CS 671 Homework 2

Yutong Shao
NetID ys392

October 16, 2022

I consent to the following agreements.

This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

I understand that if I struggle with this assignment that I will reevaluate whether this is the correct class for me to take. I understand that the homework only gets harder.

1 Convexity

1.1

Proof. Convexity implies that $\forall x_1, x_2 \in \mathbb{R}, \text{and } \lambda \in [0,1]$

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2)$$

Take $x_1, x_2 \in \mathbb{R}$. Since $f(x), g(x)$ are both convex function, then we have

$$f(\lambda x_1 + (1-\lambda)x_2) \le \lambda f(x_1) + (1-\lambda) f(x_2)$$
$$g(\lambda x_1 + (1-\lambda)x_2) \le \lambda g(x_1) + (1-\lambda) g(x_2)$$

Therefore,

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) &= f(\lambda x_1 + (1-\lambda)x_2) + g(\lambda x_1 + (1-\lambda)x_2) \\ &\le \lambda f(x_1) + (1-\lambda) f(x_2) + \lambda g(x_1) + (1-\lambda) g(x_2) \\ &= \lambda h(x_1) + (1-\lambda) h(x_2) \end{aligned}$$

Thus, we proved that $h(x)$ is also a convex function. □

1.2

Proof. According to the property of *log* function,

$$log\left(\prod_{i=1}^{k} g_i(x)\right) = \sum_{i=1}^{k} log\left(g_i(x)\right)$$

First, we prove that if $g_i(x)$ is concave and $g_i(x) \geq 0$, then $h(x) = log(g_i(x))$ is also concave in $\mathbb{R}^+$.

Concavity of $g_i(x)$ means that $\forall x_1, x_2 \in \mathbb{R}$, and $\lambda \in [0, 1]$,

$$g_i(\lambda x_1 + (1-\lambda)x_2) \geq \lambda g_i(x_1) + (1-\lambda)g_i(x_2)$$

Also, since $log(x)$ is also concave and monotonically increasing in $\mathbb{R}^+$, we have

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) &= log(g_i(\lambda x_1 + (1-\lambda)x_2)) \\ &\geq log(\lambda g_i(x_1) + (1-\lambda)g_i(x_2)) \quad \longleftarrow \text{monotonically increasing} \\ &\geq \lambda log(g_i(x_1)) + (1-\lambda)log(g_i(x_2)) \quad \longleftarrow \text{concavity of } log(x) \\ &= \lambda h(x_1) + (1-\lambda)h(x_2) \end{aligned}$$

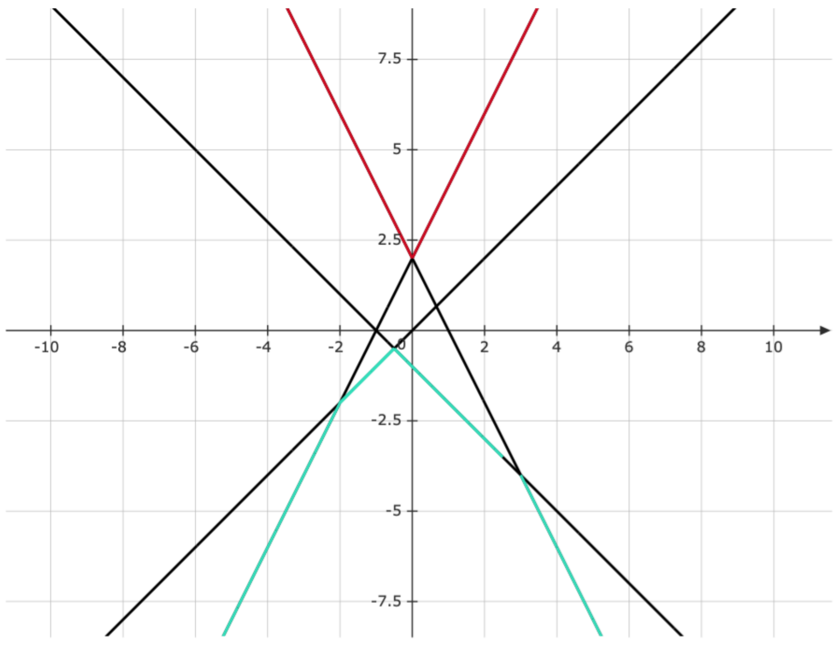
Therefore, we proved that $h(x) = log(g_i(x))$ is concave in $\mathbb{R}^+$.

Since the sum of two concave functions is also concave, we can easily generalize to the conclusion that the sum of k concave functions is also concave, which means

$$log\left(\prod_{i=1}^{k} g_i(x)\right) = \sum_{i=1}^{k} log\left(g_i(x)\right) \text{ is concave}$$

$\square$

1.3



The maximum of convex functions is also convex. The minimum of concave functions is also concave.

1.4

We can prove that $h(x) = max\{f(x), g(x)\}$ is also convex.

Proof. To prove that $\forall x_1, x_2 \in \mathbb{R}$,

$$h(\lambda x_1 + (1-\lambda)x_2) \leq \lambda h(x_1) + (1-\lambda)h(x_2) \quad \forall \lambda \in [0,1]$$

we only need to prove

$$\begin{aligned} f(A) &\leq \lambda h(x_1) + (1-\lambda)h(x_2) \\ \text{and } g(A) &\leq \lambda h(x_1) + (1-\lambda)h(x_2) \end{aligned}$$

where $A = \lambda x_1 + (1-\lambda)x_2$.

$f(x)$ is convex $\Longrightarrow f(A) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$

$g(x)$ is convex $\Longrightarrow g(A) \leq \lambda g(x_1) + (1-\lambda)g(x_2)$

Since $h(x) \geq f(x)$ and $h(x) \geq g(x)$,

$$f(A) \leq \lambda f(x_1) + (1-\lambda)f(x_2) \leq \lambda h(x_1) + (1-\lambda)h(x_2)$$

$g(A)$ follows the same logic.

Thus, we proves that $h(x) = max\{f(x), g(x)\}$ is also convex.

$\square$

1.5

$h(x) = \min\{f(x), g(x)\}$ is not necessarily convex.

A counter example:

Let $f(x) = x$, $g(x) = -x + 1$, and they are both convex function.

We can show that $h(x) = \min\{f(x), g(x)\} = \begin{cases} x, & x \leq \frac{1}{2} \\ -x+1, & x > \frac{1}{2} \end{cases}$ is strictly concave in $x \in [0,1]$.

Take $x_1 = 0, x_1 = 1, \lambda = \frac{1}{2}$, then

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) = h\left(\frac{1}{2} \times 0 + \frac{1}{2} \times 1\right) = h\left(\frac{1}{2}\right) &= \frac{1}{2} \\ > \frac{1}{2} \times h(0) + \frac{1}{2} \times h(1) &= 0 \end{aligned}$$

which clearly violates convexity.

1.6

(a)

Proof.

$$L(\boldsymbol{\omega}, b) = \sum_{i=1}^{n} \log(1 + e^{-y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)}) = \sum_{i=1}^{n} -log\frac{1}{1 + e^{-y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)}}$$

First, we prove that if $f(x)$ and $g(x)$ are both convex function, then $h(x) = f(g(x))$ is also convex.

$$\begin{aligned} h(\lambda x_1 + (1-\lambda)x_2) &= f(\lambda g(x_1) + (1-\lambda)g(x_2)) \\ &\leq \lambda f(g(x_1)) + (1-\lambda)f(g(x_2)) \\ &= \lambda h(x_1) + (1-\lambda)h(x_2) \end{aligned}$$

let

$$\begin{aligned} f_1(\boldsymbol{\omega}, b) &= -y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b) \\ f_2(x) &= e^x \\ f_3(x) &= \frac{1}{1+x} \\ f_4(x) &= -logx \\ \Longrightarrow L(\boldsymbol{\omega}, b) &= f_4(f_3(f_2(f_1(\boldsymbol{\omega}, b)))) \end{aligned}$$

$f_1(\boldsymbol{\omega}, b)$ is convex since it is affine.

$f_2(x)$ is also convex since its second order derivative e^x is greater than zero.

$f_3(x)$ is convex since its second order derivative $2(1+x)^{-3}$ is greater than zero in $x > 0$. And notice that $f_2(x) > 0 \quad \forall x \in \mathbb{R}$.

$f_4(x)$ is convex which is a hint of question 1.

And the sum of convex function is also convex. Therefore, $L(\boldsymbol{\omega}, b)$ is convex.

□

(b)

Proof.

$$L(\boldsymbol{\omega}, b, C) = \frac{1}{2}||\boldsymbol{\omega}||_2^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)), \quad C \geq 0$$

1) First we prove that $\ell 2$-norm is convex.

$\nabla||\boldsymbol{\omega}||_2^2 = 2||\boldsymbol{\omega}||_2$, $H||\boldsymbol{\omega}||_2^2 = 2 > 0$

Therefore, the first term of the function is convex.

2) Next, we prove the second term is also onvex.

Clearly, $1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)$ is affine which is both concave and convex. And we have already proved that the maximum of convex function is also convex. Thus, $\max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b))$ is also convex. Note that sum of convex function is also convex. Thus, the second term is convex.

Therefore, we proved that $L(\boldsymbol{\omega}, b, C)$ is convex.

□

1.7

$$L(\boldsymbol{\omega}, b, C) = \|\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)\|_2^2 + C\|\boldsymbol{\omega}\|_1, \quad C \geq 0$$

$L(\boldsymbol{\omega}, b, C)$ is convex.

Proof. 1) We have proved in 1.6 that $\ell 2$-norm is convex. And note that $\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)$ is a set of affine functions which is both convex and concave. Thus, $\|\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)\|_2^2$ is convex.

2) Next we prove $\ell 1$-norm is convex.

According to the properties of $\ell 1$-norm and absolute value,

$$\begin{aligned}
\|\boldsymbol{x}\|_1 &= \sum_{i=1}^n |x_i|, \quad |x+y| \leq |x| + |y| \\
\therefore \|\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}\|_1 &= \sum_{i=1}^n |\lambda x_i + (1-\lambda) y_i| \\
&\leq \sum_{i=1}^n (|\lambda x_i| + |(1-\lambda) y_i|) \\
&= \sum_{i=1}^n (\lambda |x_i| + (1-\lambda)|y_i|) \\
&= \lambda \sum_{i=1}^n |x_i| + (1-\lambda) \sum_{i=1}^n |y_i| \\
&= \lambda \|\boldsymbol{x}\|_1 + (1-\lambda)\|\boldsymbol{y}\|_1
\end{aligned}$$

Thus, $\ell 1$-norm is convex.

Therefore, we proved that the lasso risk function is convex. □

2 Gradient Descent and Newton's Method

2.1

(a)

$$\begin{aligned}\nabla f(\boldsymbol{x}) &= \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}\right) \\ &= (4x_1 - 4 + 2x_2,\ 6x_2 - 6 + 2x_1)\end{aligned}$$

(b)

$$\begin{aligned}Hf(\boldsymbol{x}) = Jf(\boldsymbol{x}) &= \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 x_2} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} 4 & 2 \\ 2 & 6 \end{pmatrix} \\ D(Hf(\boldsymbol{x})) &= \begin{vmatrix} 4 & 2 \\ 2 & 6 \end{vmatrix} = 20 > 0\end{aligned}$$

Because the determinant of Hessian matrix is greater than 0, there exist a extreme minimum value.

(c) Let $\nabla f(\boldsymbol{x}) = 0$,

$$\begin{cases} 4x_1 - 4 + 2x_2 & = 0 \\ 6x_2 - 6 + 2x_1 & = 0 \end{cases} \Longrightarrow \begin{cases} x_1 & = \frac{3}{5} = 0.6 \\ x_2 & = \frac{4}{5} = 0.8 \end{cases}$$

Since $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} > 0$ and $D(Hf(\boldsymbol{x})) > 0$, there is a local minimum at $(\frac{3}{5}, \frac{4}{5})^T$, i.e. $x^* = (0.6, 0.8)^T$.

Therefore,

$$\min f(\boldsymbol{x}) = f((0.6, 0.8)) = 9.4$$

2.2

Let $\boldsymbol{x}_t = (a_t, b_t)$, then $\nabla f(\boldsymbol{x}_t) = (4a_t - 4 + 2b_t, 6b_t - 6 + 2a_t)$.

$$\begin{aligned}\boldsymbol{x}_{t+1} &= \boldsymbol{x}_t - \alpha \nabla f(\boldsymbol{x}_t) \\ &= (a_t, b_t) - \alpha\,(4a_t - 4 + 2b_t,\ 6a_t - 6 + 2b_t) \\ &= (a_t - 4\alpha a_t + 4\alpha - 2\alpha b_t, b_t - 6\alpha b_t + 6\alpha - 2\alpha a_t) \\ &= ((1-4\alpha)a_t + 4\alpha - 2\alpha b_t, (1-6\alpha)b_t + 6\alpha - 2\alpha a_t) \\ &= (a_t, b_t)\begin{pmatrix} 1-4\alpha & -2\alpha \\ -2\alpha & 1-6\alpha \end{pmatrix} + (4\alpha, 6\alpha) = (a_{t+1}, b_{t+1}) \\ &= \boldsymbol{x}_t A + B\end{aligned}$$

where $A = \begin{pmatrix} 1-4\alpha & -2\alpha \\ -2\alpha & 1-6\alpha \end{pmatrix}$, $B = (4\alpha, 6\alpha)$

Repeat the steps above, we can get

$$\begin{aligned}
\boldsymbol{x}_{t+2} &= \boldsymbol{x}_{t+1} - \alpha \nabla f(\boldsymbol{x}_{t+1}) \\
&= (\boldsymbol{x}_t A + B)A + B \\
&= \boldsymbol{x}_t A^2 + BA + B \\
\cdots \\
\boldsymbol{x}_n &= \boldsymbol{x}_t A^n + B\left(I + A + A^2 + \cdot + A^n\right) \\
\text{where } |A - \lambda I| &= \begin{vmatrix} 0.6-\lambda & -0.2 \\ -0.2 & 0.4-\lambda \end{vmatrix} = \lambda^2 - \lambda + 0.2 = 0 \\
\Longrightarrow \text{eigenvalues of A: } \lambda_1 &= \frac{1}{10}\left(5+\sqrt{5}\right) < 1, \quad \lambda_2 = \frac{1}{10}\left(5-\sqrt{5}\right) < 1
\end{aligned}$$

According to hints, if all eigenvalues of A have absolute value smaller than 1, then $\lim_{n\to\infty} A^k = 0$, and for a square matrix A, $\lim_{n\to\infty} I + A + A^2 + \cdots + A^n = (I-A)^{-1}(I - A^{k+1})$.

Therefore,

$$\begin{aligned}
\lim_{n\to\infty} \boldsymbol{x}_n &= \lim_{n\to\infty} \boldsymbol{x}_t A^n + A\left(I + A + A^2 + \cdots + A^n\right) \\
&= B(I-A)^{-1}(I - A^{n+1})
\end{aligned}$$

Plug $\alpha = 0.1$ into the limit, we can get

$$\begin{aligned}
\lim_{n\to\infty} \boldsymbol{x}_n &= B(I-A)^{-1} \\
&= (0.4, 0.6)\begin{pmatrix} 0.4 & 0.2 \\ 0.2 & 0.6 \end{pmatrix}^{-1} \\
&= (0.6, 0.8) = \boldsymbol{x}^*
\end{aligned}$$

Therefore, $\boldsymbol{x}_{t+1}$ converges to $\boldsymbol{x}^*$ as $t \to \infty$.

2.3

No, α needs to be sufficiently small to achieve convergence. Specifically, for this question, we can solve the constraints for the learning rate α.

If we want $\boldsymbol{x}_t$ to converge to the optimal when $t \to \infty$, the eigenvalues of matrix A has to have their absolute values smaller than 1.

Therefore, by calculating eigenvalues of A, we can get the constraints below.

$$\begin{aligned}
A &= \begin{pmatrix} 1-4\alpha & -2\alpha \\ -2\alpha & 1-6\alpha \end{pmatrix} \\
|A - \lambda I| &= \begin{vmatrix} 1-4\alpha-\lambda & -2\alpha \\ -2\alpha & 1-6\alpha-\lambda \end{vmatrix} = 0 \\
\Longrightarrow \text{eigenvalues of A: } \lambda_1 &= (-\sqrt{5}-5)\alpha + 1, \quad \lambda_2 = (\sqrt{5}-5)\alpha + 1
\end{aligned}$$

Constraints are

$$\begin{cases} |\lambda_1| & = \left|(-\sqrt{5}-5)\alpha+1\right| < 1 \\ |\lambda_2| & = \left|(\sqrt{5}-5)\alpha+1\right| < 1 \end{cases} \Longrightarrow 0 < \alpha < \frac{2}{5+\sqrt{5}}$$

Thus, $\boldsymbol{x}_t$ will converge to the optimal when $t \to \infty$ only if α satisfies the above condition.

2.4

(a)

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}\right) = (4x_1 - 4 + 2x_2,\ 6x_2 - 6 + 2x_1)$$

$$Hf(\boldsymbol{x}) = \begin{pmatrix} 4 & 2 \\ 2 & 6 \end{pmatrix}$$

Therefore,

$$\begin{aligned} \nabla f(\boldsymbol{x}_0) &= (4,8)^T \\ [Hf(\boldsymbol{x}_0)]^{-1}\nabla f(\boldsymbol{x}_0) &= \frac{1}{20}\begin{pmatrix} 6 & -2 \\ -2 & 4 \end{pmatrix}\begin{pmatrix} 4 \\ 8 \end{pmatrix} = \begin{pmatrix} 0.4 \\ 1.2 \end{pmatrix} \\ \therefore \boldsymbol{x}_1 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} - \begin{pmatrix} 0.4 \\ 1.2 \end{pmatrix} &= \begin{pmatrix} 0.6 \\ 0.8 \end{pmatrix} = \boldsymbol{x}^* \end{aligned}$$

Thus, Newton's method will converge in one iteration.

(b)

$$\begin{aligned} \boldsymbol{x}_1 &= \boldsymbol{x}_0 - 0.1\nabla f(\boldsymbol{x}_0) = (1,2)^T - 0.1(4,8)^T = (0.6,1.2)^T \\ \boldsymbol{x}_2 &= \boldsymbol{x}_1 - 0.1\nabla f(\boldsymbol{x}_1) = (0.6,1.2)^T - 0.1(0.8,2.4)^T = (0.52,0.96)^T \neq \boldsymbol{x}^* \end{aligned}$$

Therefore, Gradient descent will not converge in one iteration.

(c)

Newton's method converges faster.

The major difference between the two methods is that the learning rate of Gradient Descent is fixed at α, while Newton's Method adjust the learning rate according to the second order derivative at each point it arrives at.

If $\boldsymbol{x}_t$ is far away from optimal value, Newton's method will adjust the learning rate to a higher value because the second derivative at that point is higher. As $\boldsymbol{x}_t$ converges near to optimal, the learning rate gets lower.

3 Bias and Variances of Random Forest

3.1

Prove by induction.

Proof. (1) For $n = 1$, which means there is only one feature, random forest can surely classify it correctly by adding one node.

(2) Assume for $n = k$, RF can achieve 100% accuracy if there is no depth limit.

Then for $n = k + 1$, RF can also classify the additional feature correctly by adding one more depth and node.

(3) Therefore, $\forall n \in \mathbb{R}^+$, the statement is true. □

3.2

Proof.

$$\bar{D} = \frac{1}{T}\sum_{i=1}^{T} D_i$$

$$Var\left(\bar{D}\right) = Var\left(\frac{1}{T}\sum_{i=1}^{T} D_i\right) = \frac{1}{T^2}Var\left(\sum_{i=1}^{T} D_i\right)$$

$$\because Corr(D_i, D_i) = \frac{Cov(D_i, D_j)}{\sqrt{Var(D_i)Var(D_j)}} = \rho \text{ and } Var(D_i) = Var(D_j) = \sigma^2$$

$$\therefore Cov(D_i, D_j) = \rho\sigma^2$$

$$\therefore Var\left(\sum_{i=1}^{T} D_i\right) = \sum_{i=1}^{T} Var(D_i) + Cov_{i\neq j}(D_i, D_j) = T\sigma^2 + \left(T^2 - T\right)\rho\sigma^2$$

$$= T^2\rho\sigma^2 + T(1-\rho)\sigma^2$$

$$\therefore \bar{D} = \frac{1}{T^2}\sum_{i=1}^{T} D_i = \frac{1}{T^2}\left[T^2\rho\sigma^2 + T(1-\rho)\sigma^2\right] = \frac{1-\rho}{T}\sigma^2 + \rho\sigma^2$$

□

3.3

$$\lim_{T\to\infty} Var(\bar{D}) = \rho\sigma^2$$

As T $\to \infty$, the first term disappears while $\rho\sigma^2$ remains.

3.4

The *num_of_feature* and *min_sample_size* parameters will control ρ in random forest.

If there are quite a few features and a large sample size, then the samples are more likely to be highly correlated. However, if we have many features and we control the sample size of each feature, the samples will not be highly correlated, which reduces ρ.

4 Coding for SVM

See Coding Appendix

5 Coding for Logistic Regression

5.1

(a) Assmue $\boldsymbol{x} \in \mathbb{R}^p$, and $Y \sim Bernoulli(P(Y = 1 \mid \boldsymbol{x}))$. Y takes the value of 1 or 0 in this question. The model is linear, which means

$$ln\left(\frac{P(Y=1 \mid \boldsymbol{x}, \theta)}{P(Y=0 \mid \boldsymbol{x}, \theta)}\right) = \boldsymbol{\theta}^T \boldsymbol{x}$$

$$\implies \begin{cases} y_i = 1, & P(Y=1 \mid \boldsymbol{x}, \theta) = \frac{e^{\theta^T \boldsymbol{x}}}{1+e^{\theta^T \boldsymbol{x}}} = \frac{e^{y_i \theta^T \boldsymbol{x}}}{1+e^{\theta^T \boldsymbol{x}}} \\ y_i = 0, & P(Y=0 \mid \boldsymbol{x}, \theta) = \frac{1}{1+e^{\theta^T \boldsymbol{x}}} = \frac{e^{y_i \theta^T \boldsymbol{x}}}{1+e^{\theta^T \boldsymbol{x}}} \end{cases}$$

$$\therefore P(Y = y_i \mid \boldsymbol{x}, \theta) = p(y \mid \boldsymbol{x}, \theta) = \frac{e^{y_i \theta^T \boldsymbol{x}}}{1+e^{\theta^T \boldsymbol{x}}}$$

Therefore, the negative log likelihood of $p(y \mid \boldsymbol{x}, \theta)$ is

$$\begin{aligned} J(\theta) &= -log \prod_{i=1}^{n} P(Y = y_i \mid x_i, \theta) = -\sum_{i=1}^{n} log\left(\frac{e^{y_i \theta^T \boldsymbol{x_i}}}{1+e^{\theta^T \boldsymbol{x_i}}}\right) \\ &= -\left[\sum_{i=1}^{n} log\left(e^{y_i \theta^T \boldsymbol{x_i}}\right) - \sum_{i=1}^{n} log\left(1+e^{\theta^T \boldsymbol{x_i}}\right)\right] \\ &= -\sum_{i=1}^{n}\left[y_i \theta^T x_i - log\left(1+e^{\theta^T \boldsymbol{x_i}}\right)\right] \end{aligned}$$

(b)

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_j} &= -\sum_{i=1}^{n}\left(y_i x_{ij} - \frac{e^{\theta^T x_i} \cdot x_{ij}}{1+e^{\theta^T x_i}}\right) \\ &= -\sum_{i=1}^{n} x_{ij}\left(y_i - \frac{1}{1+e^{-\theta^T x_i}}\right) \end{aligned}$$

(c)

$$\begin{aligned} \theta_j &\to \theta_j - \eta \cdot \frac{\partial J(\theta)}{\partial \theta_j} \\ &= \theta_j + \eta \cdot \sum_{i=1}^{n} x_{ij}\left(y_i - \frac{1}{1+e^{-\theta^T x_i}}\right) \end{aligned}$$

5.2

See coding Appendix

5.3

See coding Appendix

6 Boosted Decision Trees and Random Forest

See Appendix

7 Generalized Additive Models

See Appendix

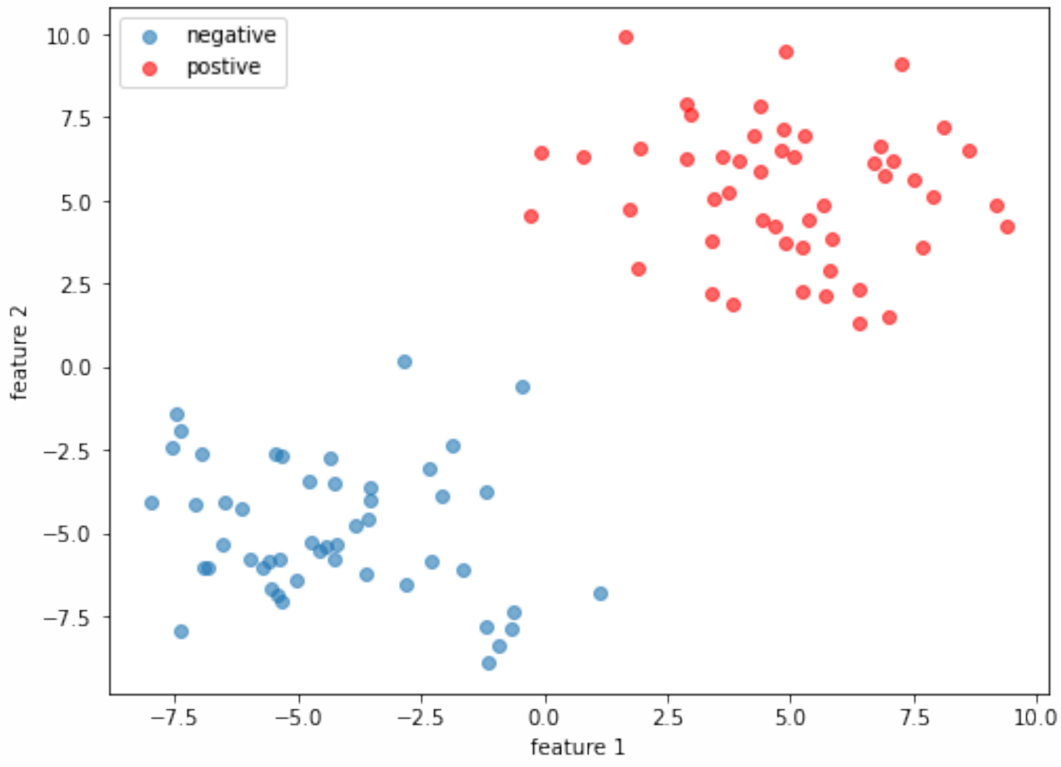
HM 2 CODING

4 Coding for SVM

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

4.1 Data generating and plotting

```python
mu_neg = np.array([-5, -5])
sigma_neg = 5 * np.identity(2)
mu_pos = np.array([5, 5])
sigma_pos = 5 * np.identity(2)

# neg_1, neg_2 = np.random.multivariate_normal(mu_neg, sigma_neg, 50).T
# pos_1, pos_2 = np.random.multivariate_normal(mu_neg, sigma_pos, 50).T

neg = np.random.multivariate_normal(mu_neg, sigma_neg, 50).T
pos = np.random.multivariate_normal(mu_pos, sigma_pos, 50).T

neg_label = -1 * np.ones(50)
pos_label = np.ones(50)

# scatter plot
fig, ax = plt.subplots(figsize=(8,6))
# ax.subplots(figsize=(8,6))
ax.scatter(neg[0], neg[1], alpha=0.6)
ax.scatter(pos[0], pos[1], alpha=0.6, c='r')
ax.set_xlabel('feature 1')
ax.set_ylabel('feature 2')
ax.legend(['negative', 'postive']);
```



4.2 Draw decision boudary

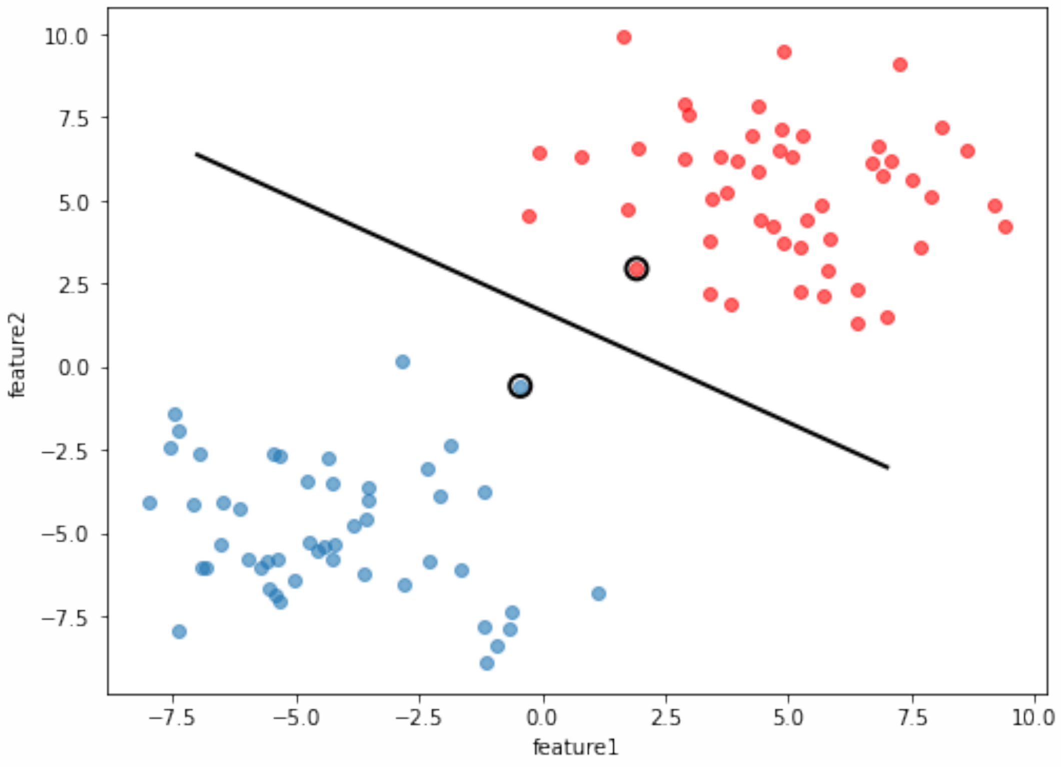
```
from sklearn.svm import SVC
```

```
# concate data
feature1 = np.concatenate((neg[0], pos[0]))
feature2 = np.concatenate((neg[1], pos[1]))
label = np.concatenate((neg_label, pos_label))

X = {
    'feature1': feature1,
    'feature2': feature2
}
# X_train and y_train
df_X = pd.DataFrame(X)
df_y = pd.DataFrame(label)

df_all = pd.merge(left=df_X, right=df_y, left_index=True, right_index=True)
df_all.columns=['feature1','feature2','label']
```

```
# fit the svc model
svc_clf = SVC(kernel='linear', C=1)
svc_clf.fit(df_X, df_y)
# svc_clf.support_vectors_
```

```
SVC(C=1, kernel='linear')
```

```
# plot the decision function

# get the coefficients and intercept of decision boundary
w = svc_clf.coef_[0]
b = svc_clf.intercept_[0]

x0 = np.linspace(-7, 7) # x
decision_boundary = (- w[0] * x0 - b) / w[1]  # line


plt.figure(figsize=(8,6))
plt.plot(x0, decision_boundary, "k-", linewidth=2)
plt.scatter(neg[0], neg[1], alpha=0.6)
plt.scatter(pos[0], pos[1], alpha=0.6, c='r')


# plot support vectors
svs = svc_clf.support_vectors_
plt.scatter(svs[:,0],svs[:,1],s=100,linewidth=2,
                        facecolors='none',edgecolors='k');
plt.xlabel('feature1')
plt.ylabel('feature2');
# plt.grid(True)
```

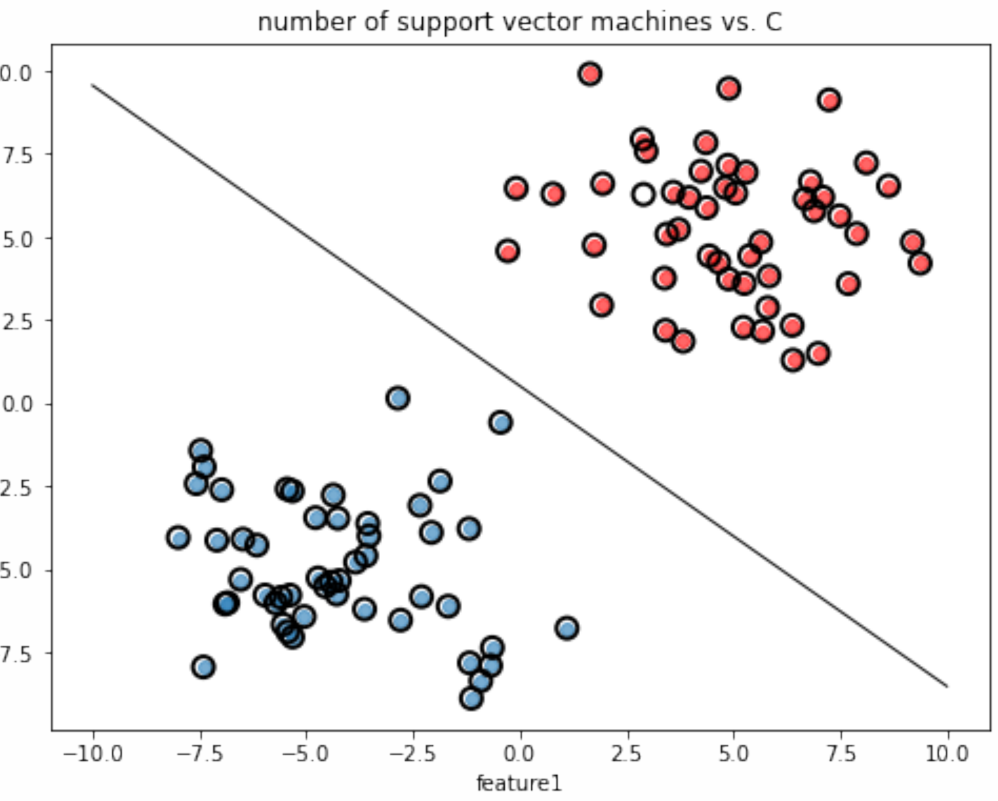


4.3

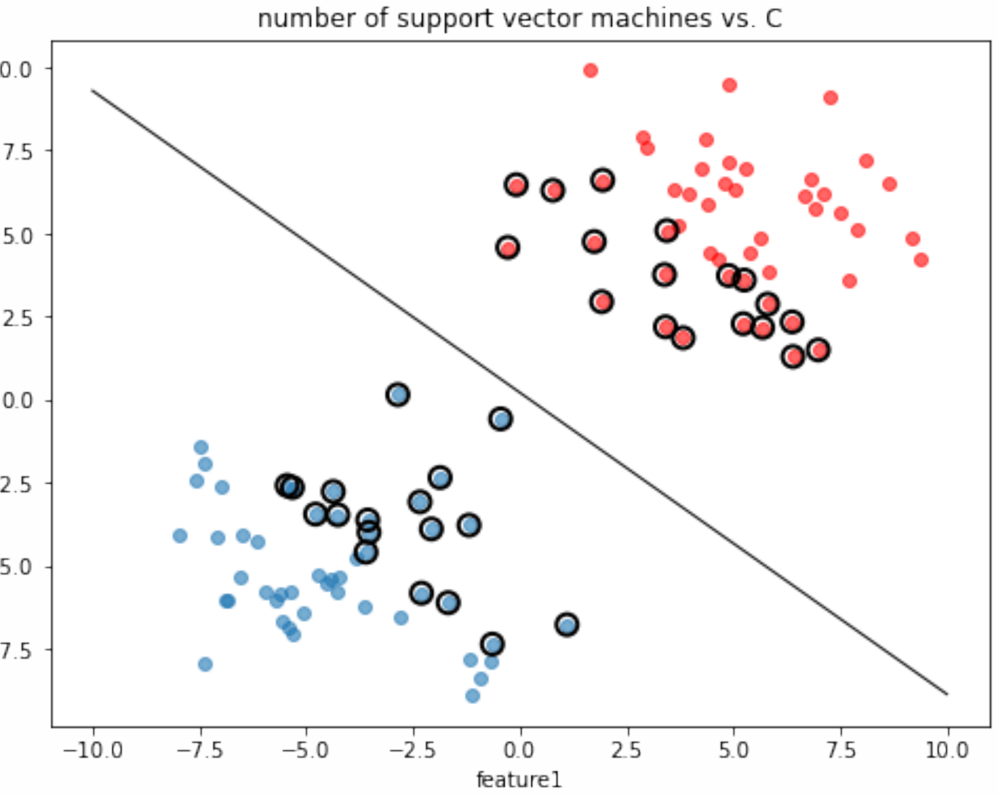
```
def TrainPlotDecisionBoundary(Cs, x0, X_train, y_train,
                              draw_support_vectors, draw_C_svs):
    """"""""""""""
    A function to plot decision boudaries of SVM.
    """"""""""""""

    ws = []  # store coefs
    bs = []  #store intercepts
    svs_list = []
    n_of_svs = []
    # for each C, train model

    for c in Cs:
        svc_clf = SVC(kernel='linear', C=c)
        svc_clf.fit(X_train, y_train)
        # get the coefficients and intercept of decision boundary
        w = svc_clf.coef_[0]
#         ws.append(w)
        b = svc_clf.intercept_[0]
#         bs.append(b)
```

```
        svs = svc_clf.support_vectors_
#         svs_list.append(svs)
        n_svs = len(svs)
        n_of_svs.append(n_svs)
        fig1, ax1 = plt.subplots(figsize=(8,6))
        decision_boundary = (- w[0] * x0 - b) / w[1]  # line
        ax1.plot(x0, decision_boundary, "k-", linewidth=1)
        plt.scatter(X_train.iloc[:50,0], X_train.iloc[:50,1], alpha=0.6)
        plt.scatter(X_train.iloc[51:,0], X_train.iloc[51:,1], alpha=0.6, c='r')

        if draw_support_vectors == True:
#     plt.figure(figsize=(8,6))
#         for w,b in zip(ws, bs):
            # draw vector machines
            plt.scatter(svs[:,0],svs[:,1],s=100,linewidth=2,
                        facecolors='none',edgecolors='k');
            plt.xlabel('feature1')
            plt.ylabel('feature2');
            plt.title('number of support vector machines vs. C')

    if draw_C_svs == True:
        fig2, ax2 = plt.subplots(figsize=(8,6))
        ax2.plot(np.log(Cs), n_of_svs)
        plt.xlabel('C')
        plt.ylabel('number of support vectors')
```

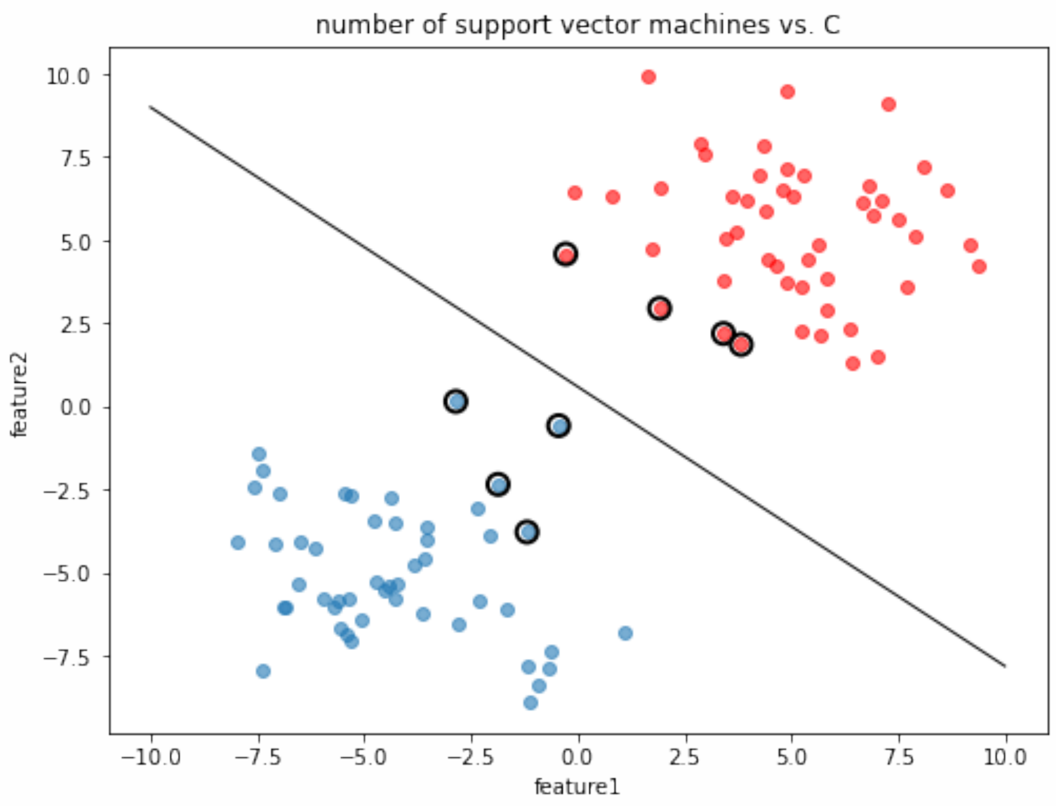
```
Cs = [0.0001, 0.001, 0.01, 1, 50, 100, 500, 1000, 100000]
X_train = df_X
y_train = df_y
x0 = np.linspace(-10,10)

TrainPlotDecisionBoundary(Cs,x0,  X_train, y_train,
                              draw_support_vectors=True, draw_C_svs=True)
```

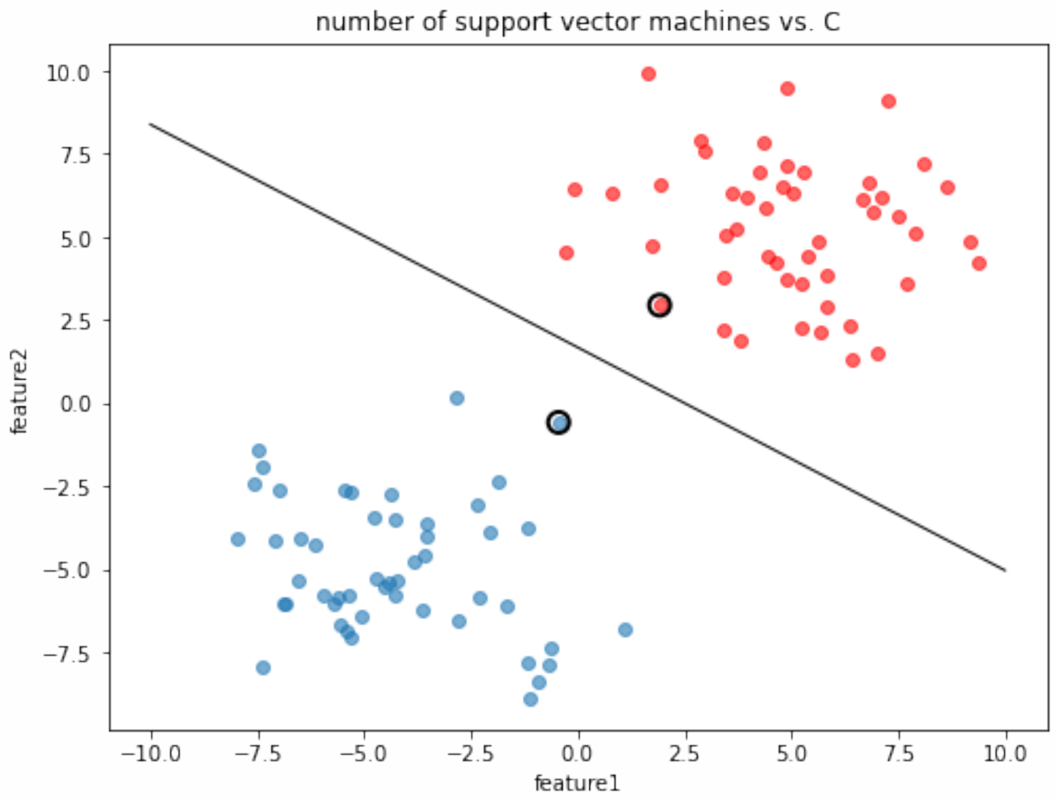


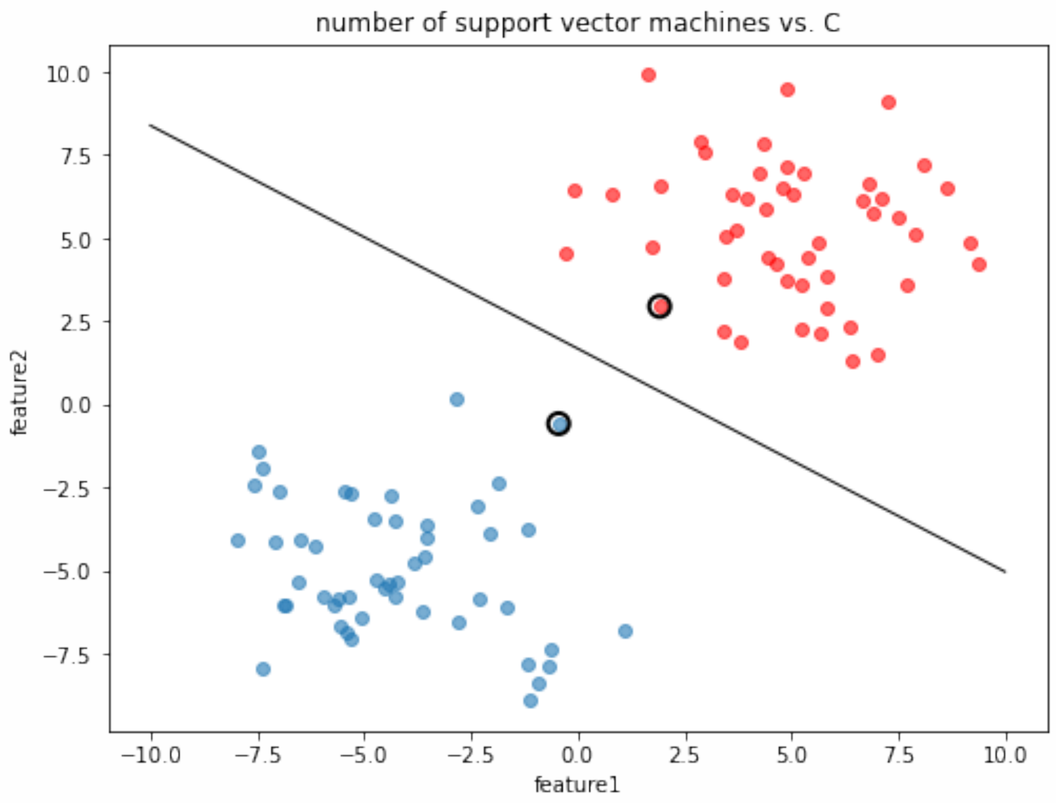
number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

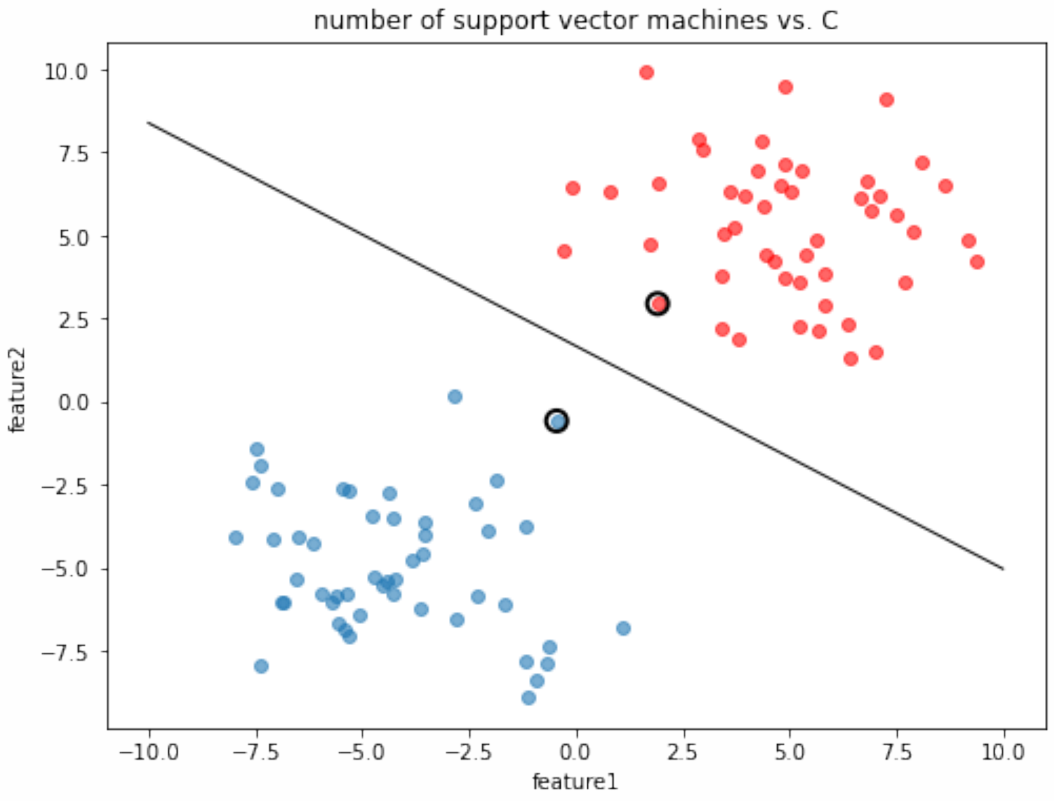


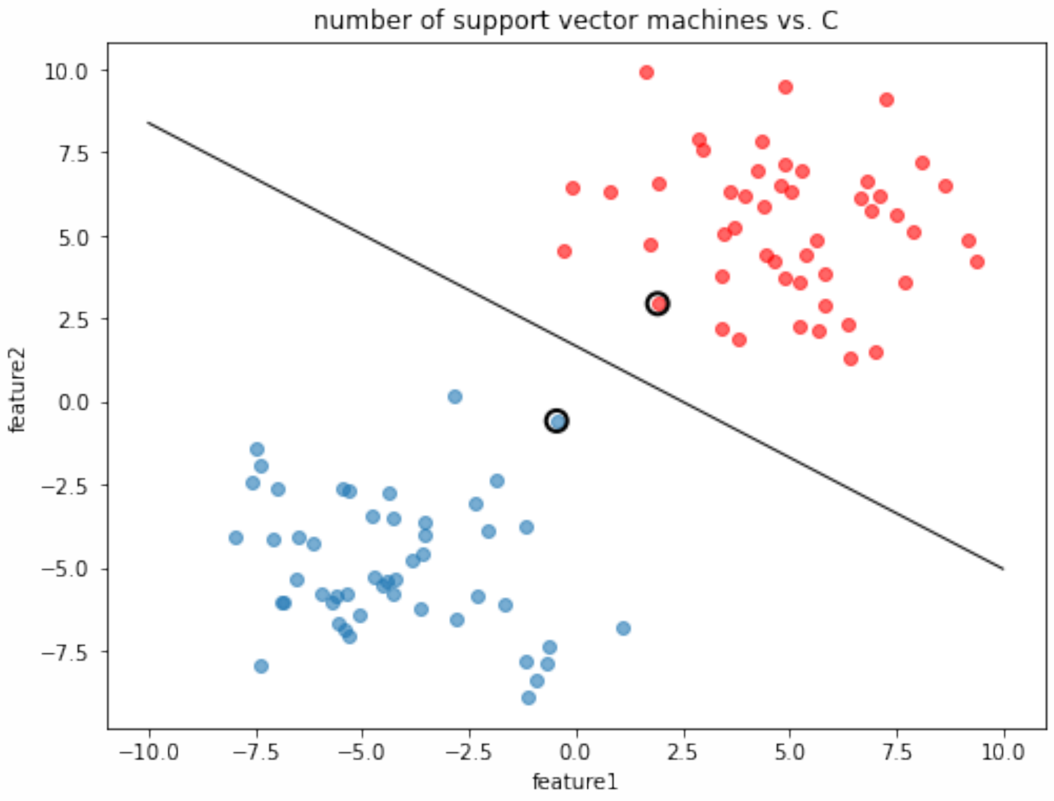
number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

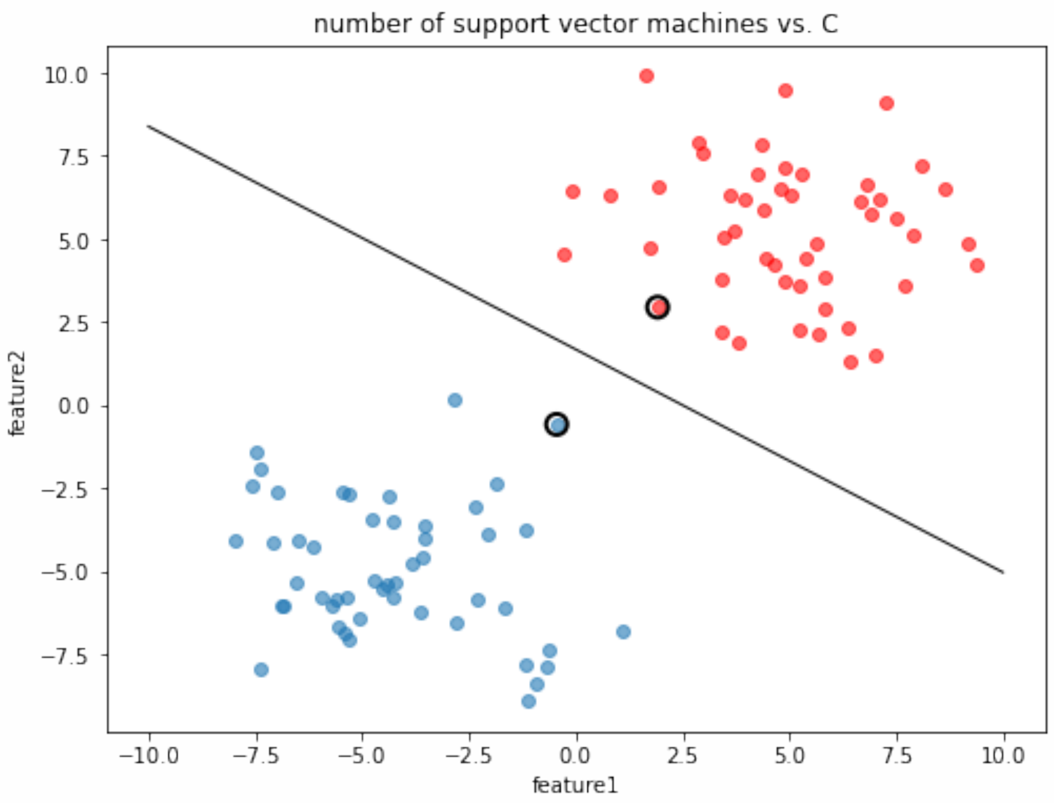

number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

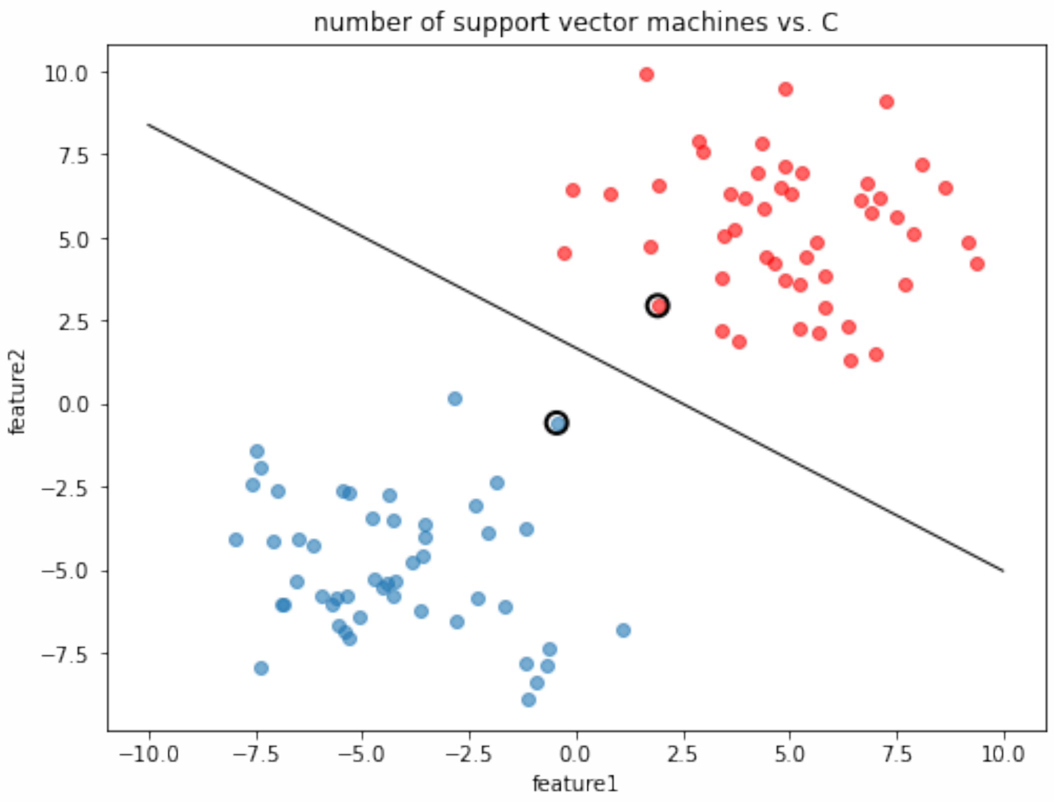

number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

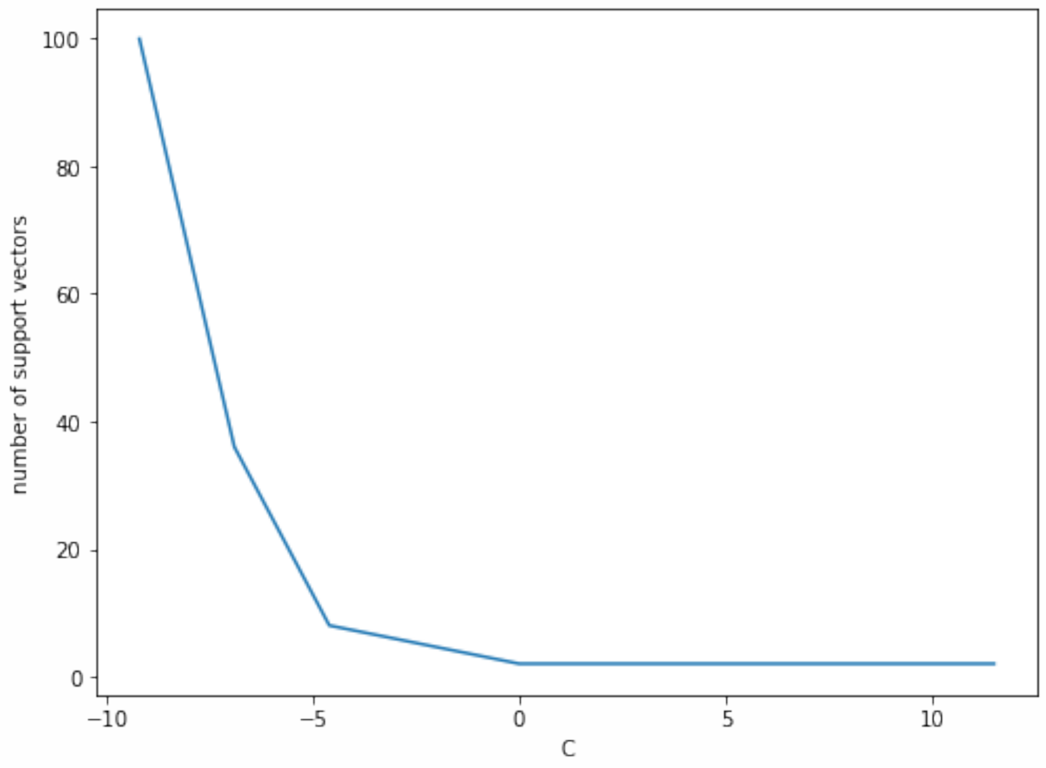

number of support vector machines vs. C
feature1
feature2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


number of support vector machines vs. C
feature1
feature2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


number of support vector machines vs. C
feature1
feature2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


number of support vector machines vs. C
feature2
feature1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



The number of support vectors decreases as C increases.

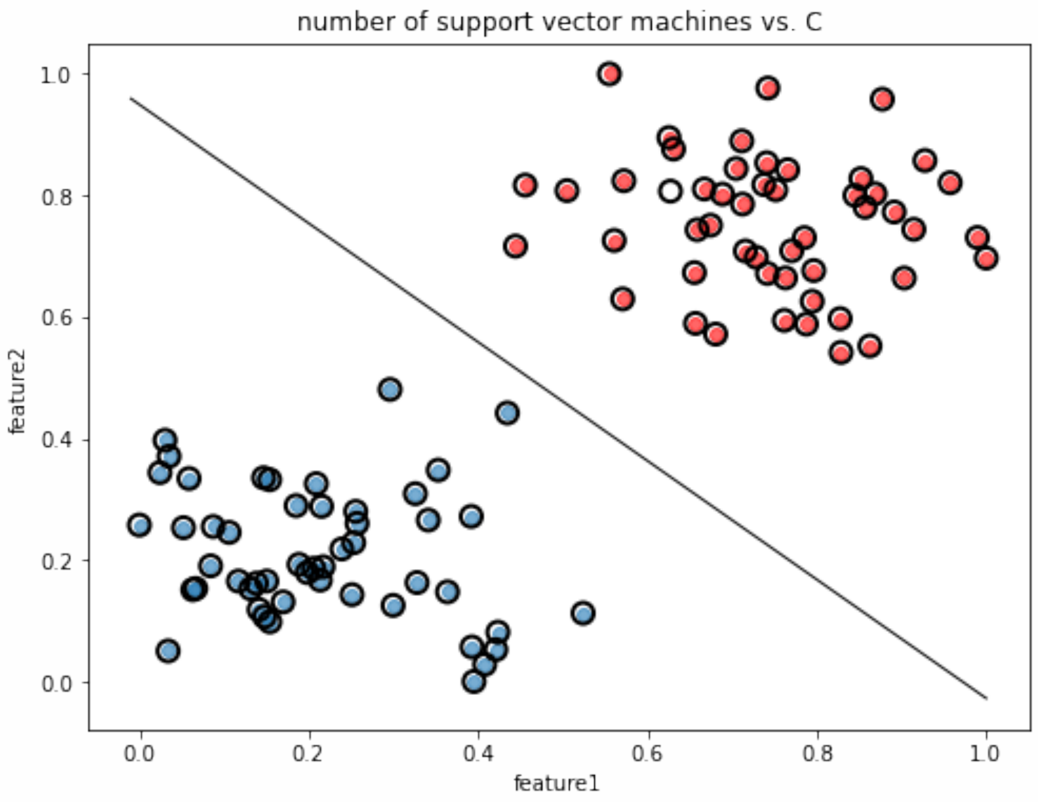
Because C identifies the penalty of misclassification. The higher C is, the larger the penalty will be. Thus, when C is relatively higher, the model will reduce the number of the violations so the margin is narrower, so there are fewer support vectors. When C is smaller, the penalty reduces, and margin is wider which allows more support vectors.

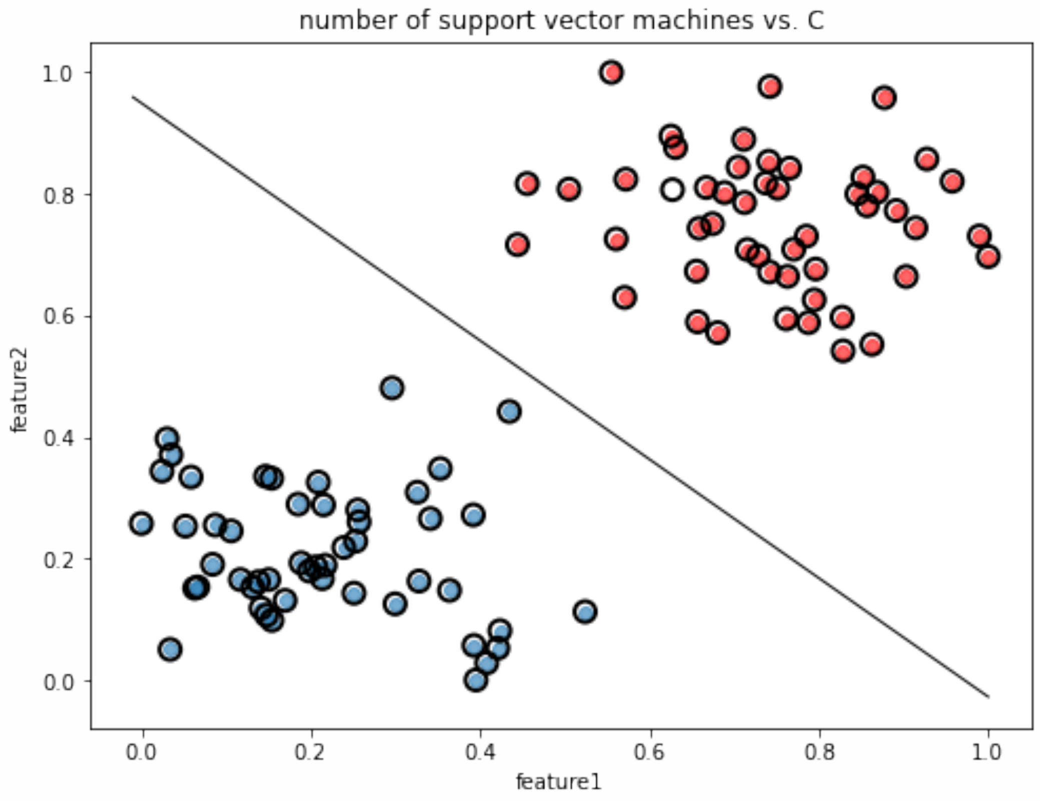
4.4 Rescale data

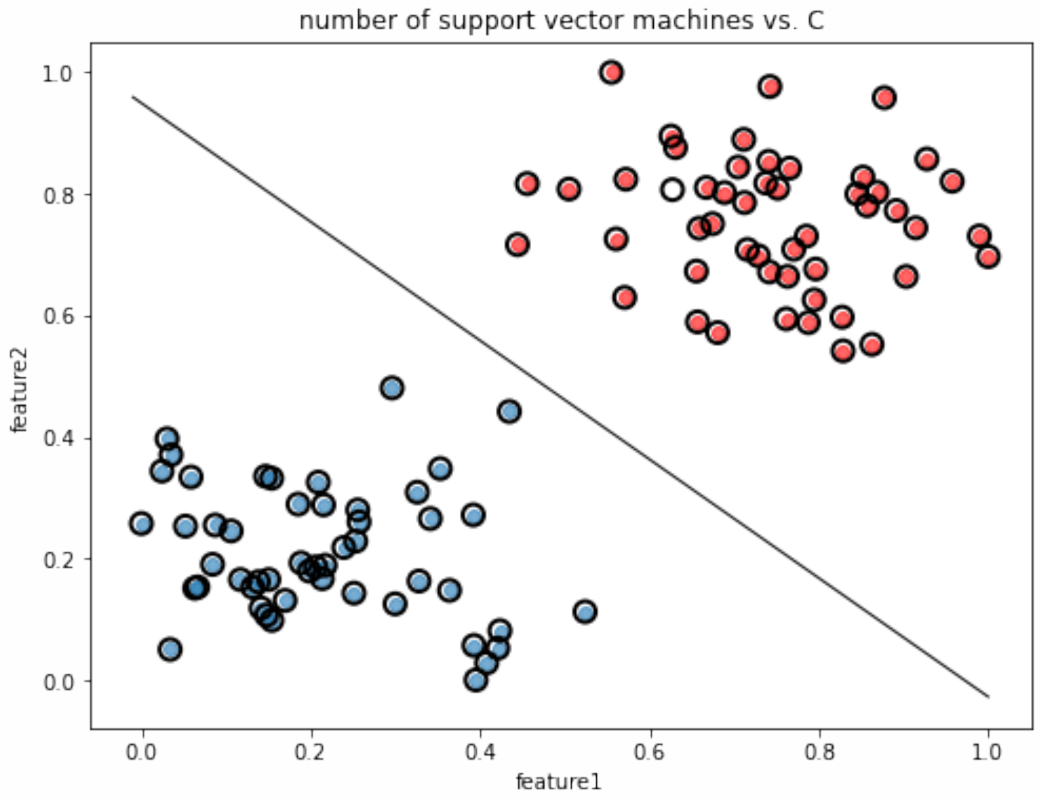
```
# from sklearn.preprocessing import MinMaxScaler

X_train = df_X
y_train = df_y
f1 = X_train.iloc[:, 0]
f2 = X_train.iloc[:, 1]
# min_max_scaler = MinMaxScaler()
min_f1 = min(f1)
max_f1 = max(f1)
min_f2 = min(f2)
max_f2 = max(f2)

f1_scaled = (f1 - min_f1) / (max_f1 - min_f1)
f2_scaled = (f2 - min_f2) / (max_f2 - min_f2)
```

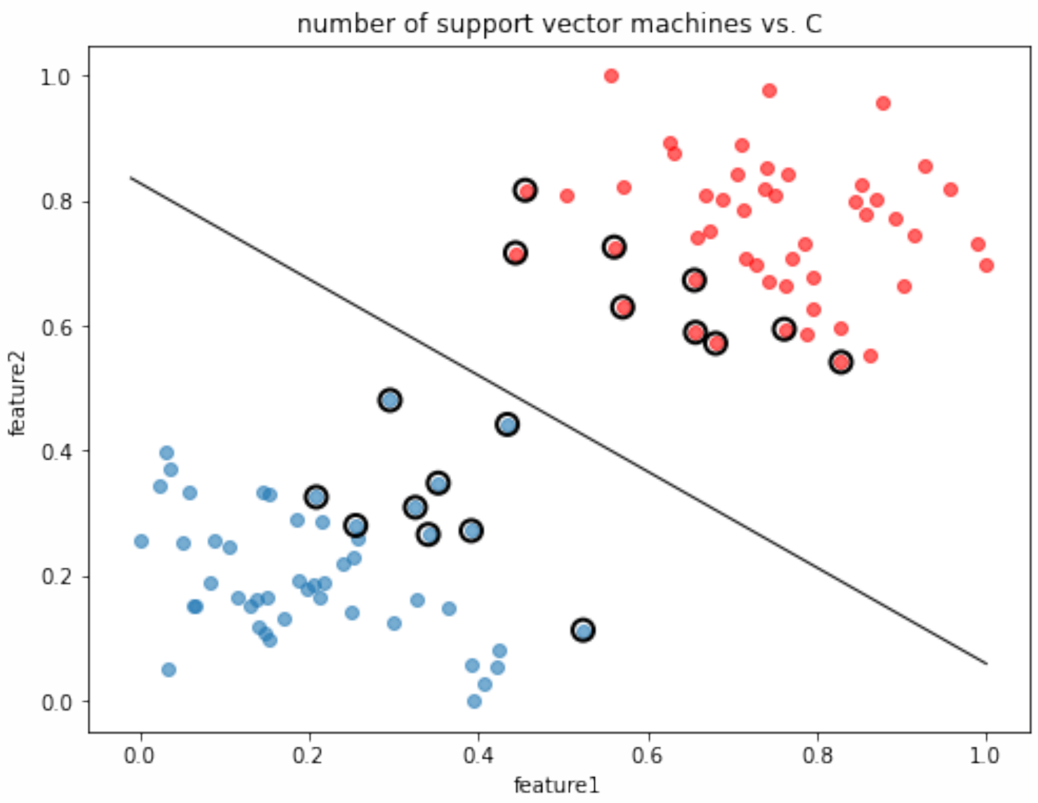
```
scaled = {}
scaled['f1_scaled'] = f1_scaled
scaled['f2_scaled'] = f2_scaled
X_train_scaled = pd.DataFrame(scaled)

# X_train_scaled
```

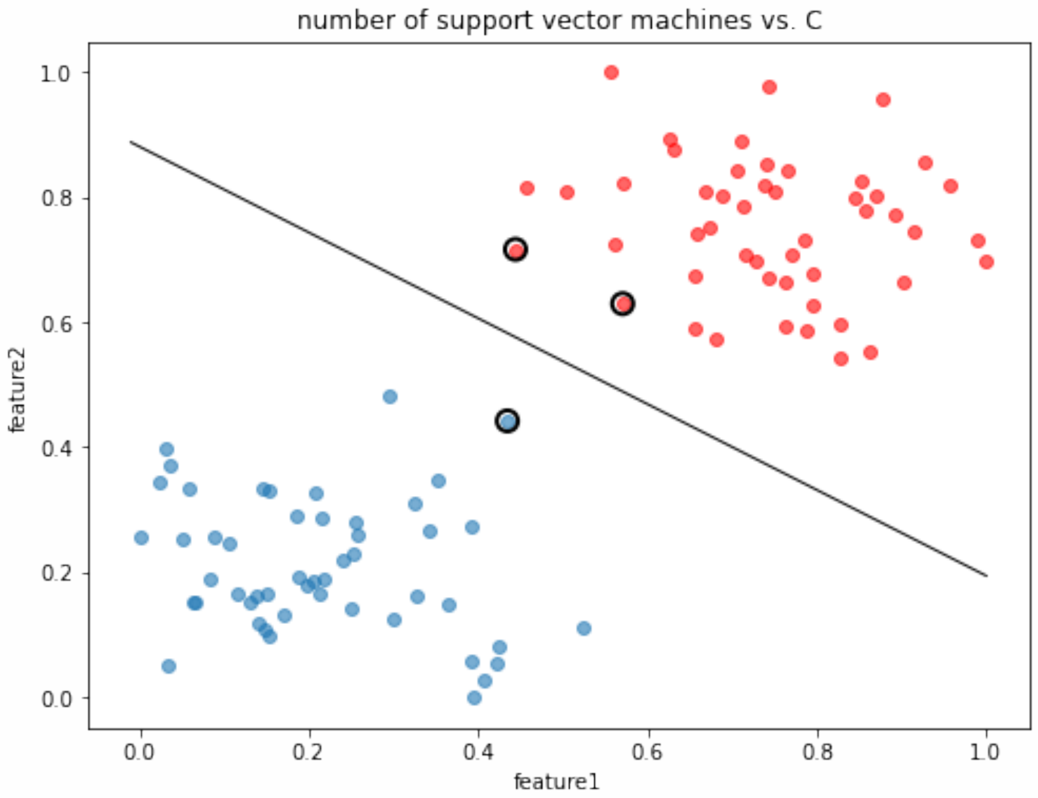
```
Cs = [0.0001, 0.001, 0.01, 1, 50, 100, 500, 1000, 100000]
# X_train = X_train_scaled
# y_train = df_y
x0 = np.linspace(-0.01,1)
TrainPlotDecisionBoundary(Cs, x0, X_train_scaled, df_y,
                              draw_support_vectors=True, draw_C_svs=True)
```

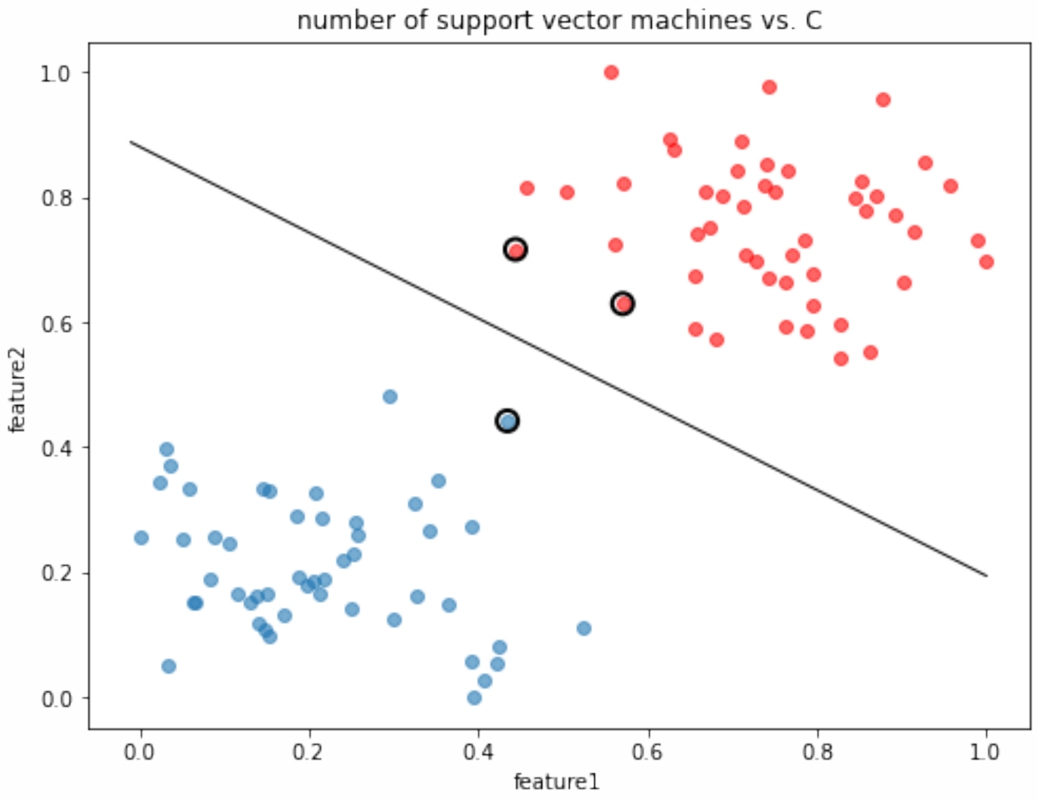


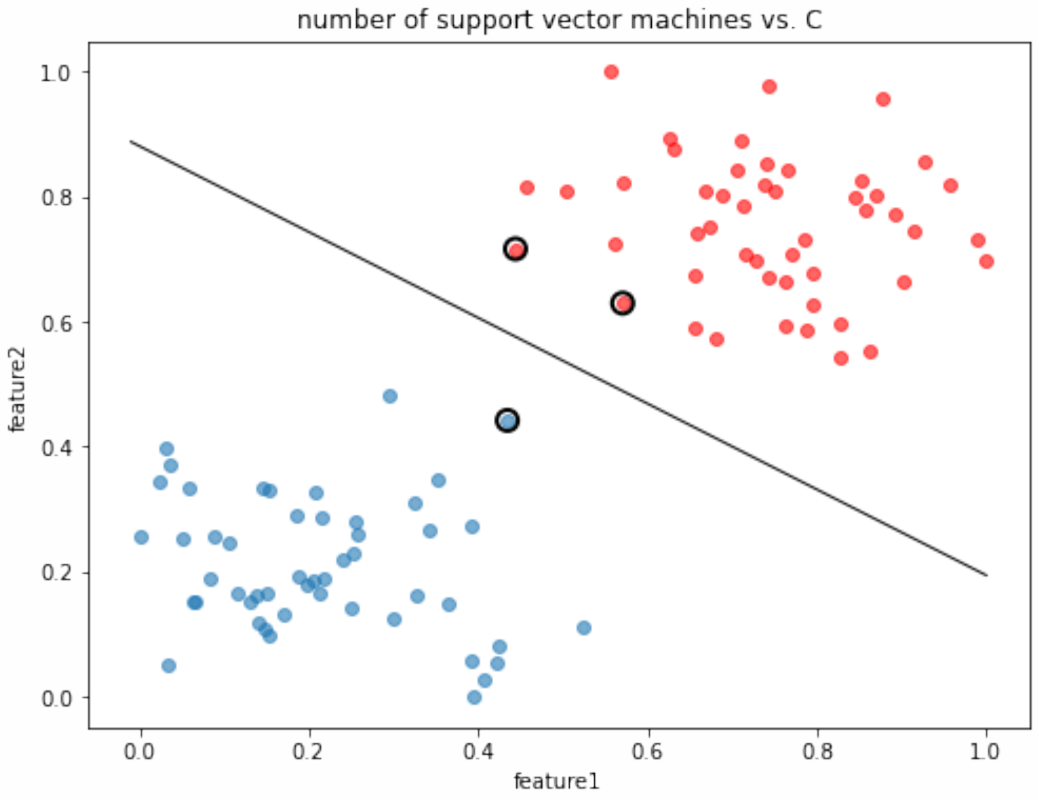

number of support vector machines vs. C
feature1
feature2
0.0
0.2
0.4
0.6
0.8
1.0

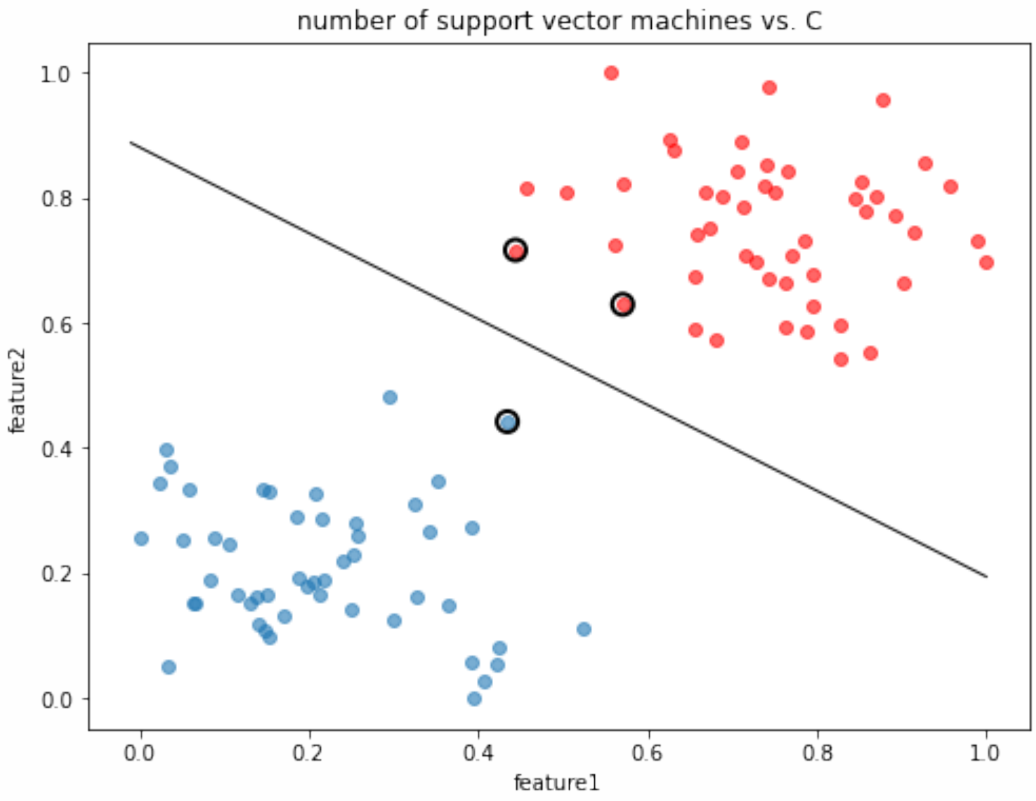

number of support vector machines vs. C
feature1
feature2
0.0
0.2
0.4
0.6
0.8
1.0

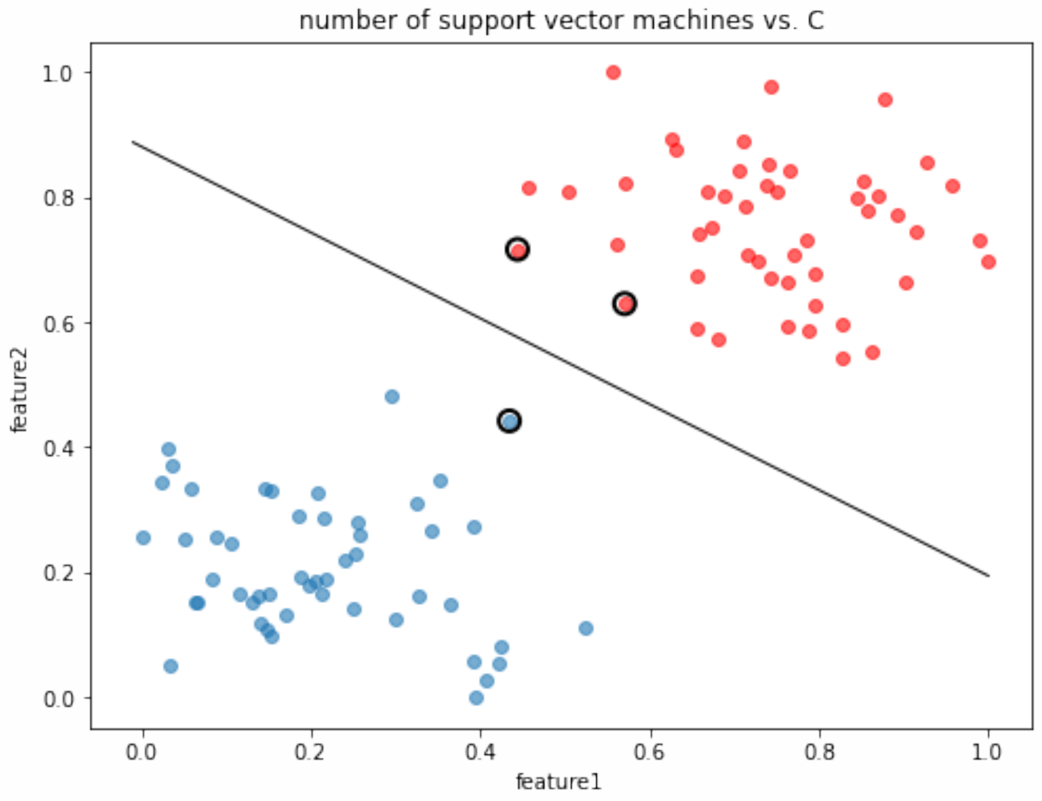

number of support vector machines vs. C
feature2
feature1
0.0
0.2
0.4
0.6
0.8
1.0

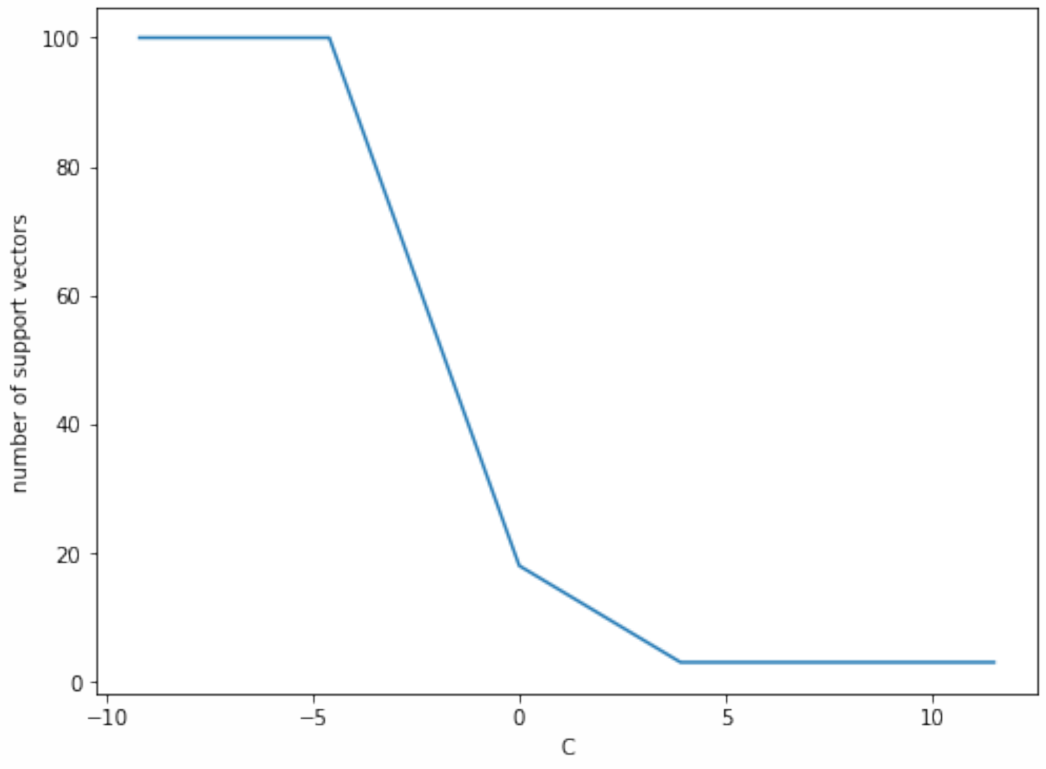

number of support vector machines vs. C
feature2
feature1
0.0
0.2
0.4
0.6
0.8
1.0


number of support vector machines vs. C
feature1
feature2


number of support vector machines vs. C
feature1
feature2


number of support vector machines vs. C
feature2
feature1
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





Yes, they are different from the previous question.

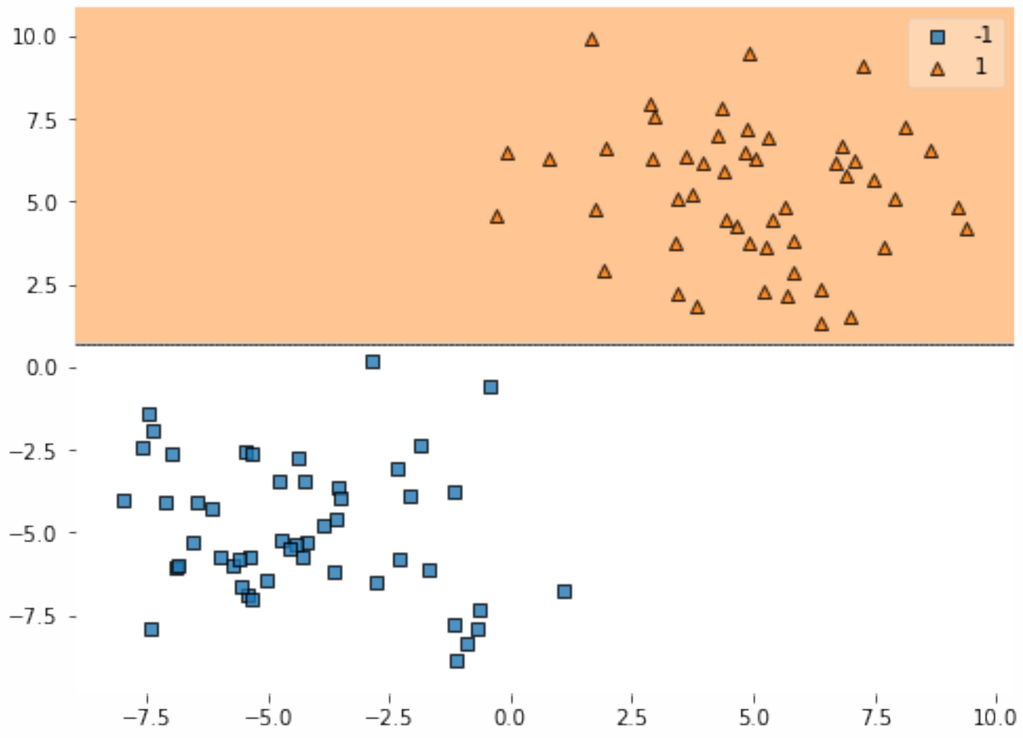
The geometric margin decreases while the effect of each feature on the predictions increases.

4.5 boosted decsion trees

```
from sklearn.ensemble import GradientBoostingClassifier
# from sklearn.inspection import DecisionBoundaryDisplay

gb_clf = GradientBoostingClassifier().fit(X_train, y_train)
gb_clf_rescaled = GradientBoostingClassifier().fit(X_train_scaled, y_train)
# disp = DecisionBoundaryDisplay.from_estimator(
#      gb_clf, X_train, response_method="predict",
#      xlabel=X_train.columns[:-1], ylabel=y_train.columns[-1],
#      alpha=0.5)
```

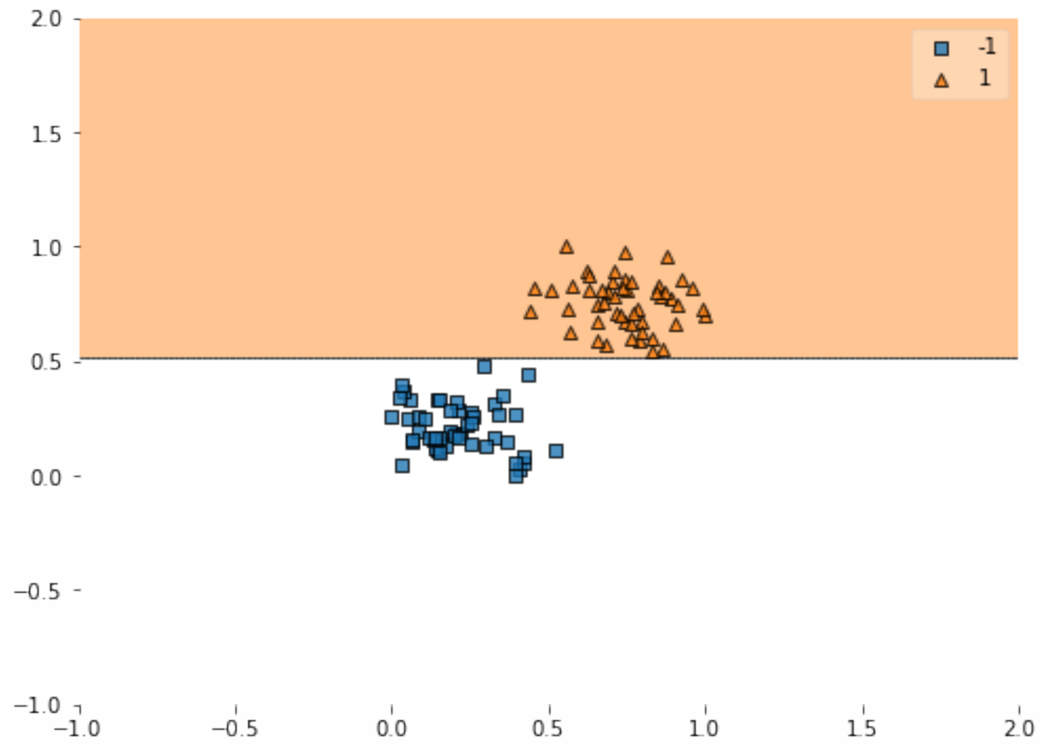
```
import matplotlib.gridspec as gridspec
from mlxtend.plotting import plot_decision_regions

fig = plt.subplots(figsize=(8,6))
plot_decision_regions(np.array(X_train), label.astype(int), gb_clf)
# plot_decision_regions(np.array(X_train_scaled), label.astype(int), gb_clf_rescaled)
```

```
<AxesSubplot:>
```



```
fig = plt.subplots(figsize=(8,6))
plot_decision_regions(np.array(X_train_scaled), label.astype(int), gb_clf_rescaled)
```

```
<AxesSubplot:>
```



The plots and decision boundaries do not change after rescaling the data when using tree model. That is consistent with my expectation.

5 Coding for Logistic Regression

```
import csv
import math
from sklearn.model_selection import train_test_split
```

5.2 Preprocessing

```python
# word dictionary
word_dict = {}
with open('dict.txt') as file:

    for line in file:
        (word, index) = line.strip().split(' ')
        index_int = int(index)
        word_dict[index_int] = str(word)

n_words = len(word_dict)
```

```python

def GetKey(dictionary, target):
    """
    get the key of certain value query
    """
    for key, value in dictionary.items():
        if value == target:

            return key

# review_dict = {}
label_dict = {}  # store labels of the reviews
review_dict = {}   # store the reviews like
                    # { 0: {23:1, 24:1, 25:1, ...}
                    #   1: {13:1, 14:1, 15:1, ...}
                    #   ...
                    #   n: {34:1, 36:1, ....} }
                # these inner dicts might have different lengths
                # because each review has different number of words

# open .tsv file
with open('moviereview.tsv') as file:

    tsv_file = csv.reader(file, delimiter="\t")
    file_list = list(tsv_file)
    for i in range(len(file_list)):    # get the index of each line
        label = file_list[i][0]
        review = file_list[i][1]
        label_int = int(label)  # convert string to int
        label_dict[i] = label_int    # store labels

        sentence_dict = {}  # for each line of review,
                            # create a dict to store the Booleans
                            # if the word appears in dictionary:
                                # set to 1
                            # else: ignore
        for word in review.split(' '):
            if word in word_dict.values():
                word_idx = GetKey(word_dict, word)
                sentence_dict[word_idx] = 1

        review_dict[i] = sentence_dict


len(review_dict[5])
len(review_dict[10])
```

```
493
```

```
# split data
# X_train5, y_train5, X_test5, y_test5 = train_test_split(review_dict, label_dict,
#                                                         test_size=0.20,
random_state=10)
def dict_slice(dict_, start, end):
    """
    a function to divide the dictionary
    into expected slices
    """
    keys = dict_.keys()
    dict_slice = {}
    for k in list(keys)[start:end]:
        dict_slice[k] = dict_[k]
    return dict_slice

# split train set and test set
X_train5 = dict_slice(review_dict, 0, round(0.8*len(review_dict)))
X_test5 = dict_slice(review_dict, round(0.8*len(review_dict)), len(review_dict))
y_train5 = dict_slice(label_dict, 0, round(0.8*len(label_dict)))
y_test5 = dict_slice(label_dict, round(0.8*len(label_dict)), len(label_dict))
```

```
# initialization
T = 30
eta = 0.1
theta = [0] * n_words
bias = 0
n_lines = len(X_train5)  # total number of reviews in the training set
n_words = len(word_dict)  # toal number of words in the dictionary
# 30 epochs
for t in range(T):
    grad_vector = [0] * n_lines  #initialize gradient vector
    for i in range(n_lines):   # interate each row of reviews
        sum1 = bias
        for index in X_train5[i]:   # for each line, iterate each word index
            sum1 += theta[index]   # sum out theta with corresponding index
                                    # this is a faster way to calculate dot product
        y_label = y_train5[i] # get the label for each review
        grad_vector[i] = y_label - 1 / (1 + math.exp(-sum1))   # calculate loss

# print(grad_vector)

    theta_sum = [0] * n_words
    bias_sum = 0
    for i in range(n_lines):
        for index in X_train5[i]:
            theta_sum[index] += grad_vector[i]
        bias_sum += grad_vector[i]

    for index in range(n_words):
        theta[index] += eta * theta_sum[index] / n_lines

    bias += eta * bias_sum / n_lines
```

```
# run test set

best_theta = theta
sup_script = 0
pred_labels = []
for key, reviews in X_test5.items():
    for idx, values in reviews.items():
        sup_script += best_theta[idx]

    P_pos = 1 / (1 + math.exp(sup_script))
    if P_pos > 0.5:
        pred_labels.append(1)

    else:
        pred_labels.append(0)


def Accuracy(true_label, predict_label):
    """""""
    a function to calculate prediction accuracy
    """""""

    correct = 0
    incorrect = 0
    for i,j in zip(true_label, predict_label):
        if i == j:
            correct += 1

        elif i != j:
            incorrect += 1

    accuracy_rate = correct / len(predict_label)

    return accuracy_rate


Accuracy(y_test5.values(), pred_labels)
```

```
0.9208333333333333
```

6 Boosted Decision Trees and Random Forest

6.1 Preprocessing

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
from sklearn.metrics import plot_roc_curve, roc_auc_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
titanic = pd.read_csv('Titanic.csv')
titanic.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	NaN	S
1	1	1	female	38.0	1	0	71.2833	C85	C
2	1	3	female	26.0	0	0	7.9250	NaN	S
3	1	1	female	35.0	1	0	53.1000	C123	S
4	0	3	male	35.0	0	0	8.0500	NaN	S

```
# drop NaN columns
df_ti = titanic.dropna(axis='columns')
df_ti.shape
df_ti1 = df_ti.copy()
```

```
(891, 6)
```

```
# encoding sex

def SexEncoder(df):

    if df['Sex'] == 'male':
        return 0
    elif df['Sex'] == 'female':
        return 1

df_encoded = pd.DataFrame(df_ti.apply(SexEncoder, axis=1), columns=['Sex_encoding'])
df_ti_encoded = pd.merge(left=df_ti, right=df_encoded, left_index=True, right_index=True)
df_ti_encoded1 = df_ti_encoded.drop(columns='Sex')
```

```
# split training set and test set

# split X and y
X = df_ti_encoded1.drop(columns='Survived')
y = df_ti_encoded1['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=10)
```

6.2 Fit models and estimate training time

```
RF_clf = RandomForestClassifier()
RF_clf.fit(X_train, y_train)
%time
```

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 5.96 µs
```

```
Ada_clf = AdaBoostClassifier()
Ada_clf.fit(X_train, y_train)
%time
```

```
CPU times: user 2 µs, sys: 1 µs, total: 3 µs
Wall time: 5.01 µs
```

Adaboost required less time to train.

Because Adaboost and boosted tree models constraints the dpeth of trees and number of leaves so that they train faster, while random forest has not restrictions on this parameters.

Adaboost default parameters:

1. n_estimator (default=50)(The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.).
2. Learning rate (default=1)(Weight applied to each classifier at each boosting iteration.)
3. ...

Random Forest default parameters:

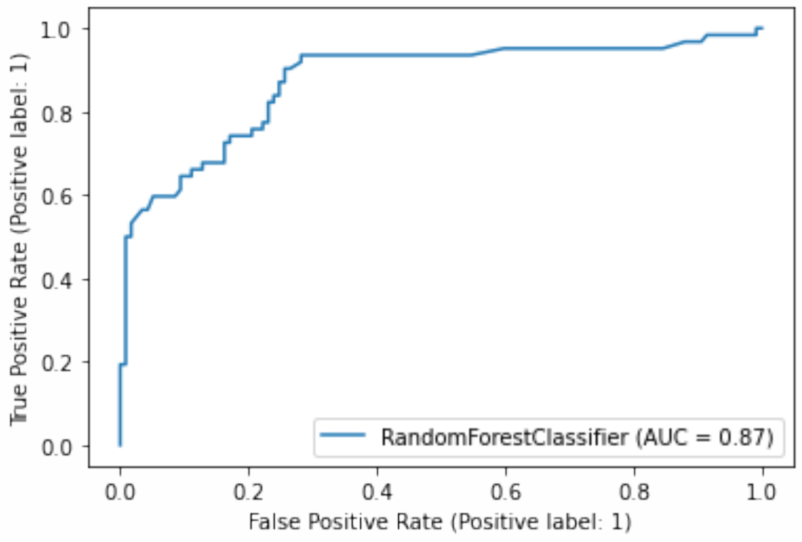
1. n_estimators (default=100)
2. max_depth=None
3. ...

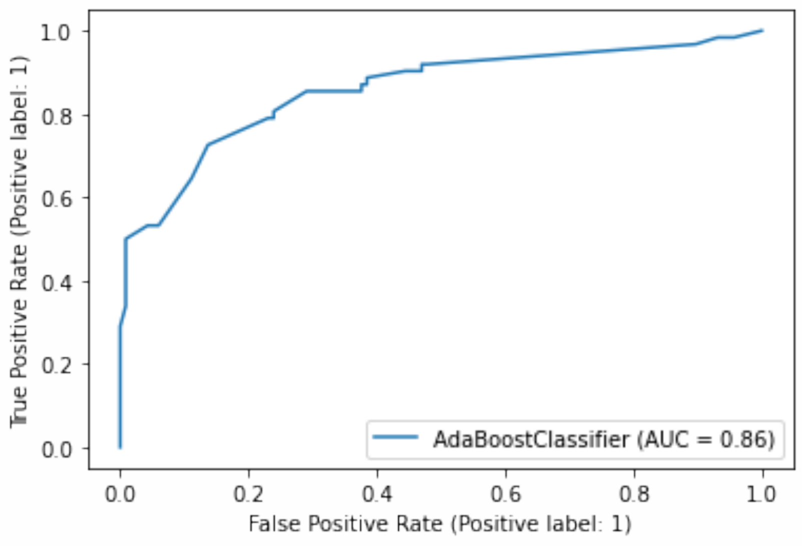
6.3 Tuning and plotting

```
# parameters
params_rf = {
    'criterion': ['gini','entropy'],
    'max_depth': [5, 10, 15, 20],
    'min_samples_split': [2, 4, 8, 12, 16],
    'min_samples_leaf': [1, 5, 10]
}

params_ab = {
    'n_estimators': [5, 10, 20, 30, 50, 75, 100],
    'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.5, 1]
}

scores={}
# hyperparameter tuning
rf_search = GridSearchCV(RF_clf, params_rf, cv=5, scoring='accuracy')
ab_search = GridSearchCV(Ada_clf, params_ab, cv=5, scoring='accuracy')

rf_search.fit(X_train, y_train)
ab_search.fit(X_train, y_train)


rf_best_param = rf_search.best_params_
ab_best_param = ab_search.best_params_
rf_best_score = rf_search.best_score_
ab_best_score = ab_search.best_score_

print(f'Best parameters for RandomForest: {rf_best_param}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: '+ \
    f'{rf_best_score:.3f}'
)
scores['RandomForest'] = rf_best_score
print('-----')

print(f'Best parameters for AdaBoost: {ab_best_param}')
print(
    f'Mean cross-validated accuracy score of the best_estimator: '+ \
    f'{ab_best_score:.3f}'
)
scores['AdaBoost'] = ab_best_score
```

```
Best parameters for RandomForest: {'criterion': 'gini', 'max_depth': 20,
'min_samples_leaf': 5, 'min_samples_split': 2}
Mean cross-validated accuracy score of the best_estimator: 0.813
-----
Best parameters for AdaBoost: {'learning_rate': 0.1, 'n_estimators': 75}
Mean cross-validated accuracy score of the best_estimator: 0.806
```

```
# use the best model to predict and plot ROC and compute AUC
# RandomForest
rf_best_model = RandomForestClassifier(criterion = 'entropy',
                                       max_depth = 15,
                                       min_samples_leaf = 5,
                                       min_samples_split = 16)
rf_best_model.fit(X_train, y_train)
plot_roc_curve(rf_best_model, X_test, y_test);

# Adaboost
ab_best_model = AdaBoostClassifier(learning_rate = 0.1,
                                  n_estimators = 75)
ab_best_model.fit(X_train, y_train)
plot_roc_curve(ab_best_model, X_test, y_test);
```





The ROC curves of two models looks very similar. And AUC score is also roughly the same.

Differences:

Random Forest use boostrap method to generate many trees with different splitting criterions, and use voting to finalize the tree model. While Adaboost combines several weak classifiers into one classifier, which would generalize better.

In this problem, the prediction result of two models are similar because the data is balanced and has resonable size. However, if sample size is small and noisy, random forest can leads to overfitting.

7 Generalized Additive Models

In [ ]:
```
pip install interpret
```

In [22]:
```
import pandas as pd
import numpy as np
from interpret.glassbox import ExplainableBoostingClassifier
from sklearn.model_selection import train_test_split
from interpret import show
from sklearn.metrics import plot_roc_curve, roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```
df_7 = pd.read_csv('penguins_trunc.csv')
df_7.head()
```

Out[6]:

	CulmenLength	CulmenDepth	FlipperLength	Species
0	39.1	18.7	181.0	0
1	39.5	17.4	186.0	0
2	40.3	18.0	195.0	0
3	36.7	19.3	193.0	0
4	39.3	20.6	190.0	0

In [8]:
```
X_7 = df_7.drop(columns=['Species'])
y_7 = df_7['Species']
```

In [9]:
```
X_train7, X_test7, y_train7, y_test7 = train_test_split(X_7, y_7, test_size=0.20, random_state=10)
```

In [10]:
```
ebm = ExplainableBoostingClassifier(random_state=15)
ebm.fit(X_train7, y_train7)
```

```
WARNING:interpret.utils.all:Passing a numpy array to schema autogen when it should be dataframe.
WARNING:interpret.utils.all:Passing a numpy array to schema autogen when it should be dataframe.
```

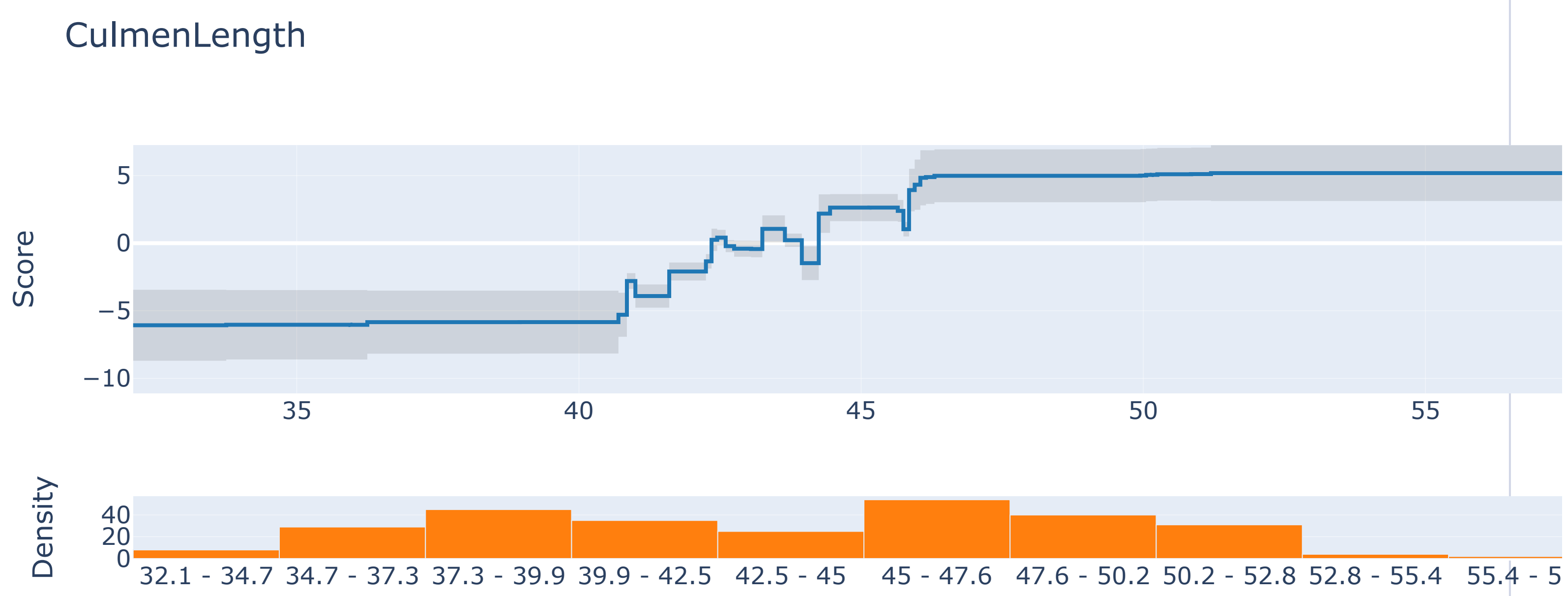
Out[10]:
```
ExplainableBoostingClassifier(feature_names=['CulmenLength', 'CulmenDepth',
                                             'FlipperLength',
                                             'CulmenLength x FlipperLength',
                                             'CulmenLength x CulmenDepth',
                                             'CulmenDepth x FlipperLength'],
                              feature_types=['continuous', 'continuous',
                                             'continuous', 'interaction',
                                             'interaction', 'interaction'],
                              random_state=15)
```

In [12]:
```
ebm_global = ebm.explain_global()
show(ebm_global)
```

Select Component to Graph

0 : Name (CulmenLength) | Type (continuous) | # Unique (145)

ExplainableBoostingClassifier_0

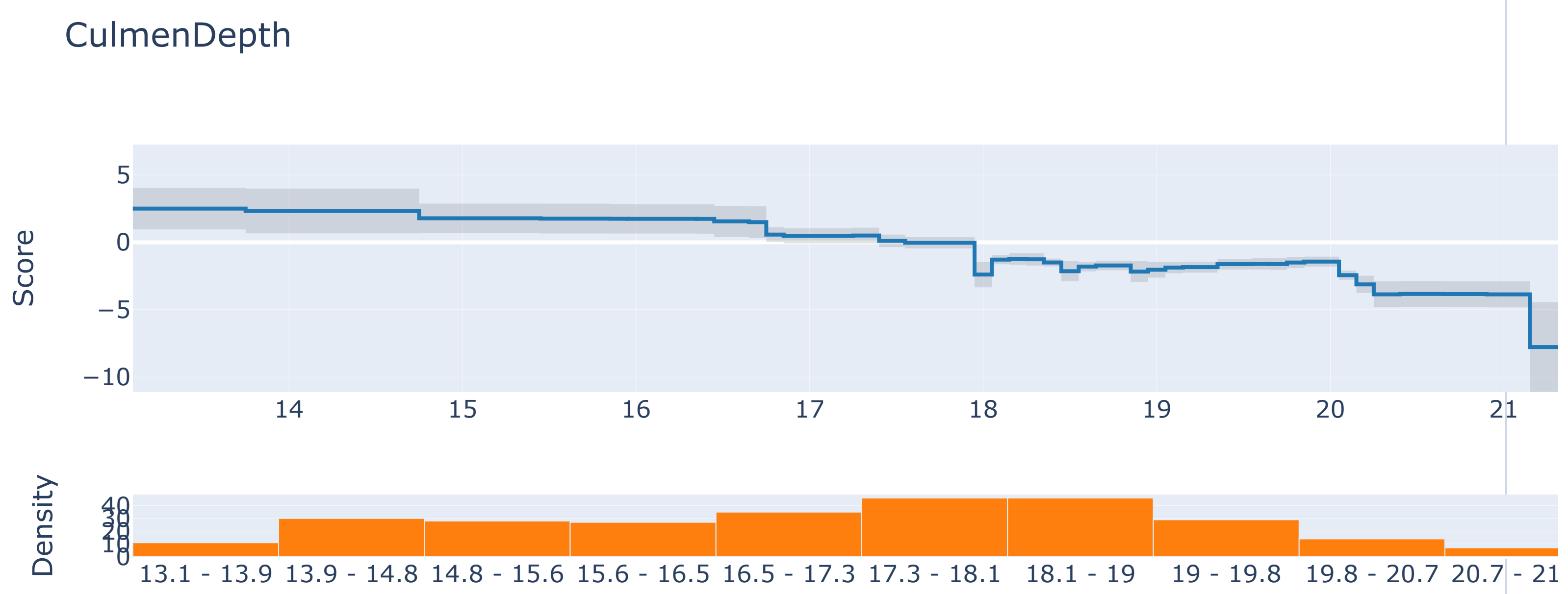


In [13]:
```
ebm_global = ebm.explain_global()
show(ebm_global)
```

Select Component to Graph

1 : Name (CulmenDepth) | Type (continuous) | # Unique (75)

ExplainableBoostingClassifier_1

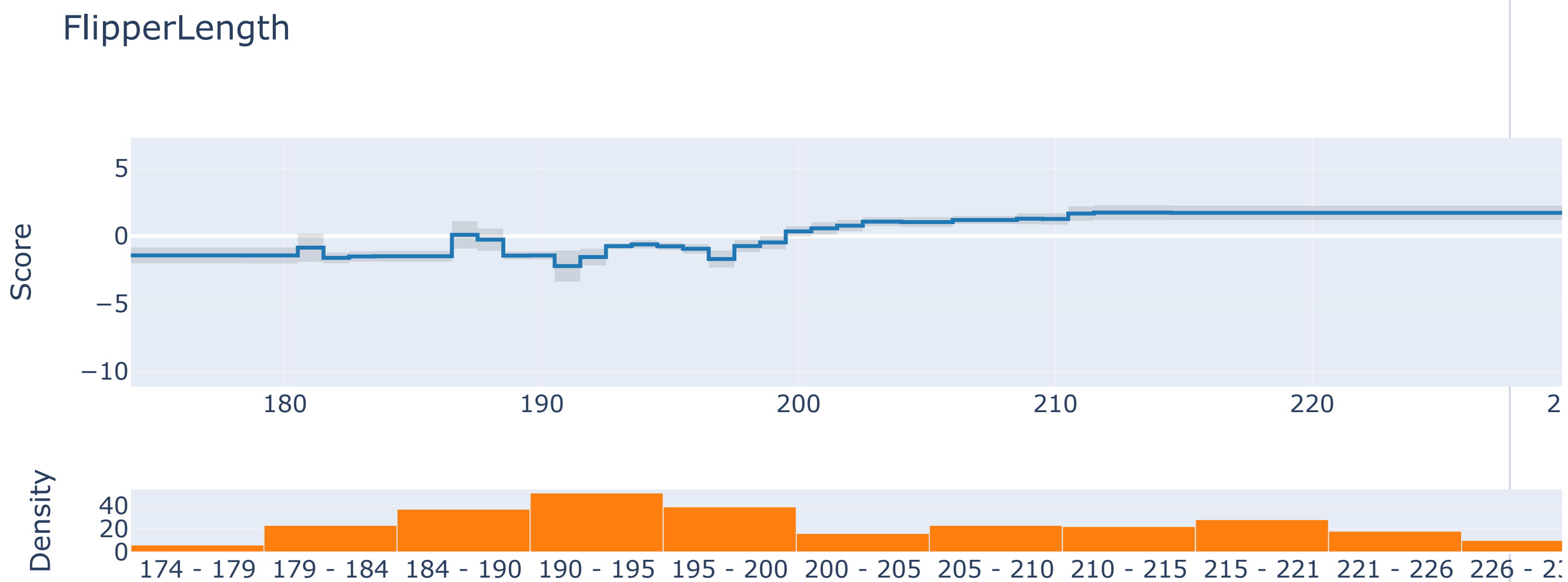


In [14]:
```
ebm_global = ebm.explain_global()
show(ebm_global)
```

Select Component to Graph

2 : Name (FlipperLength) | Type (continuous) | # Unique (53)

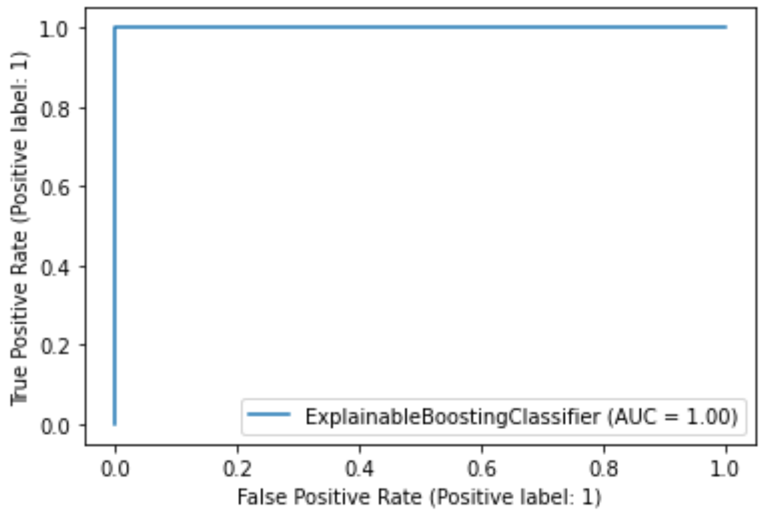
ExplainableBoostingClassifier_2

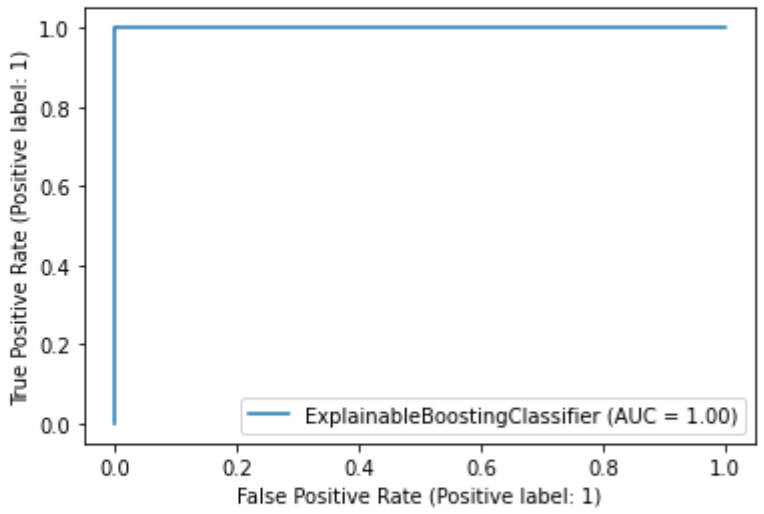


In [17]:
```
# training and test accuracy
training_accuracy = ebm.score(X_train7, y_train7)
test_accuracy = ebm.score(X_test7, y_test7)
print(training_accuracy)
print(test_accuracy)
```

```
1.0
1.0
```

In [23]:
```
plot_roc_curve(ebm, X_train7, y_train7);
plot_roc_curve(ebm, X_test7, y_test7);
```





In [ ]:
```
```