Support Vector Machines for Classification II

Swati Mishra

Applications of Machine Learning (4AL3)

Fall 2024

McMaster University
ENGINEERING

Review

- Concept of Hyperplane
- Concept of Supporting Vectors
- Support Vector Machine Classifier
- Optimization Function

Clarification: Organizing data

```
import numpy as np

observations = 10
features = 3

w = np.random.rand(features)
x = np.random.rand(features,observations)
b = np.ones(observations)
y = w.T.dot(x)+b
print(y)
y = np.dot(w,x)+b
print(y)
```

[21] ✓ 0.0s

```
[2.1119416  1.74867323 1.69624198 1.53914327 2.54255963 1.48688518
 1.28003225 2.50495114 1.85766473 2.26048399]
[2.1119416  1.74867323 1.69624198 1.53914327 2.54255963 1.48688518
 1.28003225 2.50495114 1.85766473 2.26048399]
```

```
y = x.T.dot(w)+b
print(y)
y = np.dot(x.T,w)+b
print(y)
```

[22] ✓ 0.0s

```
[2.1119416  1.74867323 1.69624198 1.53914327 2.54255963 1.48688518
 1.28003225 2.50495114 1.85766473 2.26048399]
[2.1119416  1.74867323 1.69624198 1.53914327 2.54255963 1.48688518
 1.28003225 2.50495114 1.85766473 2.26048399]
```

We organize features and observations depending upon what will lead to faster computation.

```
import numpy as np

observations = 10
features = 3

w = np.random.rand(features)
x = np.random.rand(observations,features)
b = np.ones(observations)
y = x.dot(w)+b
print(y)
y = np.dot(x,w)+b
print(y)
```

34] ✓ 0.0s

```
[1.46765968 1.50196899 1.4517751  1.86181566 1.71815237 1.57761492
 1.2321935  1.69826303 1.52414013 1.45504751]
[1.46765968 1.50196899 1.4517751  1.86181566 1.71815237 1.57761492
 1.2321935  1.69826303 1.52414013 1.45504751]
```

Try this when confused!

```
y = w.T.dot(x.T)+b
print(y)
y = np.dot(w.T,x.T)+b
print(y)
```

39] ✓ 0.0s

```
[1.46765968 1.50196899 1.4517751  1.86181566 1.71815237 1.57761492
 1.2321935  1.69826303 1.52414013 1.45504751]
[1.46765968 1.50196899 1.4517751  1.86181566 1.71815237 1.57761492
 1.2321935  1.69826303 1.52414013 1.45504751]
```

Classification Problem

- Diagnostic of Breast Cancer Wisconsin
- Classification goal to classify if a tissue is cancerous.

Variables Table

Variable Name	Role	Type
ID	ID	Categorical
Diagnosis	Target	Categorical
radius1	Feature	Continuous
texture1	Feature	Continuous
perimeter1	Feature	Continuous
area1	Feature	Continuous
smoothness1	Feature	Continuous
compactness1	Feature	Continuous
concavity1	Feature	Continuous
concave_points1	Feature	Continuous

Variables Table

Variable Name	Role	Type
symmetry1	Feature	Continuous
fractal_dimension1	Feature	Continuous
radius2	Feature	Continuous
texture2	Feature	Continuous
perimeter2	Feature	Continuous
area2	Feature	Continuous
smoothness2	Feature	Continuous
compactness2	Feature	Continuous
concavity2	Feature	Continuous
concave_points2	Feature	Continuous

Variables Table

Variable Name	Role	Type
symmetry2	Feature	Continuous
fractal_dimension2	Feature	Continuous
radius3	Feature	Continuous
texture3	Feature	Continuous
perimeter3	Feature	Continuous
area3	Feature	Continuous
smoothness3	Feature	Continuous
compactness3	Feature	Continuous
concavity3	Feature	Continuous
concave_points3	Feature	Continuous

Optimization Problem

How do I implement this?

- With the above loss, training objective becomes

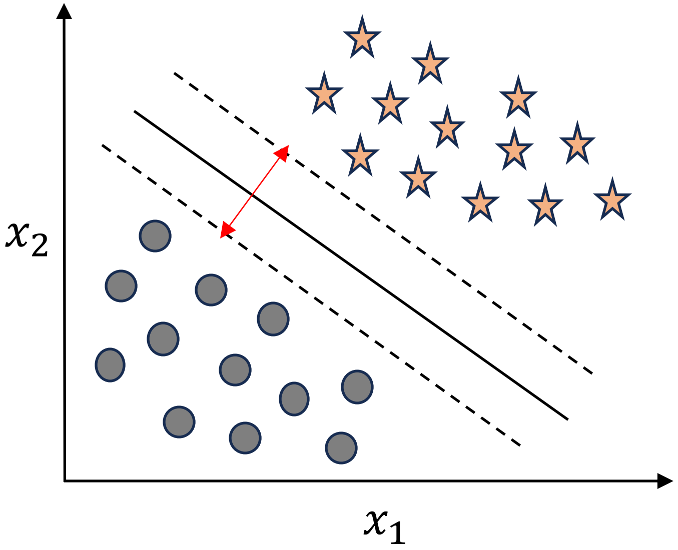
$$\text{maximize} \quad M$$

given $\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots \epsilon_n, M$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1,$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$



Optimization Problem

- The previous optimization problem can be re-written as:

$$\underset{b, w}{minimize} \left\{ C \sum_{i=1}^{n} \max[0, 1 - y_i(b + w.x_i)] + \frac{1}{2} ||w||^2 \right\}$$

Optimization Problem

- The previous optimization problem can be re-written as:

$$\underset{b,w}{\text{minimize}} \left\{ \underbrace{C \sum_{i=1}^{n} \max[0, 1 - y_i(b + w.x_i)]}_{\text{Loss}} + \underbrace{\frac{1}{2}||w||^2}_{\text{Regularization / Penalty}} \right\}$$

Optimization Problem

- For efficient computing, it is practical to combine w and b in one weight matrix:

$$\boldsymbol{W} = |w_1, w_2\ , w_3\ , \ldots., w_p, b\ |$$

p = number of features

Optimization Problem

- For efficient computing, it is practical to combine w and b in one weight matrix:

$$\boldsymbol{W} = |w_1, w_2\ , w_3\ , \ldots., w_p, b\ |$$

p = number of features

- Therefore, our optimization problem becomes:

$$\underset{w}{minimize} \quad \left\{ \underbrace{C \sum_{i=1}^{n} \max[0, 1\ - y_i(\boldsymbol{w}.x_i)]}_{\text{Loss}} + \underbrace{\frac{1}{2}||\boldsymbol{w}||^2}_{\text{Regularization / Penalty}} \right\}$$

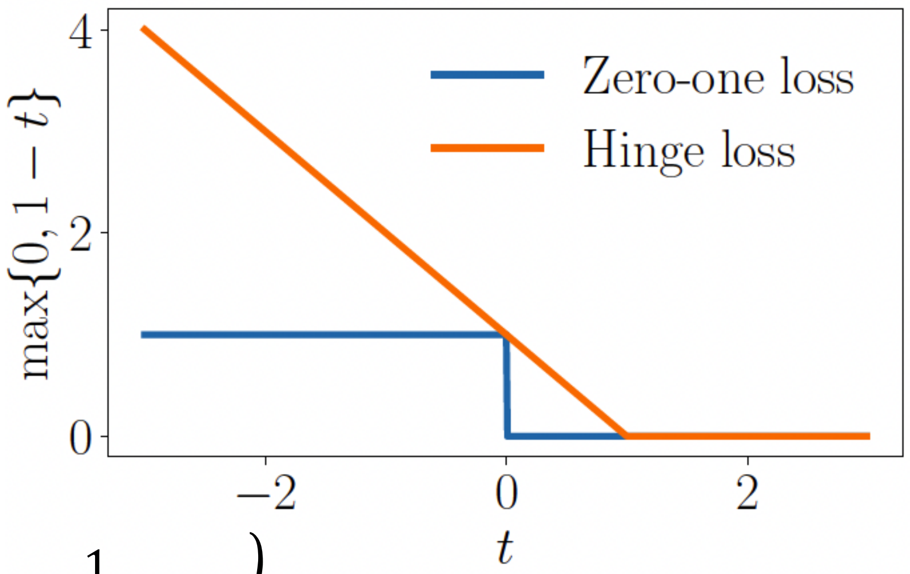
Loss Function

- Our optimization goal is:

$$\underset{w}{minimize} \left\{ \underbrace{C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)]} + \frac{1}{2}||\boldsymbol{w}||^2 \right\}$$

$$\ell(y) = \max(0, 1 - z.y)$$

Hinge Loss Function



- Our optimization goal is :

$$\underset{w}{minimize} \left\{ \underbrace{C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)]} + \frac{1}{2}||\boldsymbol{w}||^2 \right\}$$

$$\ell(y) = \max(0,1 - z.y)$$

This is hinge loss defined for an **intended** output $\boldsymbol{z} = \pm 1$, and a **predicted** classifier score $\boldsymbol{y}$.

Gradient of Hinge Loss

- For an intended output $z = \pm 1$, and a predicted classifier score y **hinge loss** is defined as:

$$\ell(y) = \max(0, 1 - z.y)$$

  - Hinge loss is convex function
  - It is not differentiable

Gradient of Hinge Loss

- For an intended output $z = \ \pm 1$, and a predicted classifier score y **hinge loss** is defined as:

$$\ell(y) = \max(0,1 - z.y)$$

  - Hinge loss is convex function
  - It is not differentiable
  - Gradient of hinge loss w.r.t. w is given by: $\frac{\partial \ell}{\partial w} = \begin{cases} 0, & y_i(\boldsymbol{w}.x_i) \geq 1 \\ -y_i x_i, & y_i(\boldsymbol{w}.x_i) < 1 \end{cases}$

Gradient of Hinge Loss

- For an intended output $z = \pm 1$, and a predicted classifier score y **hinge loss** is defined as:

$$\ell(y) = \max(0, 1 - z.y)$$

  - Hinge loss is convex function
  - It is not differentiable
  - Gradient of hinge loss w.r.t. w is given by: $\frac{\partial \ell}{\partial w} = \begin{cases} 0, & y_i(\boldsymbol{w}.x_i) \geq 1 \\ -y_i x_i, & y_i(\boldsymbol{w}.x_i) < 1 \end{cases}$

Gradient of Hinge Loss

- For an intended output $z = \pm 1$, and a predicted classifier score y **hinge loss** is defined as:

$$\ell(y) = \max(0,1 - z.y)$$

  - Hinge loss is convex function
  - It is not differentiable
  - Gradient of hinge loss w.r.t. w is given by: $\frac{\partial \ell}{\partial w} = \begin{cases} 0, & y_i(\boldsymbol{w}.x_i) \geq 1 \\ -y_i x_i, & y_i(\boldsymbol{w}.x_i) < 1 \end{cases}$

  - Total loss :

$$L = C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$$

Gradient of Hinge Loss

- For an intended output $z = \pm 1$, and a predicted classifier score y **hinge loss** is defined as:

$$\ell(y) = \max(0,1 - z.y)$$

  - Hinge loss is convex function
  - It is not differentiable
  - Gradient of hinge loss w.r.t. w is given by: $\frac{\partial \ell}{\partial w} = \begin{cases} 0, & y_i(\boldsymbol{w}.x_i) \geq 1 \\ -y_i x_i, & y_i(\boldsymbol{w}.x_i) < 1 \end{cases}$

- Total loss :

$$L = C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2} ||\boldsymbol{w}||^2$$

- Gradient of Total Loss:

$$\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - C y_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$$

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize model parameters.
  - Step 2: Compute the loss gradients
  - Step 3: Compute the loss
  - Step 4: Update the weights
  - Step 5: Repeat 2-4 for E epochs

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
    - Step 1: Initialize weights (w), learning rate (η), epoch (E)
    - Step 2: Compute the loss gradients $\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - Cy_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$
    - Step 3: Compute the loss $C\sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$
    - Step 4: Update the weights $\mathrm{w}' = \mathrm{w} - \eta\frac{\partial L}{\partial w}$
    - Step 5: Repeat 2-4 for E epochs

McMaster University

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients $\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \le 0 \\ \boldsymbol{w} - Cy_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$
  - Step 3: Compute the loss $C\sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$
  - Step 4: Update the weights $\mathrm{w}' = \mathrm{w} - \eta\frac{\partial L}{\partial w}$
  - Step 5: Repeat 2-4 for E epochs

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients $\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - C y_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$
  - Step 3: Compute the loss $C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$
  - Step 4: Update the weights $\mathrm{w}' = \mathrm{w} - \eta \frac{\partial L}{\partial w}$
  - Step 5: Repeat 2-4 for E epochs

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients $\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - Cy_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$
  - Step 3: Compute the loss $C\sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$
  - Step 4: Update the weights $\mathrm{w}' = \mathrm{w} - \eta \frac{\partial L}{\partial w}$
  - Step 5: Repeat 2-4 for E epochs

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients $\frac{\partial L}{\partial w} = \begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - C y_i x_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$
  - Step 3: Compute the loss $C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w}.x_i)] + \frac{1}{2} ||\boldsymbol{w}||^2$
  - Step 4: Update the weights $\mathrm{w}' = \mathrm{w} - \eta \frac{\partial L}{\partial w}$
  - Step 5: Repeat 2-4 for E epochs

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - **Step 2**: Compute the loss gradients
  - Step 3: Compute the loss
  - Step 4: Update the weights
  - Step 5: Repeat 2-4 for E epochs

$$\begin{cases} \boldsymbol{w} + 0, & 1 - y_i(\boldsymbol{w}.x_i) \leq 0 \\ \boldsymbol{w} - Cy_ix_i, & 1 - y_i(\boldsymbol{w}.x_i) > 0 \end{cases}$$

```python
def compute_gradient(self,X,Y):
    # organize the array as vector
    X_ = np.array([X])

    # hinge loss
    hinge_distance = 1 - (Y* np.dot(X_,self.weights))

    total_distance = np.zeros(len(self.weights))
    # hinge loss is not defined at 0
    # is distance equalt to 0
    if max(0, hinge_distance[0]) == 0:
        total_distance += self.weights
    else:
        total_distance += self.weights - (self.C * Y[0] * X_[0])

    return total_distance
```

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients
  - Step 3: Compute the loss
  - **Step 4**: Update the weights $\mathrm{w}' = \mathrm{w} - \eta \dfrac{\partial L}{\partial w}$
  - Step 5: Repeat 2-4 for E epochs

```python
def stochastic_gradient_descent(self,X,Y):

    # execute the stochastic gradient des   cent function for defined epochs
    for epoch in range(self.epoch):

        # shuffle to prevent repeating update cycles
        features, output = shuffle(X, Y)

        for i, feature in enumerate(features):
            gradient = self.compute_gradient(feature, output[i])
            self.weights = self.weights - (self.learning_rate * gradient)

        #print epoch if it is equal to thousand - to minimize number of prints
        if epoch%1000 ==0:
            loss = self.compute_loss(features, output)
            print("Epoch is: {} and Loss is (not computed): {}".format(epoch, loss))

        #check for convergence
```

Training SVMs

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients
  - Step 3: Compute the loss
  - Step 4: Update the weights
  - **Step 5**: Repeat 2-4 for E epochs

```
def stochastic_gradient_descent(self,X,Y):

    # execute the stochastic gradient des   cent function for defined epochs
    for epoch in range(self.epoch):

        # shuffle to prevent repeating update cycles
        features, output = shuffle(X, Y)

        for i, feature in enumerate(features):
            gradient = self.compute_gradient(feature, output[i])
            self.weights = self.weights - (self.learning_rate * gradient)

        #print epoch if it is equal to thousand - to minimize number of prints
        if epoch%1000 ==0:
            loss = self.compute_loss(features, output)
            print("Epoch is: {} and Loss is (not computed): {}".format(epoch, loss))

        #check for convergence
```

Training SVMs

Why are we shuffling the data?

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients
  - Step 3: Compute the loss
  - Step 4: Update the weights
  - Step 5: Repeat 2-4 for E epochs

```
def stochastic_gradient_descent(self,X,Y):

    # execute the stochastic gradient des   cent function for defined epochs
    for epoch in range(self.epoch):

        # shuffle to prevent repeating update cycles
        features, output = shuffle(X, Y)

        for i, feature in enumerate(features):
            gradient = self.compute_gradient(feature, output[i])
            self.weights = self.weights - (self.learning_rate * gradient)

        #print epoch if it is equal to thousand - to minimize number of prints
        if epoch%1000 ==0:
            loss = self.compute_loss(features, output)
            print("Epoch is: {} and Loss is (not computed): {}".format(epoch, loss))

        #check for convergence
```

Training SVMs

Where is this loss function used?

- To train the SVMs we can employ Stochastic Gradient Descent.
- Stochastic Gradient Descent is an online algorithm that minimizes the loss function by computing its gradient one sample at a time.
  - Step 1: Initialize weights (w), learning rate (η), epoch (E)
  - Step 2: Compute the loss gradients
  - **Step 3**: Compute the loss
  - Step 4: Update the weights
  - Step 5: Repeat 2-4 for E epochs

$$C\sum_{i=1}^{n}\max[0, 1 - y_i(\boldsymbol{w}.x)] + \frac{1}{2}||\boldsymbol{w}||^2$$

Kernel Trick

- What happens if the data we are working with is not linearly separable, like in this case, of solar flares?

Kernel Trick

- What happens if the data we are working with is not linearly separable, like in this case, of solar flares?
- We use polynomial functions of the predictor variables, $(x_{i1},\ x_{i2}, x_{i3}, \ldots, x_{ip})$ so that our feature space becomes, $(x_{i1}, {x_{i1}}^2, x_{i2}, {x_{i2}}^2,\ x_{i3}, {x_{i2}}^2, \ldots, x_{ip},\ {x_{ip}}^2)$.

Kernel Trick

- The optimization function for our polynomial degree function becomes .

$$\text{maximize} \quad M$$

given $\beta_0, \beta_{11}, \beta_{12}, \dots, \beta_{p1}, \beta_{p2}, \epsilon_1, \dots \epsilon_n, M$

subject to $\displaystyle\sum_{j=1}^{p} \beta_j^2 = 1,$

$$y_i(\beta_0 + \textstyle\sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} {x_{ij}}^2) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C, \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1$$

Kernel Trick

But SVM fails when feature space is large?

- The optimization function for our polynomial degree function becomes .

$$\underset{\text{given}\quad \beta_0, \beta_{11}, \beta_{12}, \ldots, \beta_{p1}, \beta_{p2}, \epsilon_1, \ldots \epsilon_n, M}{maximize \quad M}$$

subject to $\sum_{j=1}^{p} \beta_j^2 = 1,$

$$y_i(\beta_0 + \textstyle\sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} {x_{ij}}^2) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C, \sum_{j=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1$$

Kernel Trick

- What happens if the data we are working with is not linearly separable, like in this case, of solar flares.
- We use polynomial functions of the predictor variables, $(x_{i1},\ x_{i2},\ x_{i3},\dots,x_{ip})$ so that our feature space becomes, $(x_{i1}, {x_{i1}}^2, x_{i2}, {x_{i2}}^2,\ x_{i3}, {x_{i2}}^2,\dots, x_{ip},\ {x_{ip}}^2)$.
- We need to controlling for large feature spaces.

Kernel Trick

- What happens if the data we are working with is not linearly separable, like in this case, of solar flares.
- We use polynomial functions of the predictor variables, $(x_{i1},\ x_{i2}, x_{i3}, \ldots, x_{ip})$ so that our feature space becomes, $(x_{i1}, {x_{i1}}^2, x_{i2}, {x_{i2}}^2,\ x_{i3}, {x_{i2}}^2, \ldots, x_{ip},\ {x_{ip}}^2)$.
- Controlling for large feature spaces: Kernel is a function that quantifies the similarity of two observations.

Kernel Trick

- What happens if the data we are working with is not linearly separable, like in this case, of solar flares.
- We use polynomial functions of the predictor variables, $(x_{i1},\ x_{i2}, x_{i3}, \ldots, x_{ip})$ so that our feature space becomes, $(x_{i1}, {x_{i1}}^2, x_{i2}, {x_{i2}}^2,\ x_{i3}, {x_{i2}}^2, \ldots, x_{ip},\ {x_{ip}}^2)$.
- Controlling for large feature spaces: Kernel is a function that quantifies the similarity of two observations.
- Example of a linear Kernel. $$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j},$$
- Example of a polynomial Kernel $$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d.$$

Kernel Trick

- When we solve for this optimization equations, we find that the support vector classifier involves only the inner product of the two observations $x_i, x_{i'}$ making the linear support vector equation

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle,$$

Kernel Trick

- When we solve for this optimization equations, we find that the support vector classifier involves only the inner product of the two observations $x_i, x_{i'}$ making the linear support vector equation

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle,$$

- To estimate $\alpha_i \; and \; \beta_0$, we only need the inner products between all pairs of observation.

Kernel Trick

- When we solve for this optimization equations, we find that the support vector classifier involves only the inner product of the two observations $x_i, x_{i'}$ making the linear support vector equation

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle,$$

- To estimate α_i and β_0, we only need the inner products between all pairs of observation.

- So, for n set of observations, we need to perform $n(n-1)/2$, operations for all pairs.

Kernel Trick

- When we solve for this optimization equations, we find that the support vector classifier involves only the inner product of the two observations $x_i, x_{i'}$ making the linear support vector equation

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle,$$

- To estimate α_i and β_0, we only need the inner products between all pairs of observation.
- So, for n set of observations, we need to perform $n(n-1)/2$, operations for all pairs.
- Now when we are predicting, we take a new observation and compute this inner product with each of the training observations.

Design Considerations

- If the model is overfitting, reduce the polynomial degree. Conversely, if it is underfitting, you can try increasing it.

- SVM's are very sensitive to feature scales.

- Unlike Logistic Regression classifiers, SVM classifiers do not output probabilities for each class.

- At a low polynomial degree, this method cannot deal with very complex datasets.

- A high polynomial degree it creates a huge number of features, making the model too slow.

- When C is small
    - More violations to the margin are tolerated
    - Low-variance but high-bias classifier will result

Readings

***Required Readings*:**

Introduction to Statistical Learning

1. Chapter 9 – Section 9.1 – 9.3 Page 367 – 382

Supplemental Readings (Not required but recommended):

Mathematics for Machine Learning

1. Chapter 12 – Section 12.1 – 12.3 Page 370 – 383

Thank You

McMaster University