Homework 2, Machine Learning, Fall 2022

Alex Kumar

10/6/22

***IMPORTANT* Homework Submission Instructions**

1. All homeworks must be submitted in one PDF file to Gradescope.
2. Please make sure to select the corresponding HW pages on Gradescope for each question.
3. For all theory problems, please type the answer and proof using Latex or Markdown, and export the tex or markdown into a PDF file.
4. For all coding components, complete the solutions with a Jupyter notebook/Google Colab, and export the notebook (including both code and outputs) into a PDF file. Concatenate the theory solutions PDF file with the coding solutions PDF file into one PDF file which you will submit.
5. Failure to adhere to the above submission format may result in penalties.

All homework assignments must be your independent work product, no collaboration is allowed. You may check your assignment with others or the internet after you have done it, but you must actually do it by yourself. **Please copy the following statements at the top of your assignment file**:

Agreement 1) This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

Agreement 2) 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 (Chudi)

Convexity is very important for optimization. In this question, we will explore the convexity of some functions.

1.1

Let $h(x) = f(x) + g(x)$ where $f(x)$ and $g(x)$ are both convex functions. Show that $h(x)$ is also convex.

A function is convex if its second derivative is positive $\forall x$.
Since $f(x)$ and $g(x)$ are convex, then $f''(x) \geq 0$ and $g''(x) \geq 0, \forall x$.
Therefore, $\forall x$, $f''(x) + g''(x) \geq 0$.

$$h(x) = f(x) + g(x)$$
$$h''(x) = f''(x) + g''(x) \geq 0$$

Therefore $h(x)$ is convex.

1.2

In mathematics, a concave function is the negative of a convex function. Let $g_1(x), g_2(x), ..., g_k(x)$ be concave functions and $g_i(x) \geq 0$. Show that $\log(\prod_{i=1}^{k} g_i(x))$ is concave. (Note the base of log function is e.)
Hint 1: the sum of two concave function is concave.
Hint 2: $\log(a \times b) = \log(a) + \log(b)$.
Hint 3: log function is concave and monotonically increasing in $\mathbb{R}^+$.

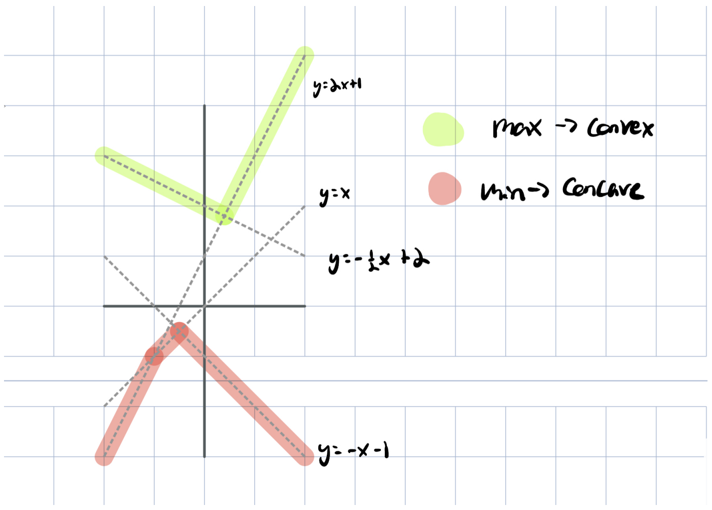
$$\log(\prod_{i=1}^{k} g_i(x)) = \log(g_1(x) * g_2(x) * ... * g_k(x))$$
$$= \log(g_1(x)) + \log(g_2(x)) + ... + \log(g_k(x)) = \sum_{i=1}^{k} \log(g_i(x))$$

Since the sum of two concave functions is concave and as the log function is concave,
Then $\sum_{i=1}^{k} \log(g_i(x))$ is concave.

Therefore, $\log(\prod_{i=1}^{k} g_i(x))$ is concave.

1.3

A line, represented by $y = ax + b$, is both convex and concave. Draw four lines with different values of a and b and highlight their maximum and minimum in two different colors. What does this reveal about the convexity of the maximum and minimum of these lines respectively?



The maximum of the lines is convex, and the minimum of the lines is concave.

1.4

Now we consider the general case. Let $f(x)$ and $g(x)$ be two convex functions. Show that $h(x) = \max(f(x), g(x))$ is also convex or give a counterexample.

By definition:

$$max(a,b) = \frac{a+b+|a-b|}{2}$$

$$|f(x)| \text{ is convex for any f(x)}$$

So:

$$max(f(x), g(x)) = \frac{f(x)+g(x)+|f(x)-g(x)|}{2}$$

As the absolute value of any function is convex, say $m(x) = f(x) - g(x)$
Then, $|m(x)| = |f(x) - g(x)|$, and both are convex. Therefore $|f(x) - g(x)|$ is convex.

As the sum of convex functions is convex, and it is given that $f(x)$ and $g(x)$ are convex,
$f(x) + g(x) + |f(x) - g(x)|$ is convex.

As multiplying a convex function by a positive scalar preserves convexity,
$\frac{f(x)+g(x)+|f(x)-g(x)|}{2}$ is still convex.

Therefore, $max(f(x), g(x))$ is convex.

1.5

Let $f(x)$ and $g(x)$ be two convex functions. Show that $h(x) = \min(f(x), g(x))$ is also convex or give a counterexample.

$h(x) = \min(f(x), g(x))$ is not convex!
If a function is convex, you must be able to draw a line between any two points on the graph where the line segment is both not intersecting the function and is above the function.

$$f(x) = e^x, g(x) = e^{-x}$$

$$x = -1 : f(-1) = 0.368, g(-1) = 0.368$$
$$x = 1 : f(1) = 0.368, g(1) = 0.368$$
$$x = 0 : f(0) = 1, g(0) = 1$$

Drawing a line segment between $x = -1$ and $x = 1$, the line segment is below the function as the y values at $x = -1$ and $x = 1$ is below the y value at $x = 0$.
Therefore, $h(x) = \min(f(x), g(x))$ can not be convex.

1.6

Since convex optimization is much more desirable then non-convex optimization, many commonly used risk functions in machine learning are convex. In the following questions, we will explore some of these empirical risk functions.
Suppose we have a dataset $\{\boldsymbol{X}, \boldsymbol{y}\} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ that we will perform classification on, where $\boldsymbol{x}_i \in \mathbb{R}^p$ is a p-dimensional feature vector, and $y_i \in \{-1, 1\}$ is the binary label. Please identify and show by proof whether the following empirical risk functions are convex with respect to weight parameter vector $\boldsymbol{\omega}$. In the model below, $\boldsymbol{\omega} = [\omega_1, \omega_2, ..., \omega_p]^T \in \mathbb{R}^p$, $b \in \mathbb{R}$ are the weight and bias. We would like to show convexity with respect to $\boldsymbol{\omega}$ and b.

(a)

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

Say $f(x)$ is convex and $g(x)$ is an affine function $Ax + b$, which is also convex.
Definition of convexity:

$$tf(x_1) + (1-t)f(x_2) \geq f(tx_1 + (1-t)x_2)$$

As $g(x)$ is convex:

$$f(g(x)) = f(g(tx_1 + (1-t)x_2) = f(A(tx_1 + (1-t)x_2) + b)$$
$$= f(A(tx_1) + A((1-t)x_2) + b) = f(t(Ax_1 + b) + (1-t)(Ax_2 + b))$$

As $f(x)$ is convex:

$$f(t(Ax_1 + b) + (1-t)(Ax_2 + b)) \leq t(f(Ax_1 + b)) + (1-t)(f(Ax_2 + b))$$
$$f(t(g(x_1)) + (1-t)(g(x_2))) \leq t(f(g(x_1))) + (1-t)(f(g(x_2)))$$

Therefore, $f(g(x))$ is convex when both $f(x)$ and $g(x)$ is convex.

As e^{-x} is convex and $y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)$ is affine, $e^{-y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)}$ is convex.
Therefore, we can think of $e^{-y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)}$ as e^{z_i}

Say $h(x) = \log(1 + e^{-x})$

Then:

$$h(z_i) = \log(1 + e^{-z_i})$$
$$h'(z_i) = \frac{-e^{-z_i}}{1 + e^{-z_i}}$$

$$h''(z_i) = \frac{e^{-z_i}}{(1+e^{-z_i})^2}$$

A function is convex if its second derivative is positive across its range.
As $\forall z_i, \quad e^{-z_i} \geq 0$ and $(1+e^{-z_i})^2 \geq 0, \quad h''(z_i) \geq 0$.

Therefore, $h(z_i)$ is convex.

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

As the sum of convex functions is convex, and as $h(z_i)$ is convex, $\sum_{i=1}^{n} h(z_i)$ is convex.

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

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

(b)

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

For the first part of the equation:

$$\frac{1}{2}||\boldsymbol{\omega}||_2^2 = \frac{1}{2}\sqrt{\sum_i \omega_i^2}^{\,2} = \frac{1}{2}\sum_i \omega_i^2$$

$$\forall \omega_i, \quad \frac{d^2}{d\omega^2}\omega_i^2 = 2$$

Every ω_i in the sum has a positive second derivative, meaning every term is convex.
The sum of convex functions is also convex.
Therefore, $||\boldsymbol{\omega}||_2^2$ is convex.

Multiplying a convex function by a positive scalar preserves convexity.
Therefore, $\frac{1}{2}||\boldsymbol{\omega}||_2^2$ is convex.

For the second part of the equation:

$$C\sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b))$$

0 can be thought of as $0 = 0x + 0$, which is a linear function. Linear functions are convex.

As shown in the previous part, $y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)$ is an affine function. It follows that $1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)$ is also affine. Affine functions are convex.

The maximum of two convex functions is convex, as proved 1.4. Therefore, $\max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b))$ is convex.

The sum of convex functions is convex, therefore $\sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b))$ is convex.

Multiplying a convex function by a positive scalar preserves convexity.
Therefore, $C \sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b))$ is convex.

As the sum of convex functions is convex,
$\frac{1}{2}||\boldsymbol{\omega}||_2^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\omega}^T \boldsymbol{x}_i + b)), \quad C \geq 0$ is convex.

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

1.7

Consider the lasso regression problem. Given a dataset $\{\boldsymbol{X}, \boldsymbol{y}\} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ where $y_i \in \mathbb{R}$, please identify and show by proof whether the following empirical risk function is convex with respect to the weight parameter vector $\boldsymbol{\omega}$.

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

For the first term in the equation:

$$\|\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)\|_2^2$$

$$\text{Imagine: } z = \boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)$$

The square of a L2 norm is convex:
For any L2 norm, $||x||$, the L2 norm convex by its properties.

$$||kx|| = |k| \cdot ||x||, \quad ||x + y|| \leq ||x|| + ||y||$$

Using the Triangle Inequality:

$$t \in [0, 1]| \quad ||tx_1 + (1-t)x_2|| \leq ||tx_1|| + ||(1-t)x_2||$$

$$||tx_1 + (1-t)x_2|| \leq |t| \cdot ||x_1|| + |(1-t)| \cdot ||x_2||$$

$$||tx_1 + (1-t)x_2|| \leq t||x_1|| + (1-t)||x_2||$$

This is the definition of convexity.

As $\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)$ is an affine function, it is convex. As shown in 1.6, the squared L2 norm is convex when its terms are convex.

Therefore $||z||_2^2 = ||\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)\|_2^2$ is convex.

For the second term in the equation:

$$C\|\boldsymbol{\omega}\|_1 = C \sum_i |\omega_i|$$

By definition, the absolute value function is convex. Therefore, $|\omega_i|$ is convex.
As the summation of convex functions is convex, $\sum_i |\omega_i|$ is convex.
As multiplying a convex function by a positive scalar preserves convexity, $C \sum_i |\omega_i|$ is convex.
Therefore, $C\|\boldsymbol{\omega}\|_1$ is convex.

As the sum of convex functions is convex, $\|\boldsymbol{y} - (X\boldsymbol{\omega} + b\mathbf{1}_n)\|_2^2 + C\|\boldsymbol{\omega}\|_1, \quad C \geq 0$ is convex.

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

2 Gradient Descent and Newton's Method (Chudi)

Optimization algorithms are important in machine learning. In this problem, we will explore ideas behind gradient descent and Newton's method.
Suppose we want to *minimize* a convex function given by

$$f(\boldsymbol{x}) = 2x_1^2 + 3x_2^2 - 4x_1 - 6x_2 + 2x_1x_2 + 13.$$

2.1

(a) Find the expression for $\nabla f(\boldsymbol{x})$, the first derivative of $f(\boldsymbol{x})$.

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} 2x_1^2 + 3x_2^2 - 4x_1 - 6x_2 + 2x_1x_2 + 13 \\ \frac{\partial}{\partial x_2} 2x_1^2 + 3x_2^2 - 4x_1 - 6x_2 + 2x_1x_2 + 13 \end{bmatrix} = \begin{bmatrix} 4x_1 + 2x_2 - 4 \\ 2x_1 + 6x_2 - 6 \end{bmatrix}$$

(b) Find the expression for $Hf(\boldsymbol{x})$, the Hessian of $f(\boldsymbol{x})$.

$$Hf(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix}$$

(c) Compute *analytically* the value of $\boldsymbol{x}^*$, at which the optimum is achieved for $f(\boldsymbol{x})$. Hint: use the first two parts.

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 4x_1 + 2x_2 - 4 \\ 2x_1 + 6x_2 - 6 \end{bmatrix} = 0$$

$$x_2 = -2x_1 + 2, x_1 = -3x_2 + 3$$

$$x_2 = -2(-3x_2 + 3) + 2 = 6x_2 - 4$$

$$x_2 = \frac{4}{5}, x_1 = \frac{3}{5}$$

$$x^* = \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix}$$

2.2

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. Since $f(\boldsymbol{x})$ is strictly convex and differentiable, gradient descent can be used to find the unique optimal solution.

Recall that gradient descent takes repeated steps in the opposite direction of the gradient of the function at the current point until convergence. The algorithm is shown below. Given an initial point $\boldsymbol{x}_0$ and a step size α,

- $t = 0$

- while $\boldsymbol{x}_t$ is not a minimum
  - $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha \nabla f(\boldsymbol{x}_t)$
  - $t = t + 1$
- end

For each iteration, $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha \nabla f(\boldsymbol{x}_t)$. Suppose $\alpha = 0.1$. Show that $\boldsymbol{x}_{t+1}$ converges to $\boldsymbol{x}^*$ obtained in 2.1 as $t \to \infty$.

Hint 1: Let A be a square matrix. $I + A + A^2 + \ldots A^k = (I - A)^{-1}(I - A^{k+1})$.

Hint 2: Let A be a square matrix. A^k converges to zero matrix if all eigenvalues of matrix A have absolute value smaller than 1.

By definition:

$$\nabla f(\boldsymbol{x}_t) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

$$\begin{bmatrix} 4x_1 + 2x_2 - 4 \\ 2x_1 + 6x_2 - 6 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -4 \\ -6 \end{bmatrix}$$

Plugging in:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha \nabla f(\boldsymbol{x}_t)$$

$$= \boldsymbol{x}_t - \alpha(\begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix} x_t + \begin{bmatrix} -4 \\ -6 \end{bmatrix}) = \boldsymbol{x}_t - \begin{bmatrix} 4\alpha & 2\alpha \\ 2\alpha & 6\alpha \end{bmatrix} x_t + \begin{bmatrix} 4\alpha \\ 6\alpha \end{bmatrix}$$

$$= (I - \begin{bmatrix} 4\alpha & 2\alpha \\ 2\alpha & 6\alpha \end{bmatrix})x_t + \begin{bmatrix} 4\alpha \\ 6\alpha \end{bmatrix}$$

Say:

$$A = I - \begin{bmatrix} 4\alpha & 2\alpha \\ 2\alpha & 6\alpha \end{bmatrix}, \quad b = \begin{bmatrix} 4\alpha \\ 6\alpha \end{bmatrix}$$

Then:

$$x_{t+1} = Ax_t + b$$

$$= A(Ax_{t-1} + b) + b = A(A(Ax_{t-2} + b) + b) + b$$

$$= A^{t+1}x_0 + (A^t + A^{t-1} + ... + A + I)b$$

$$= A^{t+1}x_0 + (I - A)^{-1}(I - A^{t+1})b$$

If A has eigenvalues with an absolute value of less than 1, then A^K converges to a 0 matrix.

$$A = I - \begin{bmatrix} 4\alpha & 2\alpha \\ 2\alpha & 6\alpha \end{bmatrix}$$

$$det(A - \lambda I) = det(\begin{bmatrix} 1 - 4\alpha - \lambda & -2\alpha \\ -2\alpha & 1 - 6\alpha - \lambda \end{bmatrix})$$

$$= (1 - 4\alpha - \lambda)(1 - 6\alpha - \lambda) - (4\alpha^2) = 0$$

$$\lambda = 1 - 5\alpha \pm \sqrt{5\alpha}$$

$$\lambda^2 - 2\lambda + 10\alpha\lambda + 20\alpha^2 - 10\alpha + 1 = 0$$

$$\lambda^2 + \lambda(10\alpha - 2) = -20\alpha^2 + 10\alpha - 1$$

Completing the square for λ:

$$(\frac{1}{2}(10\alpha - 2))^2 + \lambda^2 + \lambda(10\alpha - 2) = \frac{1}{2}(10\alpha - 2))^2 - 20\alpha^2 + 10\alpha - 1$$

$$(\frac{1}{2}(10\alpha - 2) + \lambda)^2 = 5\alpha^2$$

$$\frac{1}{2}(10\alpha - 2) + \lambda = \pm\sqrt{5}\alpha$$

$$\lambda = 1 - 5\alpha \pm \sqrt{5}\alpha$$

As $\alpha = 0.1$, $\lambda = \frac{1}{2} \pm \frac{\sqrt{5}}{10}$, both of which have an absolute value < 1.

Therefore, as $t \to \infty$, $A^k \to 0$

$$\lim_{t\to\infty} A^{t+1}x_0 + (I - A)^{-1}(I - A^{t+1})b = (I - A)^{-1}b$$

$$I - A = \begin{bmatrix} 4\alpha & 2\alpha \\ 2\alpha & 6\alpha \end{bmatrix}$$

$$(I - A)^{-1} = \begin{bmatrix} \frac{3}{10\alpha} & \frac{-1}{10\alpha} \\ \frac{-1}{10\alpha} & \frac{1}{5\alpha} \end{bmatrix}$$

$$(I - A)^{-1}b = \begin{bmatrix} \frac{3}{10\alpha} & \frac{-1}{10\alpha} \\ \frac{-1}{10\alpha} & \frac{1}{5\alpha} \end{bmatrix} \begin{bmatrix} 4\alpha \\ 6\alpha \end{bmatrix} = \begin{bmatrix} \frac{3}{5} \\ \frac{4}{5} \end{bmatrix} = x^*$$

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

2.3

Will $\boldsymbol{x}_{t+1}$ converge to $\boldsymbol{x}^*$ when α grows larger? Hint: the previous question's calculations should provide you the answer.

When A has eigenvalues of absolute value ≥ 1, then x_{t+1} is not guaranteed to converge to x^* as $t \to \infty$.

$$|\lambda| = |1 - 5\alpha \pm \sqrt{5}\alpha| \geq 1$$

At $\alpha = 0.267$, some of A's eigenvalues are equal to 1.

$$x_{t+1} = A^{t+1}x_0 + (I - A)^{-1}(I - A^{t+1})b$$

As A^k does not converge to 0 as $t \to \infty$, then x_{t+1} does not converge as $t \to \infty$.
Essentially, the step size is too large, and x_{t+1} will oscillate around x^* infinitely.

2.4

Gradient descent only uses the first derivative to make updates. If a function is twice differentiable, is the second derivative able to help find optimal solutions? Newton's method takes the second derivative into consideration and the key update step is modified as $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - (Hf(\boldsymbol{x_t}))^{-1}\nabla f(\boldsymbol{x}_t)$. The algorithm is shown below.

- $t = 0$
- while $\boldsymbol{x}_t$ is not a minimum
  - $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - (Hf(\boldsymbol{x_t}))^{-1}\nabla f(\boldsymbol{x}_t)$

  - $t = t + 1$

- end

(a) Using the initial point $\boldsymbol{x}_0 = [1, 2]^T$, give the value of $\boldsymbol{x}_1$ that Newton's method would find in one iteration. Does this method converge in one iteration? If not, give the value of $\boldsymbol{x}_2$ that Newton's method would find and report whether $\boldsymbol{x}_2$ is equal to $\boldsymbol{x}^*$.

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 4x_1 + 2x_2 - 4 \\ 2x_1 + 6x_2 - 6 \end{bmatrix}$$

$$Hf(\boldsymbol{x}) = \begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix}, Hf(\boldsymbol{x})^{-1} = \frac{1}{10}\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}$$

Using the update rule $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - (Hf(\boldsymbol{x_t}))^{-1}\nabla f(\boldsymbol{x}_t)$:

$$\nabla f(\boldsymbol{x_0}) = [4, 8]^T$$

$$x_1 = [1, 2]^T - \frac{1}{10}\begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}[4, 8]^T = \frac{1}{5}[3, 4]^T$$

It converges to x^* in one iteration.

(b) Using the initial point $\boldsymbol{x}_0 = [1, 2]^T$ and a fixed step size $\alpha = 0.1$, give the value of $\boldsymbol{x}_1$ that gradient descent would find in one iteration. Does this method converge in one iteration? If not, give the value of $\boldsymbol{x}_2$ that gradient descent would find and report whether $\boldsymbol{x}_2$ is equal to $\boldsymbol{x}^*$.

Using the update rule $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha\nabla f(\boldsymbol{x}_t)$:

$$x_1 = [1, 2]^T - \frac{1}{10}[4, 8]^T = \frac{1}{5}[3, 6]^T$$

Doesn't converge in one iteration.

$$\nabla f(x_1) = \frac{1}{5}[4, 12]^T$$

$$x_2 = \frac{1}{5}[3, 6]^T - \frac{1}{10}\frac{1}{5}[4, 12]^T = \frac{1}{25}[13, 24]^T$$

(c) According to the results above, which method converges faster? Please briefly explain why it happens.

Newton's method converges faster because gradient descent takes smaller steps, $\alpha = 0.1$. Newton's method does not impose a smaller step size on each iteration.

3 Bias and Variance of Random Forest

In this question, we will analyze the bias and variance of the random forest algorithm. **Bias** is the amount that a model's prediction differs from the target value in the training data. An underfitted model can lead to high bias. **Variance** indicates how much the estimate of the target function will alter if different training data were used. An overfitted model can lead to high variance.

3.1

Let $\{X, y\} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be the training dataset, where $\boldsymbol{x}_i$ is a feature vector and $y_i \in \{-1, 1\}$ is the binary label. Suppose all overlapping samples are consistently labeled, i.e. $\forall i \neq j$, $y_i = y_j$ if $\boldsymbol{x}_i = \boldsymbol{x}_j$. Show that 100% training accuracy tree can be achieved if there is no depth limit on the trees in the forest. Hint: the proof is short. One way to proof it is by induction.

Base case: $n = 1$, so $\hat{y} = y_1$

Assume $n = k$, where tree D correctly classifies all k samples

When $n = k + 1$, the $k + 1^{\text{th}}$ sample will be correctly classified. Either:

Case 1: The $k + 1^{\text{th}}$ sample is the same as a previous sample in $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^k$, so it is correctly classified
Case 2: The $k + 1^{\text{th}}$ sample is incorrectly classified, but can be correctly classified by branching from the leaf node that incorrectly classified it, adding a new leaf node that correctly classifies (x_{k+1}, y_{k+1})

Therefore, a tree D can correctly classify all samples if there is no depth limit.

3.2

Let $D_1, D_2, ..., D_T$ be random variables that are identically distributed with positive pairwise correlation ρ and $Var(D_i) = \sigma^2, \forall i \in \{1, ..., T\}$. Let $\bar{D}$ be the average of $D_1, ..., D_T$, i.e. $\bar{D} = \frac{1}{T}(\sum_{i=1}^T D_i)$. Show that

$$Var(\bar{D}) = \frac{1 - \rho}{T}\sigma^2 + \rho\sigma^2. \tag{1}$$

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

As:

$$Var(\sum_{i=1}^T D_i) = Cov(\sum_{i=1}^T D_i, \sum_{j=1}^T D_j) = \frac{1}{T^2}(\sum_{i=1}^T Var(D_i) + \sum_{i,j|i\neq j}^{T,T} Cov(D_i, D_j))$$

As all D_i are IID, then $= T\sigma^2 + T(T-1)\rho\sigma^2$, then:

$$\sum_{i=1}^T Var(D_i) + \sum_{i,j|i\neq j}^{T,T} Cov(D_i, D_j)$$

Plugging in:

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

$$= \frac{1 - \rho}{T}\sigma^2 + \rho\sigma^2$$

3.3

In the random forest algorithm, we can view the decision tree grown in each iteration as approximately D_i in Problem 3.2. Given this approximation, briefly explain how we expect the variance of the average of the decision trees to change as T increases.

As T increases, we expect $Var(\bar{D})$ to decrease, as the scalar multiplying σ^2 and being added to $\rho\sigma^2$ is decreasing with T.

3.4

As T becomes large, the first term in Eq(1) goes away. But the second term does not. What parameter(s) in random forest control ρ in that second term?

ρ can be thought of as the correlation between the different decision trees D. Parameters that affect the correlation between trees D_i and D_j, $i \neq j$ can be max_depth, max_features and max_samples.

max_features is the number of features that can be selected for the tree to split on. A lower value for this parameter increases variance, as when the value of this parameter is high, it is more likely that trees D_i and D_j will split on the same parameter.

max_samples is the number of samples that are used to calculate how good a certain split is. Similar to the previous parameter, a lower value for this parameter increases variance, as when the value of this parameter is high, it is more likely that trees D_i and D_j will be calculating their splits with the same data. This increases the chance that the same or similar splits are chosen.

max_depth is the maximum depth of the tree. A higher value for this parameter increases variance because as the depth gets larger, the set of possible combinations of split grows exponentially.

4 Coding for SVM (Henry)

4.1

Generate a training set of size 100 (50 samples for each label) with 2D features (X) drawn at random as follows:

- $X_{neg} \sim \mathcal{N}([-5, -5], 5 \cdot I_2)$ and correspond to negative labels (-1)
- $X_{pos} \sim \mathcal{N}([5, 5], 5 \cdot I_2)$ and correspond to positive labels (+1)

Accordingly, $X = [\substack{X_{pos} \\ X_{neg}}]$ is a 100×2 array, Y is a 100×1 array of values $\in \{-1, 1\}$. Draw a scatter plot of the full training dataset with the points colored according to their labels.

4.2

Train a linear support vector machine on the data with $C = 1$ and draw the decision boundary line that separates positives and negatives. Mark the support vectors separately (e.g., circle around the point).

4.3

Draw decision boundaries for 9 different C values across a wide range of values between 0 and infinity. Plot the number of support vectors vs. C (plot on a log scale if that's useful). How does the number of support vectors change as C increases and why does it change like that?

As C increases, the number of support vectors decreases. It decreases because the cost of having slack increases, meaning the model would rather misclassify points. This means that the margin decreases, meaning fewer points are within the margin, meaning fewer points are close enough to be support vectors.

4.4

Now, try to rescale the data to the [0,1] range and repeat the steps of the previous question (4.3) and over the same range of C values. Are the decision boundaries different from those in the previous question? What does this imply about (a) the geometric margin and (b) the relative effect of each feature on the predictions of the trained model?

The decision boundaries are slightly different. This implies that the geometric margin decreases with the scaled data, and is therefore is sensitive to the absolute range of the data. This also implies that the relative effect of each feature on the predictions is reliant on its magnitude, so the normalized data prevents a feature with naturally larger values from being too important to the geometric margin calculation.

4.5

Try using boosted decision trees (using any boosted decision tree algorithm you can find) with and without changing the scaling of the dataset and plot both decision boundaries on one plot. Do the plots differ? Is that what you expected? (Make sure you use the same random seed for both runs of your algorithm.)

The plots are very similar; the decision boundaries appear to be the same, except scaled and shifted to fit with the data. This is to be expected as Boosted Decision Trees choose a feature to split over and the boundary at which to make the split. The boundary chosen is not affected by how the features of the data is scaled, only scored based on how the points are sorted between the two new nodes in respect to their labels.

5 Coding for Logistic Regression (Henry)

Download the Movie Review data file from Sakai named moviereview.tsv. The data file is in tab-separated-value (.tsv) format. The data is from the Movie Review Polarity data set (for more details, see http://www.cs.cornell.edu/people/pabo/movie-review-data/). Currently, the original data was distributed as a collection of separate files (one movie review per file). The data we are using converted this to have one line per example, with the label 0 or 1 in the first column (the review is negative or positive) followed by all of the words in the movie review (with none of the line breaks) in the second column. We provide a dictionary file to limit the vocabulary to be considered in this assignment. The dictionary.txt uses the format: [word, index], which represents the k^{th} word (so it has the index k) in the dictionary.

5.1

In this section, your goal is to derive the update formula for the logistic regression when optimizing with gradient descent. Gradient descent is as follows:

```
To optimize J($\theta$, x, y)
choose learning rate $\eta$
t as current time, T as max time
$\theta_0 \rightarrow$ initial value
**while** t < T: **do**
    Update: $\theta_j \rightarrow \theta_j - \eta \cdot \frac{\partial J(\theta, \mathbf{x}, y)}{\partial \theta_j}$
**end while**
```

Assume you are given a data set with n training examples and p features. We first want to find the negative conditional log-likelihood of the training data in terms of the design matrix $\mathbf{X}$, the labels $\mathbf{y}$, and the parameter vector θ. This will be your objective function $J(\theta)$ for gradient descent. Here we assume that each feature vector $x_{i,\cdot}$ contains a bias feature, that is, $x_{i,1} = 1 \;\; \forall i \in 1, ..., n$. Thus, $x_{i,\cdot}$ is a $(p+1)$-dimensional vector where $x_{i,1}$ =1. As such, the bias parameter is folded into our parameter vector θ.

(a) Derive the negative log-likelihood of $p(y|\mathbf{X}, \theta)$. (Hint: look at the course notes.)

$$p(y_i = 1|x_i, \theta) = \frac{e^{\theta^T x_i}}{1 + e^{\theta^T x_i}}, \quad p(y_i = 0|x_i, \theta) = 1 - p(y_i = 1|x_i, \theta) = \frac{1}{1 + e^{\theta^T x_i}}$$

$$p(y|\mathbf{X},\theta) = \prod_i^n p(y_i|x_i,\theta) = \prod_i^n p(y_i = 1|x_i,\theta)^{y_i} p(y_i = 0|x_i,\theta)^{1-y_i}$$

$$= \prod_i^n (\frac{e^{\theta^T x_i}}{1+e^{\theta^T x_i}})^{y_i} (\frac{1}{1+e^{\theta^T x_i}})^{1-y_i} = \prod_i^n (\frac{e^{\theta^T x_i y_i}}{(1+e^{\theta^T x_i})^{y_i}})((1+e^{\theta^T x_i})^{y_i-1}) = \prod_i^n \frac{e^{\theta^T x_i y_i}}{1+e^{\theta^T x_i}}$$

$$-\log(p(y|\mathbf{X},\theta)) = -\log(\prod_i^n \frac{e^{\theta^T x_i y_i}}{1+e^{\theta^T x_i}})$$

$$= -\sum_i^n \log(\frac{e^{\theta^T x_i y_i}}{1+e^{\theta^T x_i}}) = -\sum_i^n \log(e^{\theta^T x_i y_i}) - \log(1+e^{\theta^T x_i})$$

$$= -\sum_i^n \theta^T x_i y_i - \log(1+e^{\theta^T x_i}) = J(\theta)$$

(b) Then, derive the partial derivative of the negative log-likelihood $J(\theta)$ with respect to θ_j, $j \in 1, ..., p+1$.

Note: $\theta_j^T x_i = x_{ij}$

$$\frac{\partial J(\theta)}{\partial \theta_j} = -\sum_{i=1}^n y_i x_{ij} - \frac{x_{ij} e^{\theta^T x_i}}{1+e^{\theta^T x_i}} = \sum_{i=1}^n -y_i x_{ij} + \frac{x_{ij} e^{\theta^T x_i}}{1+e^{\theta^T x_i}}$$

(c) The update rule is $\theta_j \to \theta_j - \eta \cdot \frac{\partial J(\theta)}{\partial \theta_j}$. Based on the update rule, write the gradient descent update for parameter element θ_j.

We will implement this momentarily.

$$\theta_j \to \theta_j - \eta \cdot \sum_{i=1}^n -y_i x_{ij} + \frac{x_{ij} e^{\theta^T x_i}}{1+e^{\theta^T x_i}}$$

5.2

In this section, you are going to do some preprocessing. You are going to create something similar to a bag-of-words feature representation, where the feature vector $x_{i,v}$ for movie review i and vocabulary word v in the dictionary is set to 1 if movie review i contains at least one occurrence of vocabulary v. An example of the output from your pre-processing for one movie review might be as follows: 20:1 22:1 30:1 35:1 45:1 ...., which expresses that vocabulary words 20, 22, 30, 35, 45, etc. are in the movie review. The feature vector ignores words not in the vocabulary of dict.txt. You can use csv and math packages in python.

5.3

In this section, implement a sentiment polarity analyzer using binary logistic regression. Specifically, you will learn the parameters of a binary logistic regression model that predicts a sentiment polarity (i.e., the label) for the corresponding feature vector of each movie review using gradient descent, as follows:

- Initialize all model parameters to 0.
- Use gradient descent to optimize the parameters for a binary logistic regression model. The number of times gradient descent loops through all of the training data (number of epochs). Set your learning rate as a constant η=0.1.

- Perform gradient descent updates on the training data for 30 epochs.

Do not hard-code any aspects of the data sets into your code. You should only use the packages that are provided. You may use sklearn for creating the train/test split. You can use csv and math packages in python.

Let us provide a note about efficient computation of dot-products. In linear models like logistic regression, the computation is often dominated by the dot-product $\theta^T \mathbf{x}$ of the parameters $\theta \in \mathbb{R}^p$ with the feature vector $\mathbf{x} \in \mathbb{R}^p$. When a standard representation of $\mathbf{x}$ is used, the dot-product requires $O(p)$ computation, because it requires a sum over all entries in the vector: $\theta^T \mathbf{x} = \sum_{j=1}^{p} \theta_j x_{\cdot,j}$ However, if our feature vector is represented sparsely, we can observe that the only elements of the feature vector that will contribute a non-zero value to the sum are those where $x_{\cdot,j} \neq 0$, since this would allow $\theta_j x_{\cdot,j}$ to be nonzero. This requires only computation proportional to the number of non-zero entries in $\mathbf{x}$, which is generally very small for model compared to the size of the vocabulary.

6 Boosted Decision Trees and Random Forest (Henry)

In this assignment, you are going to fit and tune random forest and boosted decision trees to predict whether a given passenger survived the sinking of the Titanic based on the variables provided in the data set. You may use sklearn and gridsearch.

6.1

First, download the Titanic.csv file from Sakai, and drop the columns that have "na" values from the data set. Convert the gender variable into a binary variable with 0 representing male, and 1 representing female. Then split 80% of the data into train and 20% of the data into test set.

6.2

Fit a random forest and boosted decision tree model on the training set with **default** parameters, and estimate the time it required to fit each of the models. Which model required less time to train? Why do you think that is? (Hint: look at the default values of the parameters.)

With default parameters, Adaboost took less time to train compared to the RandomForest. This is because Adaboost has 50 estimators by default, while RandomForest has 100 estimators by default.

6.3

Choose a range of parameter values for each of the algorithms (tell us what parameters you played with), and tune on the training set over 5 folds. Then, draw the ROC and provide the AUC for each of the tuned models on the whole training set (in one figure) and test set (in another figure). Comment on the similarities/differences between the two models (if any).

Gridsearch over parameters = n_estimators: [10, 50, 100, 300], max_depth: [1, 2, 3], criterion: [gini, entropy]
It appears that the Boosted Decision Tree model overfits on the training data when compared to the Random Forest model, which generalizes well. The Boosted Trees model has AUC = 0.98 on the train set, but AUC = 0.77 on the test set. The Random Forest model has AUC = 0.86 on the train set and AUC = 0.89 on the test set.

7 Generalized Additive Models

In this question, you are going to fit a generalized additive model (GAM) to predict the species of penguins on a given dataset. You can download the dataset penguins_trunc.csv from Sakai. The dataset is preprocessed and you can use it directly.

Split 80% of the data into a training set and 20% of the data into a test set. Please use one of the following methods to fit a GAM model: (1) logistic regression with ℓ_1 penalty, (2) Explainable Boosting Machine, (3) FastSparse, (4) pyGAM. Report training and test accuracy and plot the shape function for each feature on a separate plot.

Hint 1: For logistic regression with ℓ_1 penalty, you can use sklearn packages. You need to first transform each continuous feature into a set of binary features by splitting on the mid-point of every two consecutive values realized in the dataset, i.e., CulmenLength ≤ 32.6, CulmenLength ≤ 33.3. You can then fit regularized logistic regression to these indicator variables and construct the component function for variable j by weighting the indicator variables that pertain to feature j by their learned coefficients and adding them up. You can use sklearn LogisticRegression. Please remember to set penalty to "ℓ_1" and algorithm to "liblinear".

Hint 2: Explainable Boosting Machine is a tree-based, cyclic gradient boosting generalized additive modeling package. It can be pip installed for Python 3.6+ Linux, Mac, Windows. More details about the algorithm and usage are available in `https://github.com/interpretml/interpret` and `https://interpret.ml/docs/ebm.html`.

Hint 3: FastSparse implements fast classification techniques for sparse generalized linear and additive models in R. To install this package, please follow the instruction here `https://github.com/jiachangliu/fastSparse/tree/main/installation`. Example code about how to run FastSparse is available `https://github.com/jiachangliu/fastSparse/tree/main/application_and_usage` and example code to visualize shape functions is `https://github.com/jiachangliu/fastSparse/tree/main/visualization`.

Hint 4: You may also use pygam package to fit a GAM model which is available here `https://github.com/dswah/pyGAM`.

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.

CS 671: Homework 2

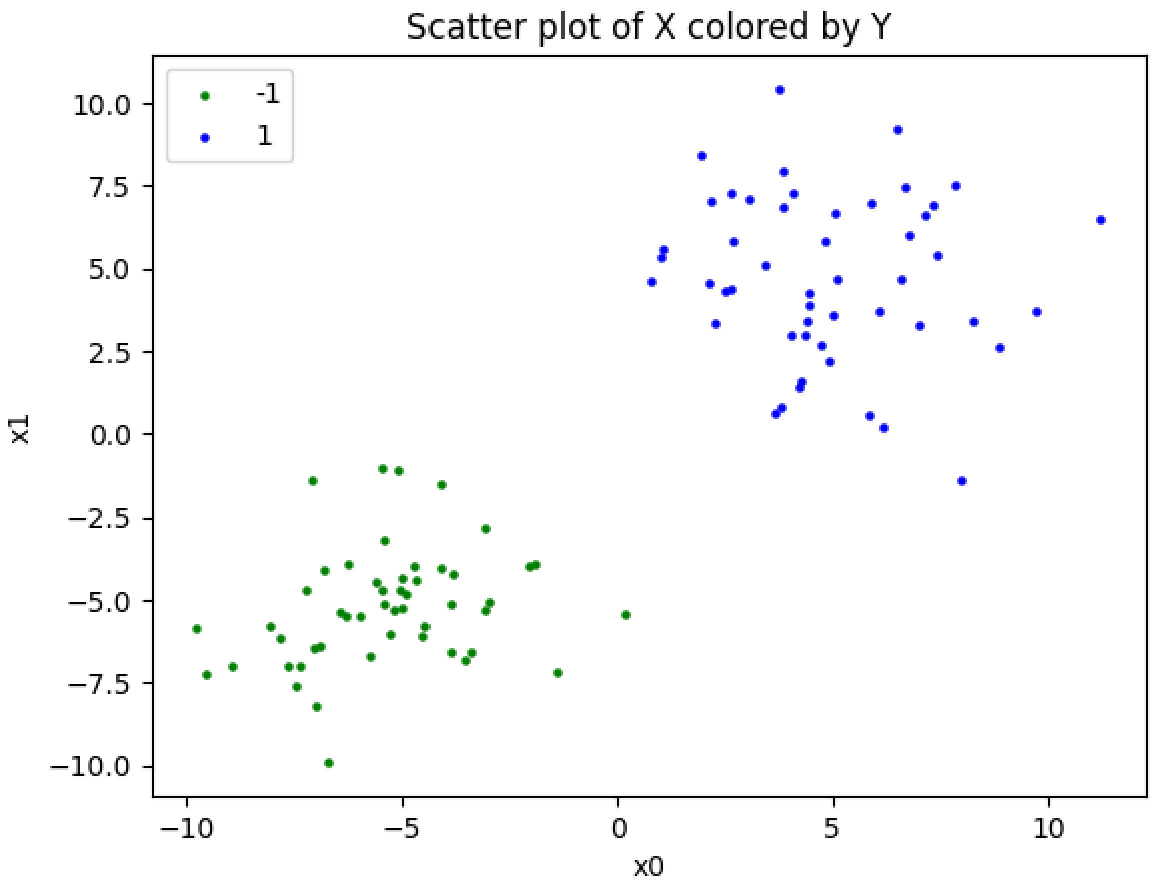
Alex Kumar

Question 4

In [ ]:
```
### Imports
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn import preprocessing
from sklearn.ensemble import AdaBoostClassifier
from sklearn import tree
from sklearn.inspection import DecisionBoundaryDisplay
```

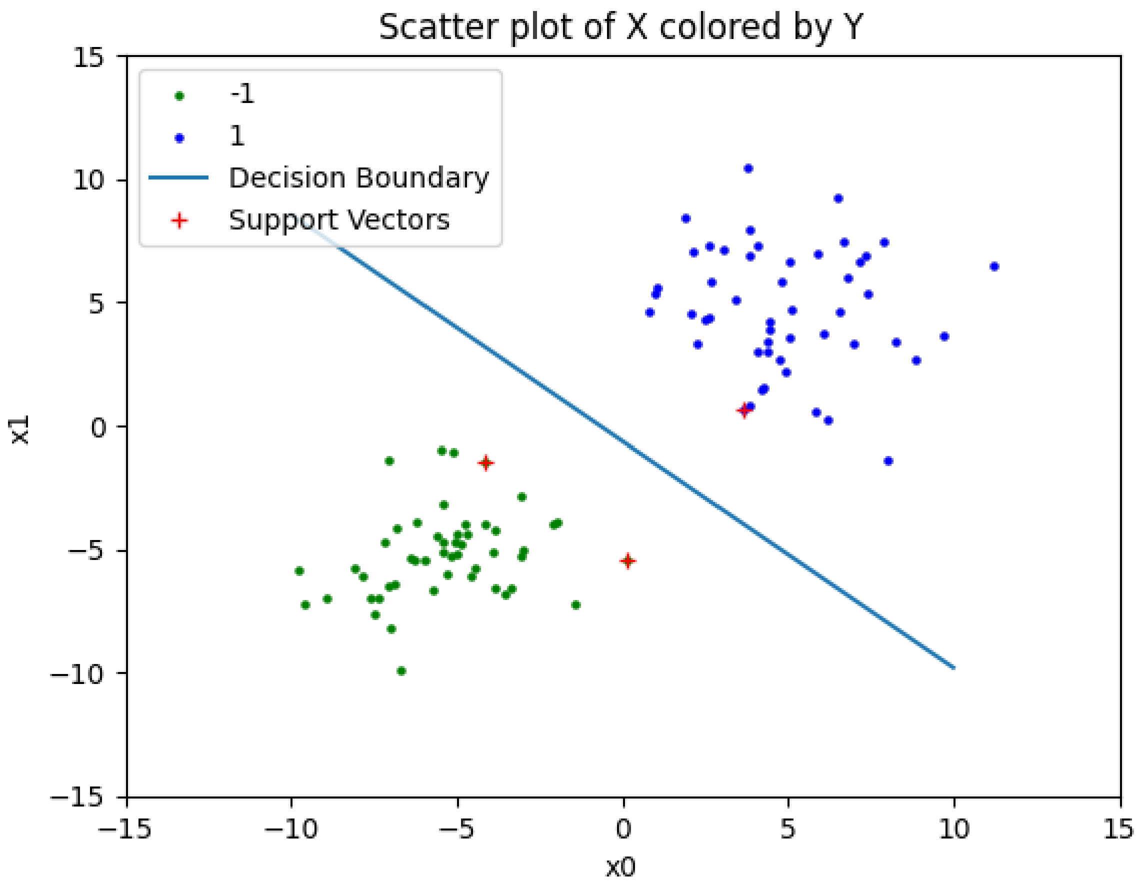
In [ ]:
```
### 4.1: Create training set X
NUM_SAMPLES = 50
I2 = np.array([[1, 0], [0, 1]])

X_neg = list(np.random.multivariate_normal([-5, -5], 5*I2, NUM_SAMPLES))
X_pos = list(np.random.multivariate_normal([5, 5], 5*I2, NUM_SAMPLES))
y_neg, y_pos = [[[-1]]*50][0], [[[1]]*50][0]

X_data = np.concatenate((X_neg, X_pos))
Y_data = np.concatenate((y_neg, y_pos))
```

In [ ]:
```
### 4.1 Scatter plot of data
def scat_plot(X_data, Y_data):
    # General function to plot X colored by Y
    plt.figure(1)
    plt.scatter(X_data[:, 0][:50], X_data[:, 1][:50], c='g', s=5)
    plt.scatter(X_data[:, 0][50:], X_data[:, 1][50:], c='b', s=5)
    plt.xlabel("x0")
    plt.ylabel("x1")
    plt.title("Scatter plot of X colored by Y")
    plt.legend([-1, 1], loc = "upper left")
    return

scat_plot(X_data, Y_data)
```

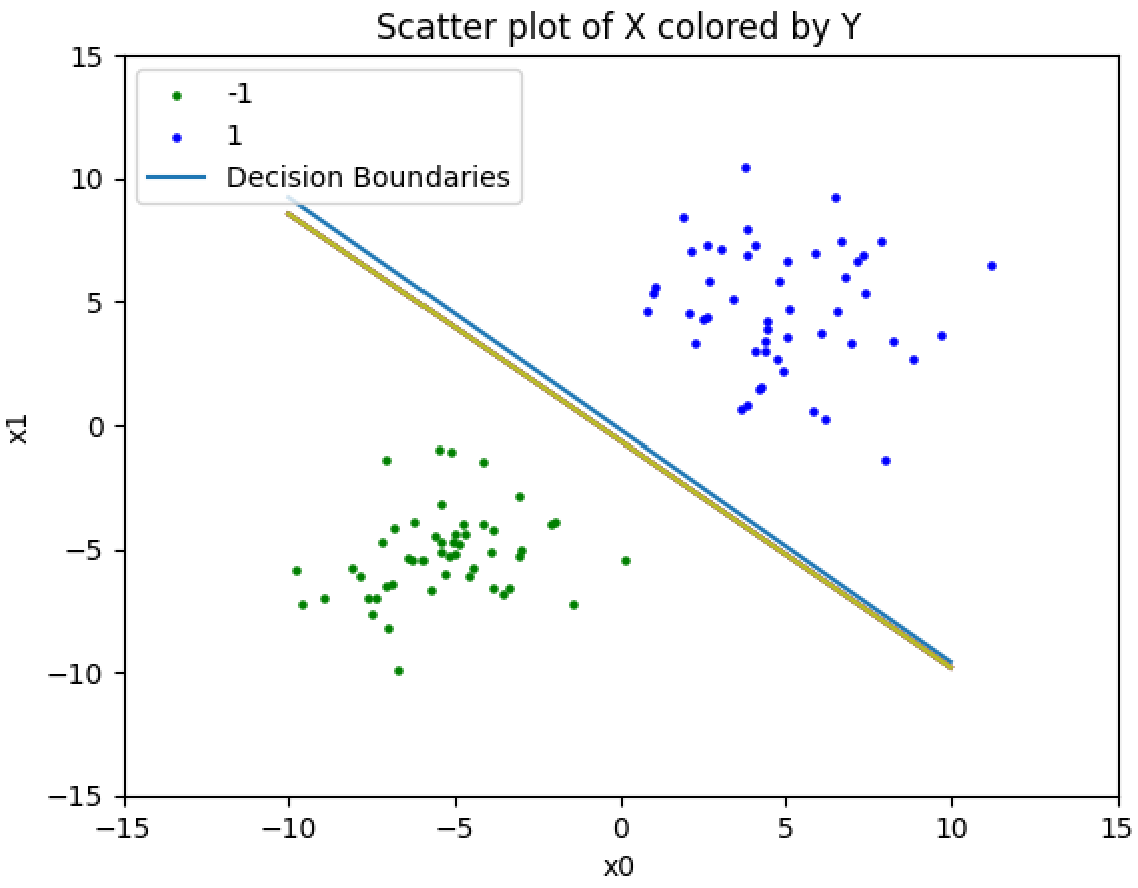


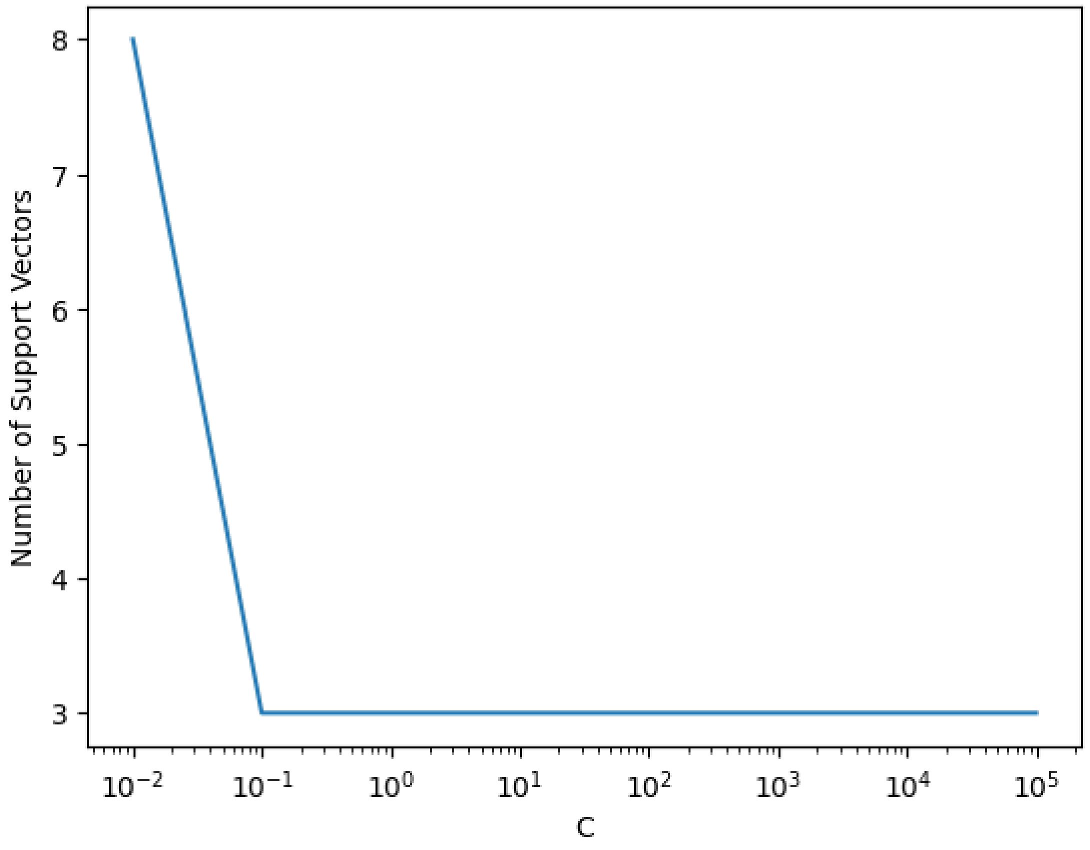
In [ ]:
```
### 4.2 SVM C=1

def train_svm(X_data, Y_data, c):
    # General function to train linear SVM & return boundary and support vecto
    lin_svm = svm.SVC(kernel="linear", C=c)
    lin_svm.fit(X_data, np.reshape(Y_data, (len(Y_data),)))
    # Get decision boundary and support vectors
    w, b = lin_svm.coef_[0], lin_svm.intercept_[0] # lambda_* and lambda_0
    sup_vec = lin_svm.support_vectors_
    return w, b, sup_vec

w, b, sup_vec = train_svm(X_data, Y_data, 1)
x_pts = np.linspace(-10, 10)
y_pts = -(w[0] / w[1]) * x_pts - (b / w[1])        # make decision boundary po

# Plot with decision boundary and support vecotrs
scat_plot(X_data, Y_data)
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.plot(x_pts, y_pts)
for v in sup_vec:
    plt.plot(v[0], v[1], 'r+')
plt.legend(["-1", "1", "Decision Boundary", "Support Vectors"], loc = "upper le
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x2978bb760>
```



```
In [ ]: ### 4.3 C vs num support vectors
        Cs = [0.01, 0.1, 0.5, 1, 10, 100, 1000, 10000, 100000]

        scat_plot(X_data, Y_data)
        plt.xlim(-15, 15)
        plt.ylim(-15, 15)

        c_vs_supvec = []

        for c in Cs:
            w, b, sup_vec = train_svm(X_data, Y_data, c)
            c_vs_supvec.append([c, len(sup_vec)])
            x_pts = np.linspace(-10, 10)
            y_pts = -(w[0] / w[1]) * x_pts - (b / w[1])
            plt.plot(x_pts, y_pts)
        plt.legend(["-1", "1", "Decision Boundaries"], loc = "upper left")

        plt.figure(2)
        c_vs_supvec = np.reshape(c_vs_supvec, (len(c_vs_supvec), 2))
        plt.plot(c_vs_supvec[:, 0], c_vs_supvec[:, 1])
        plt.xscale("log")
        plt.xlabel("C")
        plt.ylabel("Number of Support Vectors")

Out[ ]: Text(0, 0.5, 'Number of Support Vectors')
```

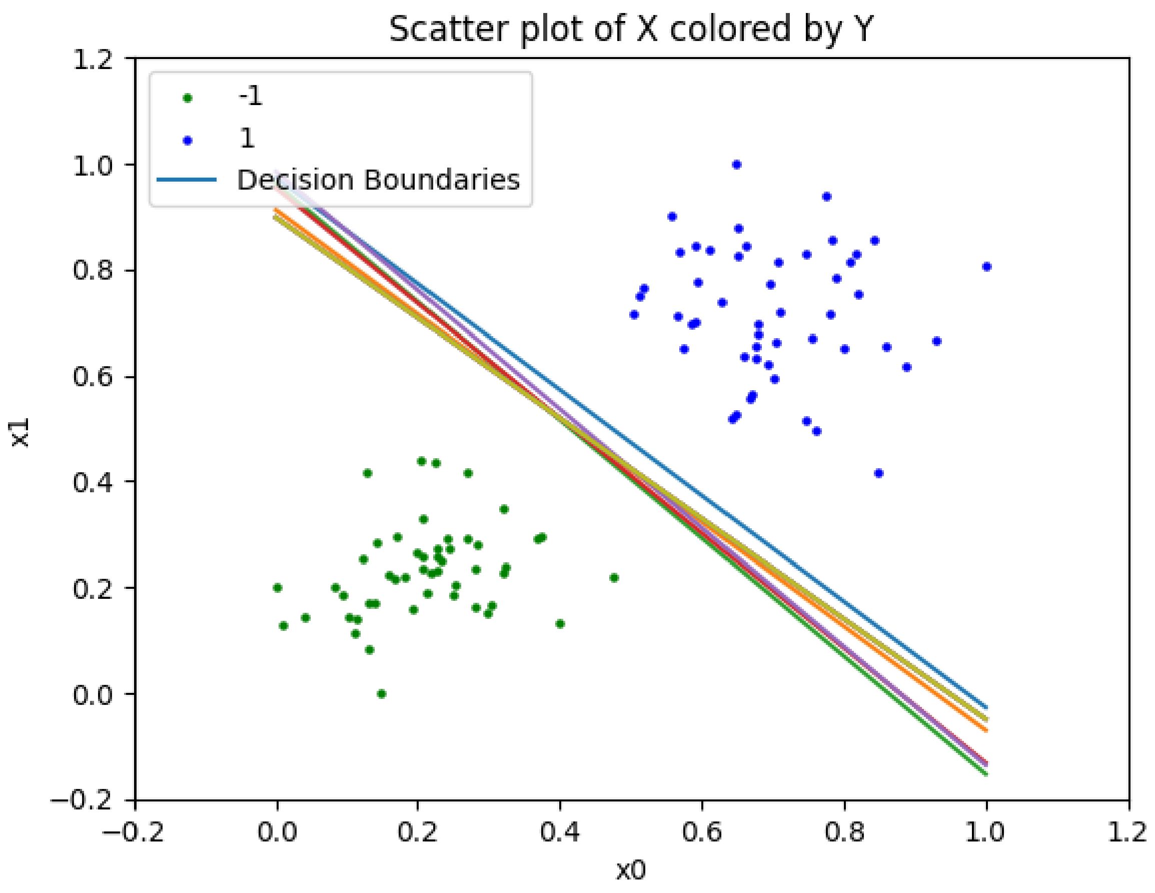


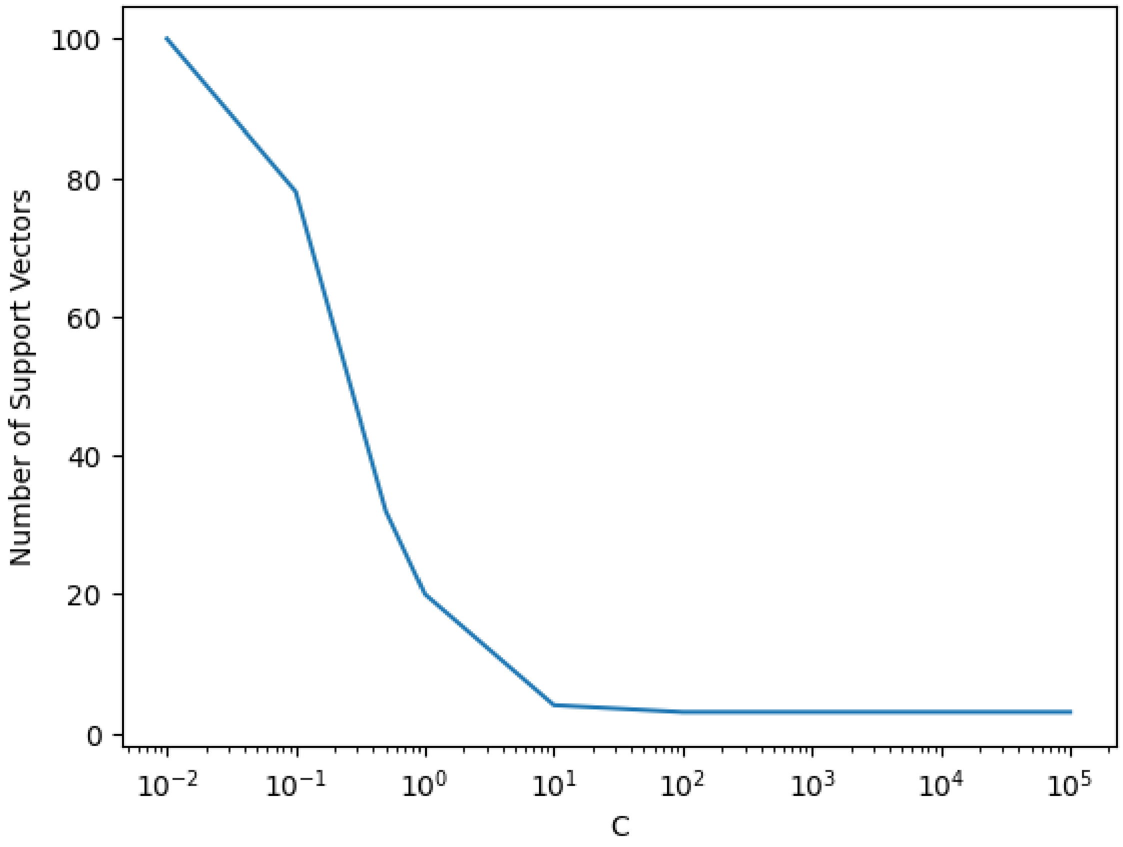


In [ ]:
```
### 4.4 With normalized data
X_norm = preprocessing.MinMaxScaler().fit_transform(X_data)
```

```
Cs = [0.01, 0.1, 0.5, 1, 10, 100, 1000, 10000, 100000]

scat_plot(X_norm, Y_data)
plt.xlim(-0.2, 1.2)
plt.ylim(-0.2, 1.2)

c_vs_supvec = []

for c in Cs:
    w, b, sup_vec = train_svm(X_norm, Y_data, c)
    c_vs_supvec.append([c, len(sup_vec)])
    x_pts = np.linspace(0, 1)
    y_pts = -(w[0] / w[1]) * x_pts - (b / w[1])
    plt.plot(x_pts, y_pts)
plt.legend(["-1", "1", "Decision Boundaries"], loc = "upper left")

plt.figure(2)
c_vs_supvec = np.reshape(c_vs_supvec, (len(c_vs_supvec), 2))
plt.plot(c_vs_supvec[:, 0], c_vs_supvec[:, 1])
plt.xscale("log")
plt.xlabel("C")
plt.ylabel("Number of Support Vectors")
```

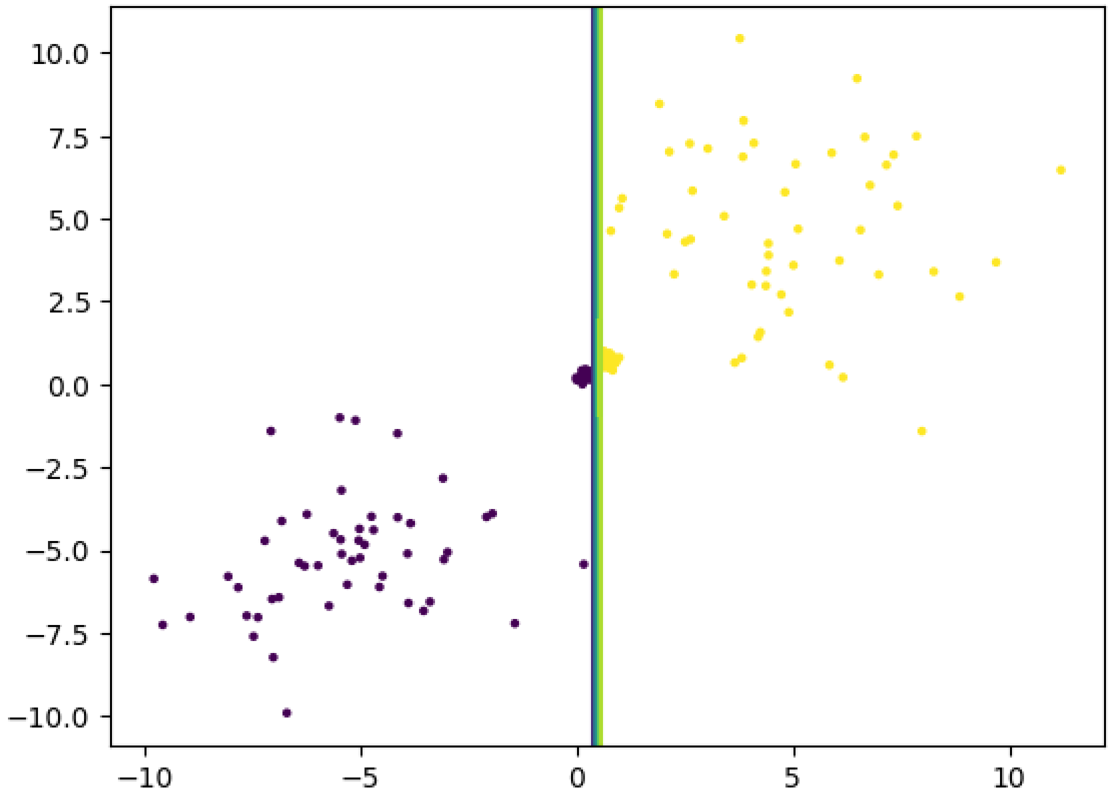
```
Out[ ]: Text(0, 0.5, 'Number of Support Vectors')
```





In [ ]:
```
### 4.5 Boosted decision trees
boosted_tree = AdaBoostClassifier(random_state=0, n_estimators=10)
boosted_tree.fit(X_data, np.reshape(Y_data, (len(Y_data),)))

# W/ norm data
boosted_tree2 = AdaBoostClassifier(random_state=0, n_estimators=10)
boosted_tree2.fit(X_norm, np.reshape(Y_data, (len(Y_data),)))


plt.figure(3)
ax = plt.axes()

tree_boundary = DecisionBoundaryDisplay.from_estimator(boosted_tree, X_data,
                                                        response_method="decisio
                                                        plot_method="contour", a
tree_boundary.ax_.scatter(X_data[:, 0], X_data[:, 1], c=Y_data, s=5)

tree_boundary2 = DecisionBoundaryDisplay.from_estimator(boosted_tree2, X_norm,
                                                         response_method="decisi
                                                         plot_method="contour",
tree_boundary2.ax_.scatter(X_norm[:, 0], X_norm[:, 1], c=Y_data, s=5)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x292c24640>
```



```
In [ ]:
```

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.

CS 671: Homework 2

Alex Kumar

Question 5

In [ ]:
```
### Imports
import math as m
import csv
from sklearn.model_selection import train_test_split
```

In [ ]:
```
### 5.2 Read files and preprocess
# Make dictionary with key: word, value: index
d = {}
with open("dict.txt") as f:
    for word in f:
        w, i = word.split()
        d[w] = i


def dictMaker(line, d):
    # Make dict for x_i: key: word index, value: indicator
    temp_d = {}
    for word in line.split(" "):
        if word in d:
            temp_d[d[word]] = 1
    return temp_d

X, y = [], []
# Read each line and process data
with open("moviereview.tsv") as f:
    tsv = csv.reader(f, delimiter="\t")
    for line in tsv:
        temp_d = dictMaker(line[1], d)
        X.append(temp_d)                    # dict
        y.append(int(line[0]))              # label
```

In [ ]:
```
### 5.3 Binary logistic regression
T, nu, theta = 30, 0.1, {}

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randon

for k in d.keys():
    # dict with key: word idx, value: init 0
    theta[d[k]] = 0
```

```
def dotprod(theta, x_i):
    # dot product of theta and x_i
    total = 0
    for idx in x_i.keys():
        total += theta[idx]
    return total

for i in range(T):
    print("########## EPOCH: ", i, " ###########")
    for i in range(len(x_train)):
        x_i, y_i = x_train[i], y_train[i]   # [x: dict, y: label]

        dotproduct = dotprod(theta, x_i)
        p_i = -y_i + ( (m.e**(dotproduct)) / (1 + m.e**(dotproduct)) )

        for j in x_i.keys():
            theta[j] = theta[j] - nu * p_i
```

```
########## EPOCH:  0  ###########
########## EPOCH:  1  ###########
########## EPOCH:  2  ###########
########## EPOCH:  3  ###########
########## EPOCH:  4  ###########
########## EPOCH:  5  ###########
########## EPOCH:  6  ###########
########## EPOCH:  7  ###########
########## EPOCH:  8  ###########
########## EPOCH:  9  ###########
########## EPOCH:  10  ###########
########## EPOCH:  11  ###########
########## EPOCH:  12  ###########
########## EPOCH:  13  ###########
########## EPOCH:  14  ###########
########## EPOCH:  15  ###########
########## EPOCH:  16  ###########
########## EPOCH:  17  ###########
########## EPOCH:  18  ###########
########## EPOCH:  19  ###########
########## EPOCH:  20  ###########
########## EPOCH:  21  ###########
########## EPOCH:  22  ###########
########## EPOCH:  23  ###########
########## EPOCH:  24  ###########
########## EPOCH:  25  ###########
########## EPOCH:  26  ###########
########## EPOCH:  27  ###########
########## EPOCH:  28  ###########
########## EPOCH:  29  ###########
```

In [ ]:
```
### Predictions
def updateConfusion(confusion, pred, y):
    if y == 1:                     # P
        if pred >= 0.5:              # TP
            confusion[0] += 1
        else:                      # FP
            confusion[1] += 1
    else:                          # N
```

```
        if pred < 0.5:              # TN
            confusion[3] += 1
        else:                     # FN
            confusion[2] += 1
    return confusion


confusion = [0, 0, 0, 0]        # [TP, FP, FN, TN]
for i in range(len(x_test)):
    x, y = x_test[i], y_test[i]

    dot = dotprod(theta, x)
    pred = m.e**(dot) / (1 + m.e**(dot))

    confusion = updateConfusion(confusion, pred, y)
print("[TP, FP, FN, TN]: ", confusion)
```

```
[TP, FP, FN, TN]:  [102, 8, 23, 107]
```

```
In [ ]:
```

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.

CS 671: Homework 2

Alex Kumar

Question 6

In [ ]:
```
### Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [ ]:
```
### 6.1 Read and split data
df = pd.read_csv("Titanic.csv")
df = df.dropna(axis=1)
df["Sex"].replace(["female", "male"], [1, 0], inplace=True)

x, y = df[["Pclass", "Sex", "SibSp", "Parch", "Fare"]], df[["Survived"]]
y = np.ravel(y)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, randon
```

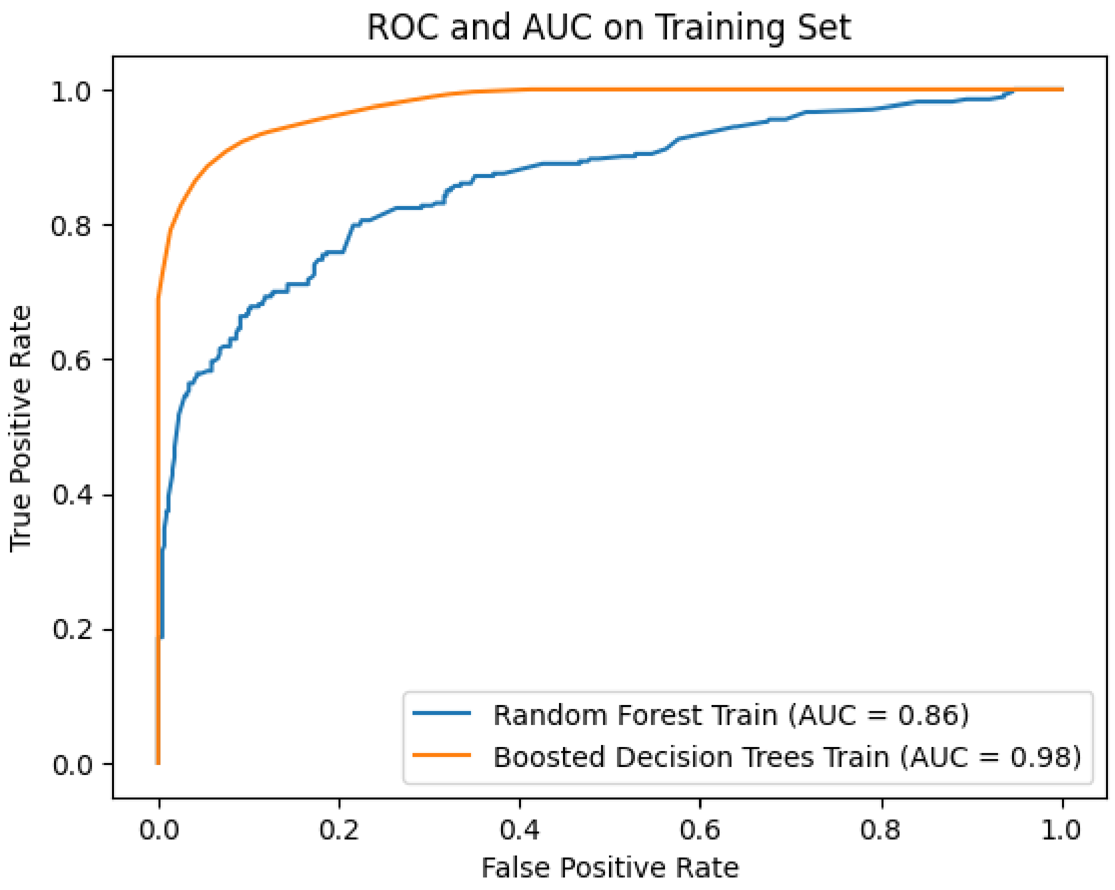
In [ ]:
```
### 6.2 Train and fit
# Random forest
forest = RandomForestClassifier()
start = time.time()
forest.fit(x_train, y_train)
stop = time.time()
print("Time to train Random Forest: ", str(stop-start))

# Boosted trees
trees = AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1))
start = time.time()
trees.fit(x_train, y_train)
stop = time.time()
print("Time to train Boosted Decision Trees: ", str(stop-start))
```

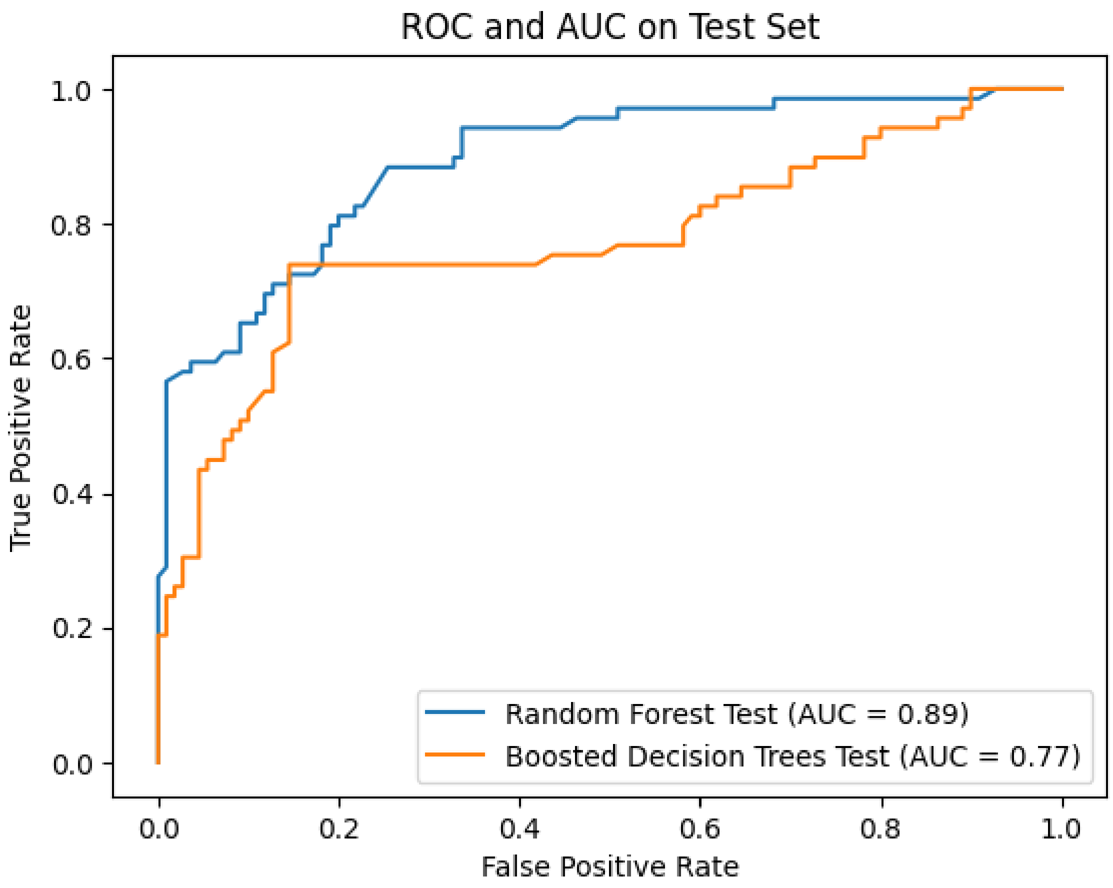
```
Time to train Random Forest:  0.07824397087097168
Time to train Boosted Decision Trees:  0.032958030700683594
```

In [ ]:
```
### 6.3 Grid Search and ROC/AUC
parameters = {"n_estimators": [10, 50, 100, 300], "max_depth": [1, 2, 3], "crit

forest_scores = {}
trees_scores = {}
kf = KFold(n_splits=5)
for n in parameters["n_estimators"]:
    for d in parameters["max_depth"]:
        for c in parameters["criterion"]:
            for train_i, valid_i in kf.split(x, y):
                x_train_temp, x_valid = x.iloc[train_i], x.iloc[valid_i]
                y_train_temp, y_valid = y[train_i], y[valid_i]

                forest = RandomForestClassifier(n_estimators=n, max_depth=d, cr
                trees = AdaBoostClassifier(n_estimators=n, base_estimator=Decis

                forest.fit(x_train_temp, y_train_temp)
                trees.fit(x_train_temp, y_train_temp)

                forest_score = forest.score(x_valid, y_valid)
                trees_score = trees.score(x_valid, y_valid)

                forest_scores[forest_score] = [n, d, c]
                trees_scores[trees_score] = [n, d, c]
```

In [ ]:
```
def roc_maker(y, pred, name, idx):
    fpr, tpr, thresholds = metrics.roc_curve(y, pred)
    roc_auc = metrics.auc(fpr, tpr)
    display = metrics.RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc, estima
    display.plot(ax=idx)


best_forest_p = forest_scores[max(forest_scores.keys())]
best_trees_p = trees_scores[max(trees_scores.keys())]

opt_forest = RandomForestClassifier(n_estimators=best_forest_p[0],
                                    max_depth=best_forest_p[1], criterion=best_
opt_trees = AdaBoostClassifier(n_estimators=best_trees_p[0],
                               base_estimator=DecisionTreeClassifier(max_depth=
                                                                      criterion=

forest.fit(x_train, y_train)
trees.fit(x_train, y_train)

forest_train_pred = forest.predict_proba(x_train)[:, 1]
trees_train_pred = trees.predict_proba(x_train)[:, 1]
plt.figure(1)
ax = plt.axes()
roc_maker(y_train, forest_train_pred, "Random Forest Train", ax)
roc_maker(y_train, trees_train_pred, "Boosted Decision Trees Train", ax)
plt.title("ROC and AUC on Training Set")
plt.show()

forest_test_pred = forest.predict_proba(x_test)[:, 1]
trees_test_pred = trees.predict_proba(x_test)[:, 1]
plt.figure(2)
```

```
ax = plt.axes()
roc_maker(y_test, forest_test_pred, "Random Forest Test", ax)
roc_maker(y_test, trees_test_pred, "Boosted Decision Trees Test", ax)
plt.title("ROC and AUC on Test Set")
plt.show()
```




ROC and AUC on Test Set
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Random Forest Test (AUC = 0.89)
Boosted Decision Trees Test (AUC = 0.77)

```
In [ ]:
```

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.

CS 671: Homework 2

Alex Kumar

Question 7

In [ ]:
```
### Imports
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from interpret.glassbox import ExplainableBoostingClassifier
from interpret import show
from IPython.display import Image, display
```

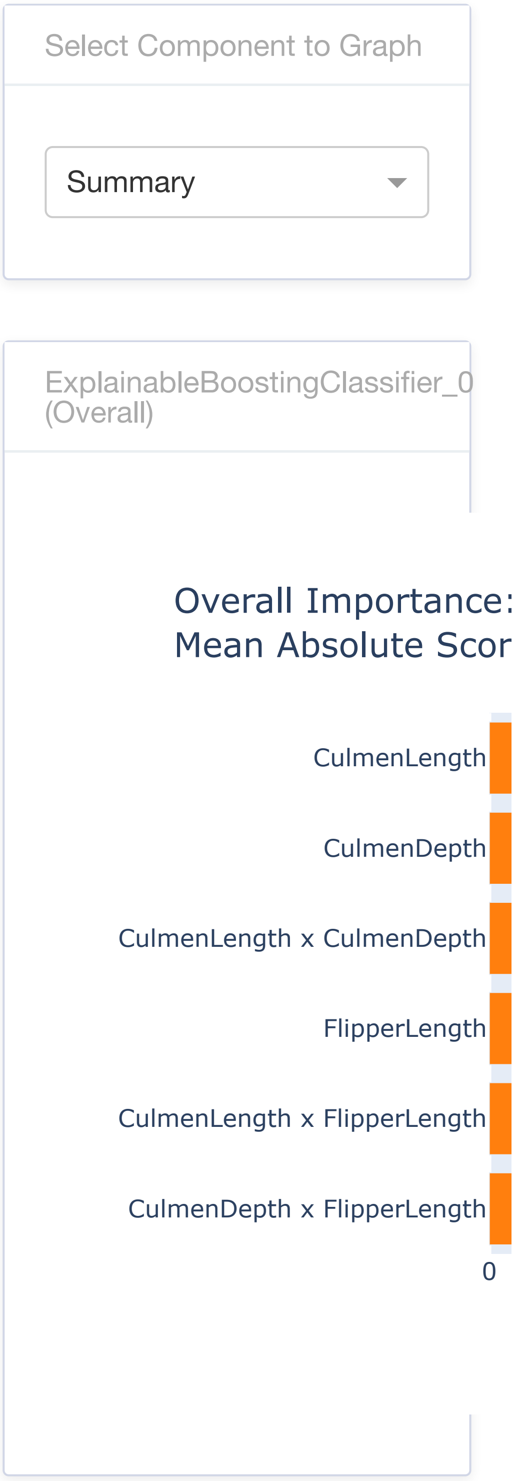
In [ ]:
```
### Read and split data
df = pd.read_csv("penguins_trunc.csv")
x, y = df[["CulmenLength", "CulmenDepth", "FlipperLength"]], df[["Species"]]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, randon

ebm = ExplainableBoostingClassifier(random_state=0)
ebm.fit(x_train, y_train)
```

Out[ ]:
```
▾         ExplainableBoostingClassifier
ExplainableBoostingClassifier(random_state=0)
```

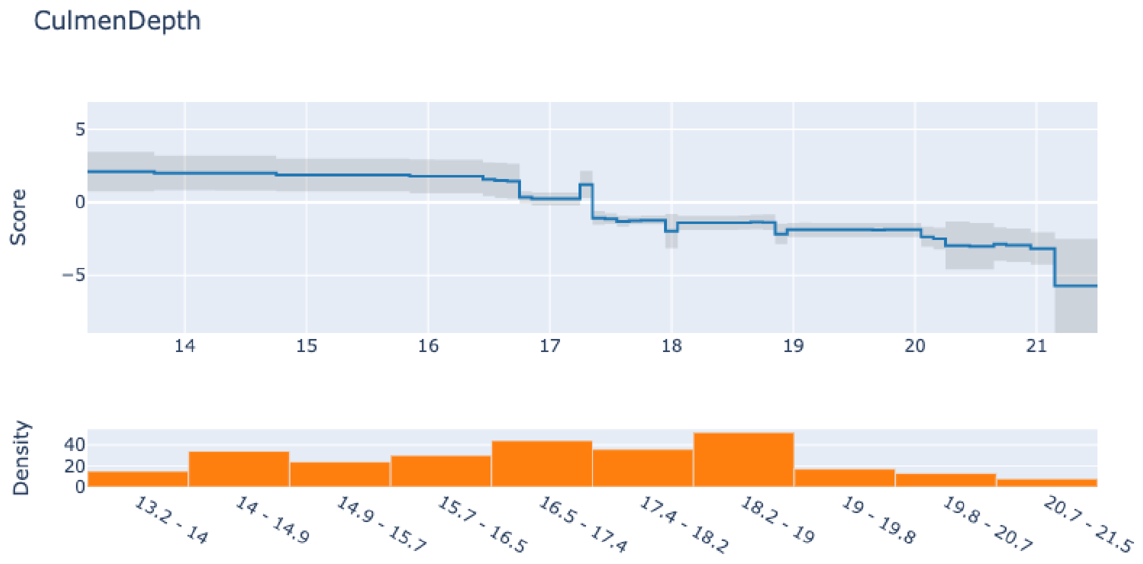
In [ ]:
```
### Accuracy and plotted shape functions
train_score = ebm.score(x_train, y_train)
test_score = ebm.score(x_test, y_test)
print(train_score)
print(test_score)

ebm_global = ebm.explain_global()
show(ebm_global)
```

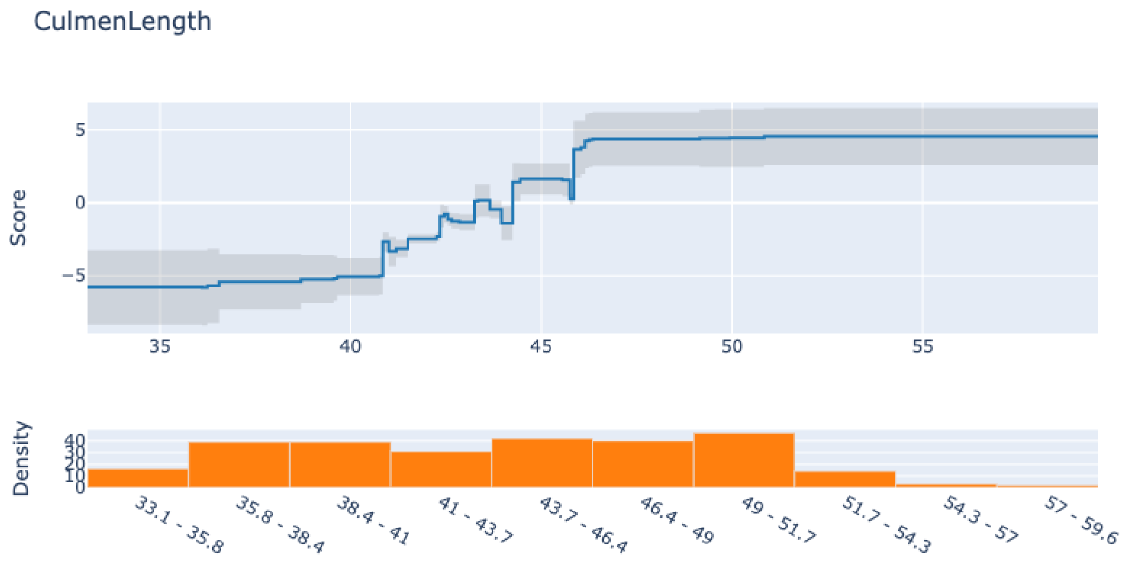
```
1.0
0.9855072463768116
```

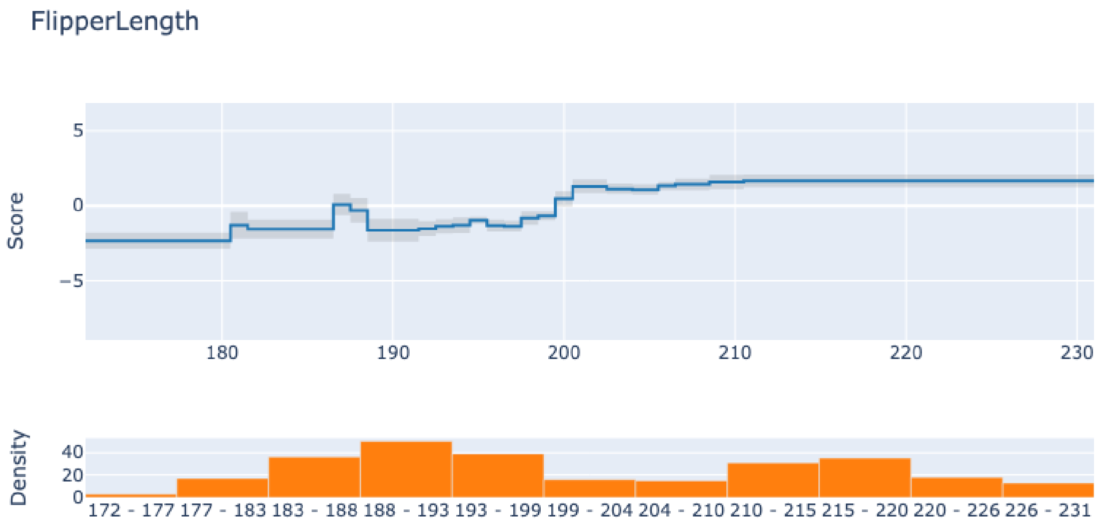
```
/Users/alexanderkumar/miniconda3/envs/671/lib/python3.8/site-packages/interpre
t/visual/udash.py:5: UserWarning:
The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`
  import dash_html_components as html
/Users/alexanderkumar/miniconda3/envs/671/lib/python3.8/site-packages/interpre
t/visual/udash.py:6: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
/Users/alexanderkumar/miniconda3/envs/671/lib/python3.8/site-packages/interpre
t/visual/udash.py:7: UserWarning:
The dash_table package is deprecated. Please replace
`import dash_table` with `from dash import dash_table`

Also, if you're using any of the table format helpers (e.g. Group), replace
`from dash_table.Format import Group` with
`from dash.dash_table.Format import Group`
  import dash_table as dt
```


Select Component to Graph
Summary
ExplainableBoostingClassifier_0 (Overall)
Overall Importance:
Mean Absolute Scor
CulmenLength
CulmenDepth
CulmenLength x CulmenDepth
FlipperLength
CulmenLength x FlipperLength
CulmenDepth x FlipperLength
0

```
In [ ]: ### Display component graphs from above (saved as pngs)
        display(Image(filename='CulmenDepth.png'),
                Image(filename='CulmenLength.png'),
                Image(filename='FlipperLength.png'))
```


CulmenDepth
Score
5
0
−5
14
15
16
17
18
19
20
21
Density
40
20
0
13.2 - 14
14 - 14.9
14.9 - 15.7
15.7 - 16.5
16.5 - 17.4
17.4 - 18.2
18.2 - 19
19 - 19.8
19.8 - 20.7
20.7 - 21.5


CulmenLength
Score
5
0
−5
35
40
45
50
55
Density
40
30
20
10
0
33.1 - 35.8
35.8 - 38.4
38.4 - 41
41 - 43.7
43.7 - 46.4
46.4 - 49
49 - 51.7
51.7 - 54.3
54.3 - 57
57 - 59.6


FlipperLength
Score
5
0
−5
180
190
200
210
220
230
Density
40
20
0
172 - 177
177 - 183
183 - 188
188 - 193
193 - 199
199 - 204
204 - 210
210 - 215
215 - 220
220 - 226
226 - 231