variables
- Output variable
  - Quantitative output: $Y \in \mathbb{R}$, Qualitative output: $G \in \mathcal{G}$ (for group)
  - Responses, dependent variables
- Training Data: a set of measurements: $\{(x_i, y_i) : i = 1, ..., N\}$
- Goal: Use the inputs to predict the values of outputs

Statistical Decision Theory

- Let $X \in \mathbb{R}^p, Y \in \mathbb{R}$ with joint distribution $Pr(X, Y)$
- Goal: Seek a function $f(X)$ for predicting Y with Loss function $L(Y, f(X))$ which penalizing errors in prediction
  - For regression, the most common choice is squared error loss:

$$EPE(f) = \mathbb{E}_{X,Y}(Y - f(X))^2 = \mathbb{E}_X \mathbb{E}_{Y|X}([Y - f(X)]^2 | X)$$

and we see that it suffices to minimize EPE pointwise:

$$f(x) = argmin_c \mathbb{E}_{Y|X}([Y - c]^2 | X = x) = \mathbb{E}(Y | X = x)$$

Thus the prediction of Y at any point X = x is the conditional mean.

Nearest-Neighbor Methods

- k-NN fit for $\hat{Y}$ is defined as follows:

  $\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$, $N_k(x) = \{x \in \mathcal{T} : d(x,p) \leq r_k, r_k \text{ is k-th closest points } x_i \in \mathcal{T}\}$

  It can also be written as $\hat{f}(x) = Avg(y_i | x_i \in N_k(x))$

- **Thm.** Under mild regularity conditions on $Pr(X,Y)$, as N,k are large enough s.t. k/N close to 0,

  $$\hat{f}(x) \to \mathbb{E}(Y|X=x)$$

- NN method is a direct attempt to estimate conditional expectation using the training data
  - Expectation is approximated by averaging over sample data
  - Conditioning at a point is relaxed to conditioning on some region "close" to the target point

Least Square Estimator

- Assume that the regression function is approximately linear.

$$\beta^* = argmin_{\beta}\mathbb{E}(Y - X^T\beta)^2 \text{ where } Y \in \mathbb{R},\ X, \beta \in \mathbb{R}^{p+1}$$

We can solve for β theoretically:

$$\frac{\partial}{\partial\beta}\mathbb{E}(Y - X^T\beta)^2 = -2\mathbb{E}(XY) + 2\beta\mathbb{E}(XX^T) = 0$$

$$\beta^* = [\mathbb{E}(XX^T)]^{-1}\mathbb{E}(XY)$$

Note that the least squares solution $\hat{\beta} = (\mathtt{X}^T\mathtt{X})^{-1}\mathtt{X}^T\mathtt{Y}$ where $\mathtt{X} \in \mathbb{R}^{N\times(p+1)},\ \mathtt{Y} \in \mathbb{R}^N$
we know that for large enough N, $\frac{1}{N}\mathtt{X}^T\mathtt{X} \approx \mathbb{E}(XX^T), \frac{1}{N}\mathtt{X}^T\mathtt{Y} \approx \mathbb{E}(XY)$
i.e. the least squares solution amounts to replacing the expectation by averages over the training data

Remark.

- Both k-NN and LS end up approximation conditional expectations by averages.
- LS assumes f is well approximated by a globally linear function
- K-NN assumes f is well approximated by a locally constant function

Statistical Decision Theory: classification

- Let $\mathcal{G}$ be set of all possible classes, K be cardinality of $\mathcal{G}$,

  Define $\mathcal{L} \in \mathbb{R}^{K\times K}$ by $\mathcal{L}_{i,j} = \begin{cases} 0 & \text{if } i = j \\ L(i,j) \text{ nonnegative price} & \text{o.t.} \end{cases}$

  Then the expected prediction error is

$$EPE = \mathbb{E}[L(G, \hat{G}(X)] \text{ where } \hat{G} \in \mathcal{G}$$

$$= \mathbb{E}_X \sum_{k=1}^{K} L(\mathcal{G}_k, \hat{G}(X)) Pr(G = \mathcal{G}_k | X)$$ because G is discrete random variable

Statistical Decision Theory: classification

- Thus, it suffices to minimize EPE pointwise:

$$\hat{G}(x) = argmin_{g\in\mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g) Pr(G = \mathcal{G}_k | X = x)$$

- Now define loss function by $L(Y, f(X)) = I(Y \neq f(X))$, which is called 0-1 loss function.
- With the 0-1 loss function,

$$\begin{aligned} \hat{G}(x) &= argmin_{g\in\mathcal{G}} \sum_{g\neq\mathcal{G}} Pr(G = \mathcal{G}_k | X = x) \\ &= argmin_{g\in\mathcal{G}} \sum_{g\neq\mathcal{G}} Pr(G = \mathcal{G}_k | X = x) + Pr(G = g | X = x) - Pr(G = g | X = x) \\ &= argmin_{g\in\mathcal{G}} 1 - Pr(G = g | X = x) = argmax_{g\in\mathcal{G}} Pr(g | X = x) \end{aligned}$$

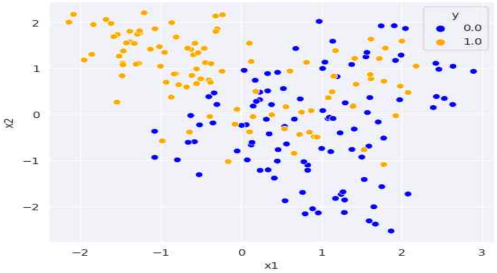
this solution is known as **Bayes classifier**

Example: binary classification

- Data distribution: $\mathcal{G} = \{\mathcal{G}_{\text{blue}}, \mathcal{G}_{\text{orange}}\}$ with $P(G = \mathcal{G}_{\text{blue}}) = P(G = \mathcal{G}_{\text{orange}}) = 0.5$

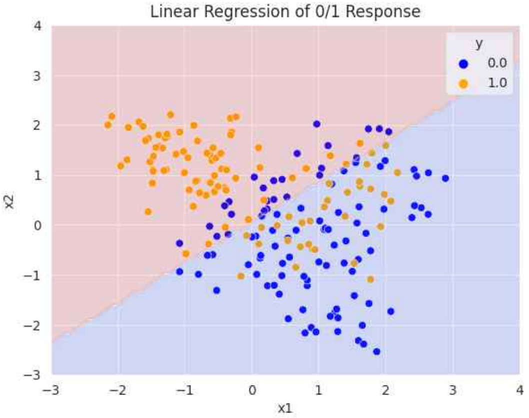
$\mu_1^{(\text{blue})}, \ldots, \mu_{10}^{(\text{blue})} \sim \mathcal{N}([1,0]^T, I),\ \mu_1^{(\text{orange})}, \ldots, \mu_{10}^{(\text{orange})} \sim \mathcal{N}([0,1]^T, I)$

For each observation, we picked an μ_k at random with probability 1/10, and sampling from $\mathcal{N}(\mu_k, I/5)$, thus leading to a mixture of Gaussian clusters for each class.



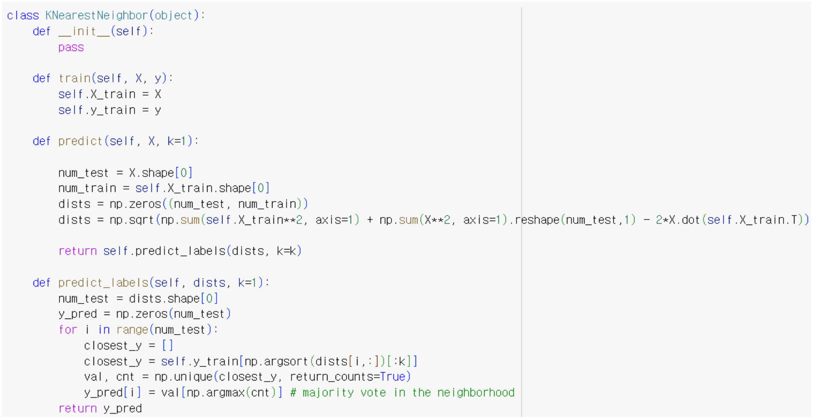
Example: LS

```
desinged_X = np.c_[np.ones(len(y)),X]
beta_hat = np.linalg.inv(desinged_X.T @ desinged_X) @ desinged_X.T @ y

X_new_with_bias = np.c_[np.ones([len(X_new), 1]), X_new]
y_pred = X_new_with_bias @ beta_hat
y_pred[y_pred <= 0.5] = 0
y_pred[y_pred > 0.5] = 1
```



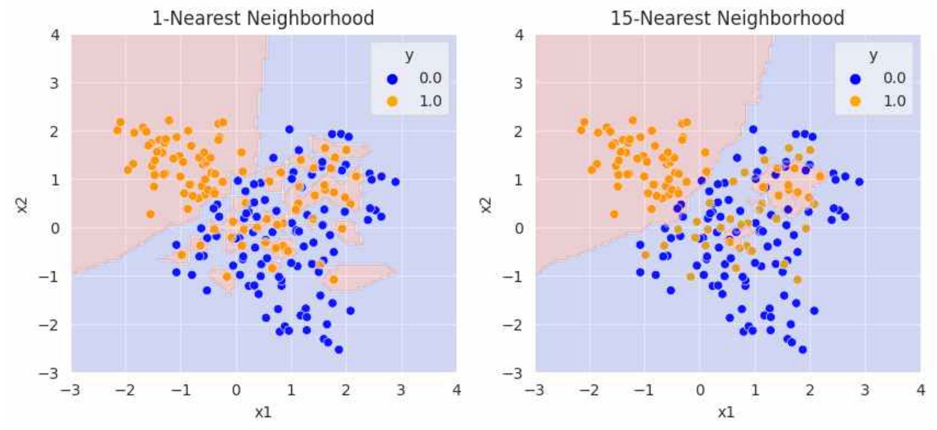
Example: k-NN classifier

$Pr(G = \mathcal{G}_k | X = x)$ be proportion for training sample $x \in N_k(x)$



```python
class KNearestNeighbor(object):
    def __init__(self):
        pass

    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X, k=1):

        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        dists = np.sqrt(np.sum(self.X_train**2, axis=1) + np.sum(X**2, axis=1).reshape(num_test,1) - 2*X.dot(self.X_train.T))

        return self.predict_labels(dists, k=k)

    def predict_labels(self, dists, k=1):
        num_test = dists.shape[0]
        y_pred = np.zeros(num_test)
        for i in range(num_test):
            closest_y = []
            closest_y = self.y_train[np.argsort(dists[i,:])[:k]]
            val, cnt = np.unique(closest_y, return_counts=True)
            y_pred[i] = val[np.argmax(cnt)] # majority vote in the neighborhood
        return y_pred
```

Example: k-NN classifier



Example: Bayes classifier

$\mathcal{G} = \{\mathcal{G}_{\text{blue}}, \mathcal{G}_{\text{orange}}\}$ with $P(G = \mathcal{G}_{\text{blue}}) = P(G = \mathcal{G}_{\text{orange}}) = 0.5$

$\mu_1^{(\text{blue})}, \ldots, \mu_{10}^{(\text{blue})} \sim \mathcal{N}([1,0]^T, I),\ \mu_1^{(\text{orange})}, \ldots, \mu_{10}^{(\text{orange})} \sim \mathcal{N}([0,1]^T, I)$

$P(X = x | G = \mathcal{G}_{\text{blue}}) = \frac{1}{10} \sum_{i=1}^{10} \phi(x; \mu_i^{(\text{blue})}, I/5)$

$P(X = x | G = \mathcal{G}_{\text{orange}}) = \frac{1}{10} \sum_{i=1}^{10} \phi(x; \mu_i^{(\text{orange})}, I/5)$

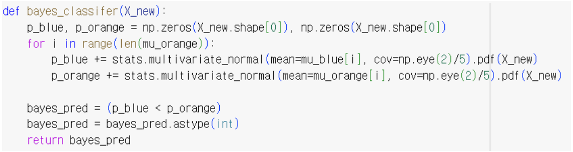
By Bayes' Theorem,

$P(G = \mathcal{G}_{\text{blue}} | X = x) \propto P(X = x | G = \mathcal{G}_{\text{blue}}) P(G = \mathcal{G}_{\text{blue}}) \propto \frac{1}{10} \sum_{i=1}^{10} \phi(x; \mu_i^{(\text{blue})}, I/5)$

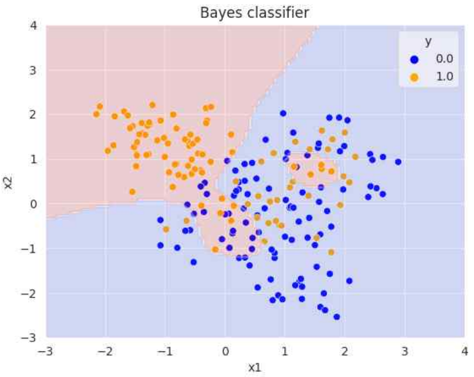
Similarly to orange. Then decision boundary is:

$\frac{1}{10} \sum_{i=1}^{10} \phi(x; \mu_i^{(\text{blue})}, I/5) = \frac{1}{10} \sum_{i=1}^{10} \phi(x; \mu_i^{(\text{orange})}, I/5)$

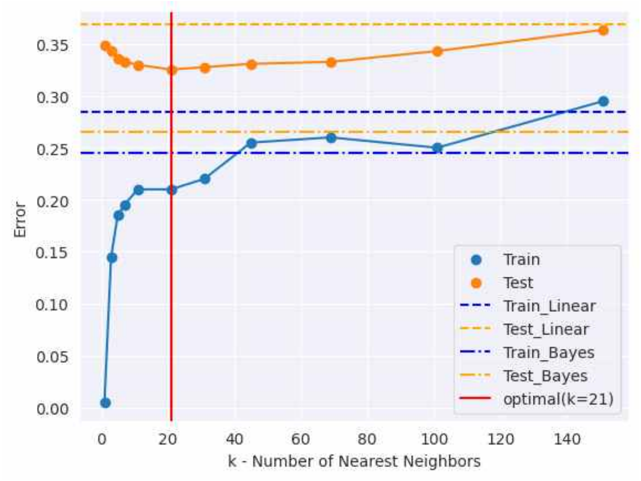
Example: Bayes classifier



```
def bayes_classifer(X_new):
    p_blue, p_orange = np.zeros(X_new.shape[0]), np.zeros(X_new.shape[0])
    for i in range(len(mu_orange)):
        p_blue += stats.multivariate_normal(mean=mu_blue[i], cov=np.eye(2)/5).pdf(X_new)
        p_orange += stats.multivariate_normal(mean=mu_orange[i], cov=np.eye(2)/5).pdf(X_new)

    bayes_pred = (p_blue < p_orange)
    bayes_pred = bayes_pred.astype(int)
    return bayes_pred
```



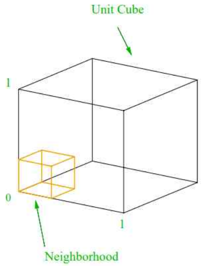
Example: Error plot

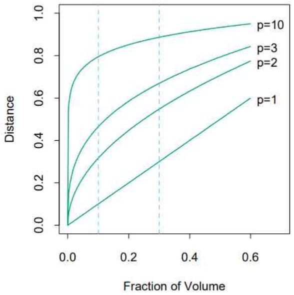


Curse of dimensionality

- **Edge effect problem**

  Consider a fraction r of the unit volume, the expected edge length will be $e_p(r) = r^{\frac{1}{p}}$





Curse of dimensionality

Example 1) Consider N data points **uniformly distributed in a p-dimensional unit ball** centered at the origin. Suppose we consider a **nearest-neighbor estimate** at the origin.

Then, median distance from the origin to the closest data point d(N,p):

Let Y be distance from the origin to closest data. Then, $1 - F_Y(d) = P(Y \geq d) = (1 - d^p)^N$

$0.5 = 1 - (1 - d^p)^N, \; d(N,p) = (1 - 0.5^{\frac{1}{N}})^{\frac{1}{p}}$

For N = 500, p = 10 , d(p,N) $\approx$ 0.52. Hence most data points are closer to the boundary of the sample space than to any other data point.

Such neighborhoods are no longer "local".

Curse of dimensionality

Example 2) Consider a linear model $Y = X^T\beta + \epsilon,$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$

For arbitrary test point $x_0,\ \hat{y}_0 = x_0^T\hat{\beta} = x_0^T\beta + \sum_{i=1}^N l_i(x_0)\epsilon_i$ where $l_i(x_0) = (\mathtt{X}(\mathtt{X}^T\mathtt{X})^{-1}x_0)_i,$ because $\sum_{i=1}^N l_i(x_0)\epsilon_i = (\mathtt{X}(\mathtt{X}^T\mathtt{X})^{-1}x_0)^T\epsilon$ and $\beta + (\mathtt{X}^T\mathtt{X})^{-1}\mathtt{X}^T(\mathtt{Y} - \mathtt{X}\beta) = \hat{\beta}$. Since under this model the least squares estimates are unbiased, we find that

$$
\begin{aligned}
EPE(x_0) &= \mathbb{E}_{y_0|x_0}\mathbb{E}_{\mathcal{T}}(y_0 - \hat{y}_0)^2 \\
&= \mathbb{E}_{y_0|x_0}\mathbb{E}_{\mathcal{T}}(y_0 - \mathbb{E}_{\mathcal{T}}(\hat{y}_0) + \mathbb{E}_{\mathcal{T}}(\hat{y}_0) - \hat{y}_0)^2 \\
&= Var_{\mathcal{T}}(\hat{y}_0) + \mathbb{E}_{y_0|x_0}(y_0 - x_0^T\beta + x_0^T\beta - \mathbb{E}_{\mathcal{T}}(\hat{y}_0))^2 \\
&= \sigma^2 x_0^T\mathbb{E}_{\mathcal{T}}[(\mathtt{X}^T\mathtt{X})^{-1}]x_0 + Var(y_0|x_0) + \cancelto{0}{bias^2(\mathbb{E}_{\mathcal{T}}(\hat{y}_0))} \\
&= \sigma^2(1 + x_0^T\mathbb{E}_{\mathcal{T}}[(\mathtt{X}^T\mathtt{X})^{-1}]x_0)
\end{aligned}
$$

Curse of dimensionality

And if N is large and $\mathcal{T}$ were selected at random and assumes $\mathbb{E}(X) = 0$. Then, $\mathtt{X}^T\mathtt{X} \to NCov(X)$ and

$$\begin{aligned}
\mathbb{E}_{x_0} EPE(x_0) &\approx \mathbb{E}_{x_0} x_0^T Cov^{-1}(X) x_0 \sigma^2/N + \sigma^2 \\
&= \mathbb{E}_{x_0} tr(Cov^{-1}(X) x_0 x_0^T \sigma^2/N) + \sigma^2 \\
&= tr(Cov^{-1}(X) \mathbb{E}_{x_0}(x_0 x_0^T) \sigma^2/N) + \sigma^2 \\
&= tr(Cov^{-1}(X) Cov(x_0)) \sigma^2/N + \sigma^2 = \frac{\sigma^2}{N} \boxed{p} + \sigma^2
\end{aligned}$$

If N is large and/or σ^2 is small, this growth in variance is **negligible**. By imposing some heavy restrictions on the class of models being fitted, **we have avoided the curse of dimensionality.**

Kernel Methods

- Specifying the **nature of the local** neighborhood by kernel $K_\lambda(x_0, x)$
- In general, we can define a local regression estimate by

$$\hat{\theta} = argmin_\theta RSS(f_\theta, x_0) = argmin_\theta \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - f_\theta(x_i))^2$$

- Example of $f_\theta(x)$
  1. $f_\theta(x) = \theta_0$ (Nadaraya-Watson)
  2. $f_\theta(x) = \theta_0 + \theta_1 x$ (Local regression)

- In Nadaraya-Watson, $RSS(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - \theta_0)^2$

$$\frac{\partial}{\partial \theta_0} RSS(f_\theta, x_0) = -2\sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - \theta_0) = 0, \ \hat{\theta} = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}$$

Furthermore, define $K_k(x_0, x_i) = I(\|x_i - x_0\| \leq r_k)$ where r_k is k-th closest distance to x_0 in $\mathcal{T}$, $f_{\hat{\theta}}(x_0) = \frac{\sum_{i=1}^{N} I(\|x_i - x_0\| \leq r_k) y_i}{K} = \frac{1}{K} \sum_{x_i \in N_k(x_0)} y_i$

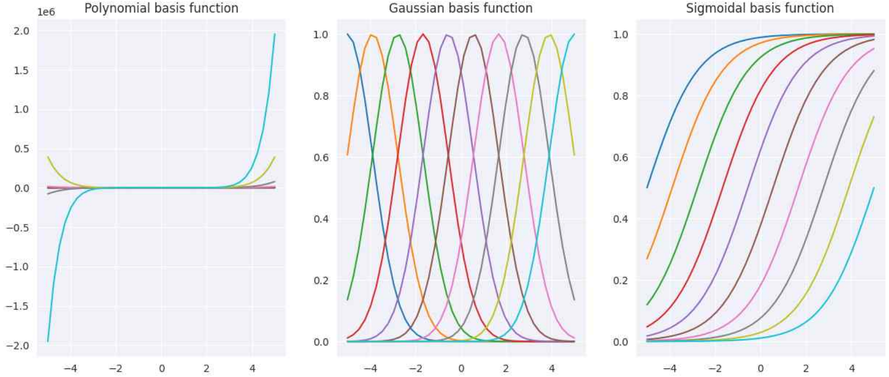
Basis functions

- $f_\theta(x_0) = \theta_0 + \sum_{j=1}^{M-1} \theta_j \phi_j(x_0)$ where ϕ_j is basis functions.$(\phi_j(x) : \mathbb{R}^p \rightarrow \mathbb{R})$
- Let $\phi_0(x) = 1$, then $f_\theta(x_0) = \theta^T \phi(x_0)$, $\theta = (\theta_0, \ldots, \theta_{M-1})^T, \ \phi = (\phi_0, \ldots, \phi_{M-1})^T$

- **Thm.** Consider a model $Y = \theta^T \phi(X) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
  Then maximum likelihood:
  $\hat{\theta} = argmax_\theta P(Y|X, \theta, \sigma^2) = \prod_{i=1}^{N} \mathcal{N}(y_i; \theta^T \phi(x_i), \sigma^2)$

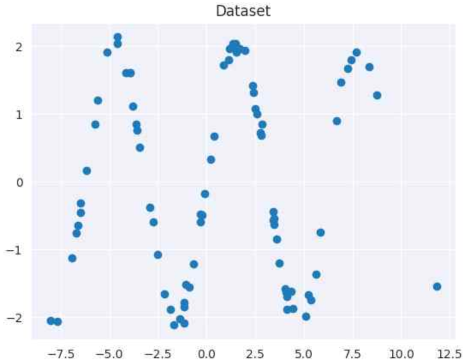
$$= (\Phi^T \Phi)^{-1} \Phi^T Y \quad \text{where} \quad \Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N) \end{pmatrix} \in \mathbb{R}^{N \times M}$$

Basis functions examples



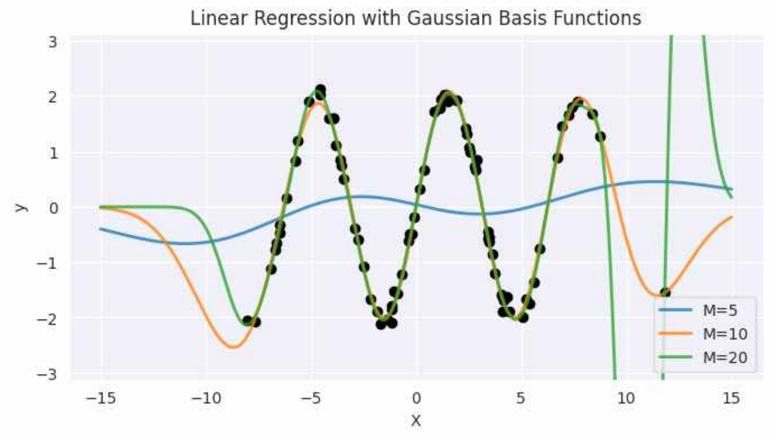
Basis functions examples

```python
N = 80
X = 5*np.random.randn(N,1)
y = 2 * np.sin(X).flatten() + np.random.normal(0, 0.1, X.shape[0]) # add noise
```



```python
def gaussian_plot(M=10):
    basis_means = np.linspace(np.min(X), np.max(X), M)
    scale = basis_means[1] - basis_means[0]
    Phi = np.zeros((N,M)) # (number of samples) * (number of bases)

    for m in range(M):
        Phi[:,m] = gaussian_basis(X,basis_means[m],s=scale).reshape(N,)

    W = np.linalg.inv(Phi.T@Phi)@Phi.T@y
    W = W.reshape(-1,1)

    Phi_pred = np.zeros((1000,M))

    for m in range(M):
        Phi_pred[:,m] = gaussian_basis(X_pred,basis_means[m],s=scale).reshape(1000,)

    plt.plot(X_pred, Phi_pred@W, linewidth=2, label="M="+str(M),alpha=0.8)
```

Basis functions examples

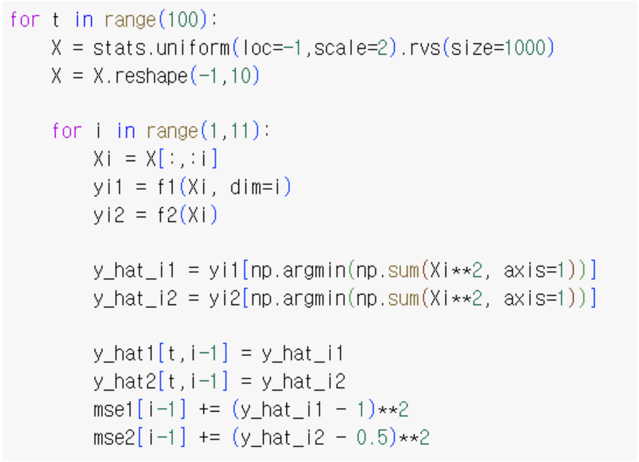


Bias-Variance Decomposition: MSE

- $$\begin{aligned} MSE(x_0) &= \mathbb{E}_{\mathcal{T}}(f(x_0) - \hat{y}_0)^2 \\ &= \mathbb{E}_{\mathcal{T}}[f(x_0) - \mathbb{E}_{\mathcal{T}}(\hat{y}_0) + \mathbb{E}_{\mathcal{T}}(\hat{y}_0) - \hat{y}_0]^2 \\ &= Var_{\mathcal{T}}(\hat{y}_0) + bias^2(\hat{y}_0) \end{aligned}$$
- Case1) Bias dominates, $Y = f(X) = \exp(-8\|X\|^2)$
- Case2) Variance dominates, $Y = f(X) = \frac{1}{2}(x_1 + 1)^3$

➢ Use 1-NN to predict at origin where $X \sim Unif[-1, 1]^p$

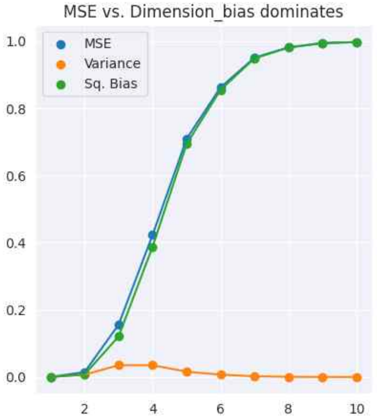
Bias-Variance Decomposition: MSE



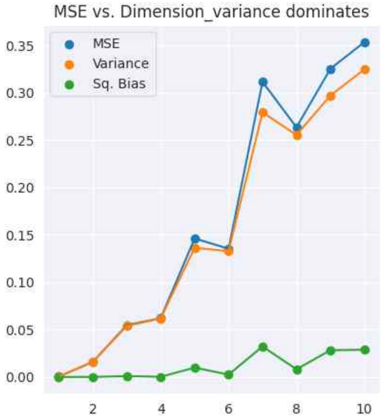
```python
for t in range(100):
    X = stats.uniform(loc=-1,scale=2).rvs(size=1000)
    X = X.reshape(-1,10)

    for i in range(1,11):
        Xi = X[:,:i]
        yi1 = f1(Xi, dim=i)
        yi2 = f2(Xi)

        y_hat_i1 = yi1[np.argmin(np.sum(Xi**2, axis=1))]
        y_hat_i2 = yi2[np.argmin(np.sum(Xi**2, axis=1))]

        y_hat1[t,i-1] = y_hat_i1
        y_hat2[t,i-1] = y_hat_i2
        mse1[i-1] += (y_hat_i1 - 1)**2
        mse2[i-1] += (y_hat_i2 - 0.5)**2
```

Bias-Variance Decomposition: MSE

$Y = f(X) = \exp(-8\|X\|^2)$



$Y = f(X) = \frac{1}{2}(x_1 + 1)^3$



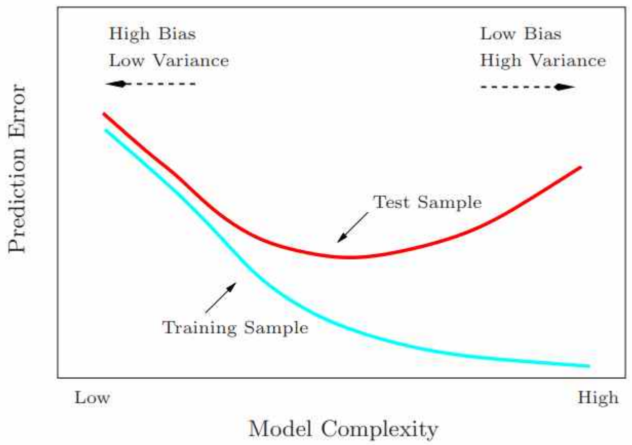
Bias-Variance Decomposition: EPE

- Consider a model $Y = f(X) + \epsilon$ with $\mathbb{E}(\epsilon) = 0,\ Var(\epsilon) = \sigma^2$
- The expected prediction error at x_0 can be decomposed:

$$\begin{aligned} EPE(x_0) &= \mathbb{E}[(Y - \hat{f}(x_0))^2 | X = x_0] \\ &= \sigma^2 + bias^2(\hat{f}(x_0)) + Var_{\mathcal{T}}(\hat{f}(x_0)) \end{aligned}$$

- Too much fitting, the model adapts itself too closely to the training data, and will not generalize well (i.e., **have large test error**
- In contrast, if the model is not complex enough, it will underfit and may have **large bias**, again resulting in poor generalization.

Bias-Variance Decomposition: EPE



Bias-Variance Decomposition: EPE

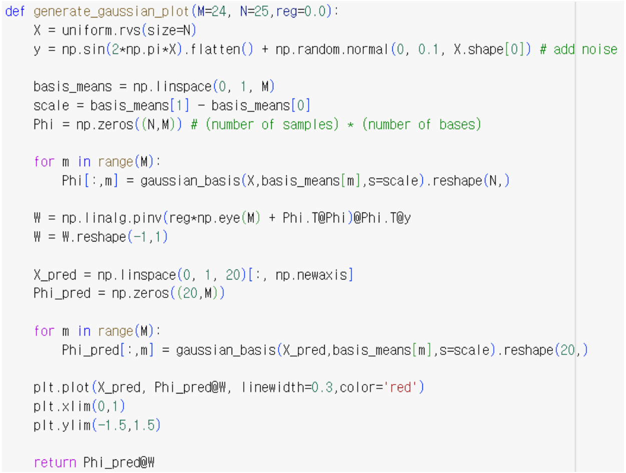
- **Roughness Penalty:** $PRSS(f;\lambda) = RSS(f) + \lambda J(f)$
- Conder a model$Y = \theta^T\phi(X) + \epsilon$ with $\epsilon \sim \mathcal{N}(0,\sigma^2)$, $J(f) = \theta^T\theta$
- Then, penalized least squares: $PRSS(f;\lambda) = \sum_{i=1}^N (y_i - \theta^T\phi(x_i))^2 + \lambda\theta^T\theta$

- **Thm.** For this model,

$$\hat{\theta} = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T \mathrm{Y}$$

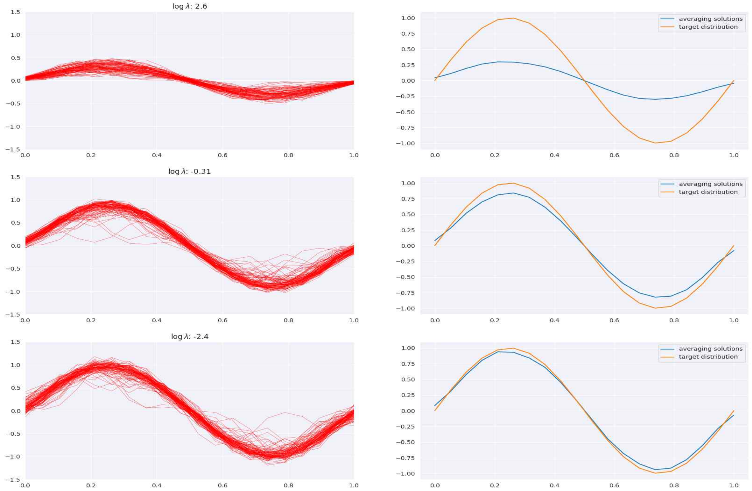
minimizes penalized least-squares.

Bias-Variance Decomposition: EPE



```
def generate_gaussian_plot(M=24, N=25,reg=0.0):
    X = uniform.rvs(size=N)
    y = np.sin(2*np.pi*X).flatten() + np.random.normal(0, 0.1, X.shape[0]) # add noise

    basis_means = np.linspace(0, 1, M)
    scale = basis_means[1] - basis_means[0]
    Phi = np.zeros((N,M)) # (number of samples) * (number of bases)

    for m in range(M):
        Phi[:,m] = gaussian_basis(X,basis_means[m],s=scale).reshape(N,)

    W = np.linalg.pinv(reg*np.eye(M) + Phi.T@Phi)@Phi.T@y
    W = W.reshape(-1,1)

    X_pred = np.linspace(0, 1, 20)[:, np.newaxis]
    Phi_pred = np.zeros((20,M))

    for m in range(M):
        Phi_pred[:,m] = gaussian_basis(X_pred,basis_means[m],s=scale).reshape(20,)

    plt.plot(X_pred, Phi_pred@W, linewidth=0.3,color='red')
    plt.xlim(0,1)
    plt.ylim(-1.5,1.5)

    return Phi_pred@W
```

Bias-Variance Decomposition: EPE


log λ: 2.6
log λ: -0.31
log λ: -2.4
averaging solutions
target distribution