CS 484
Introduction to Machine Learning

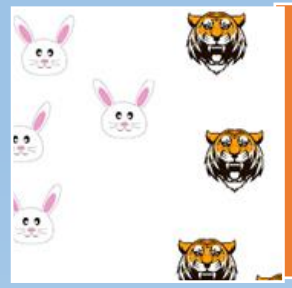
Week 10, March 25, 2021

Spring Semester 2021

ILLINOIS TECH

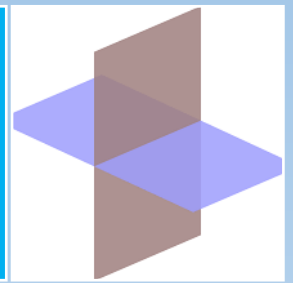
College of Computing

Week 10 Agenda: Support Vector Machine



Motivation

Hyperplane

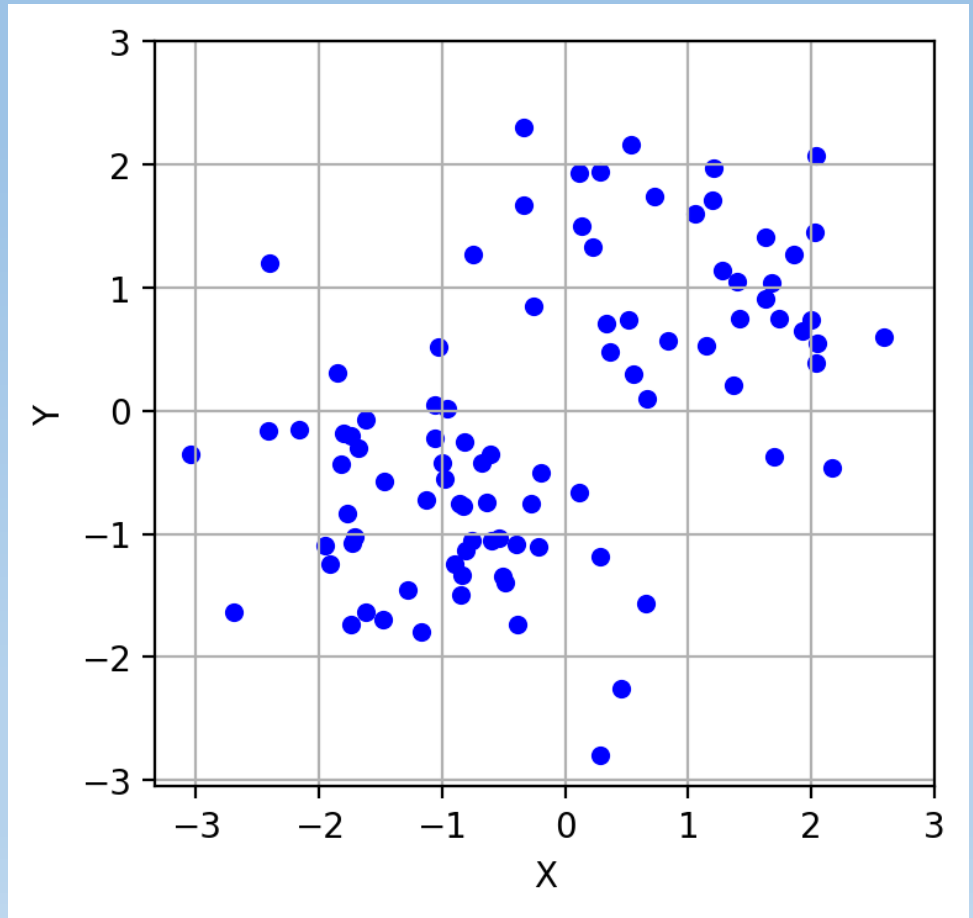


Support Vector

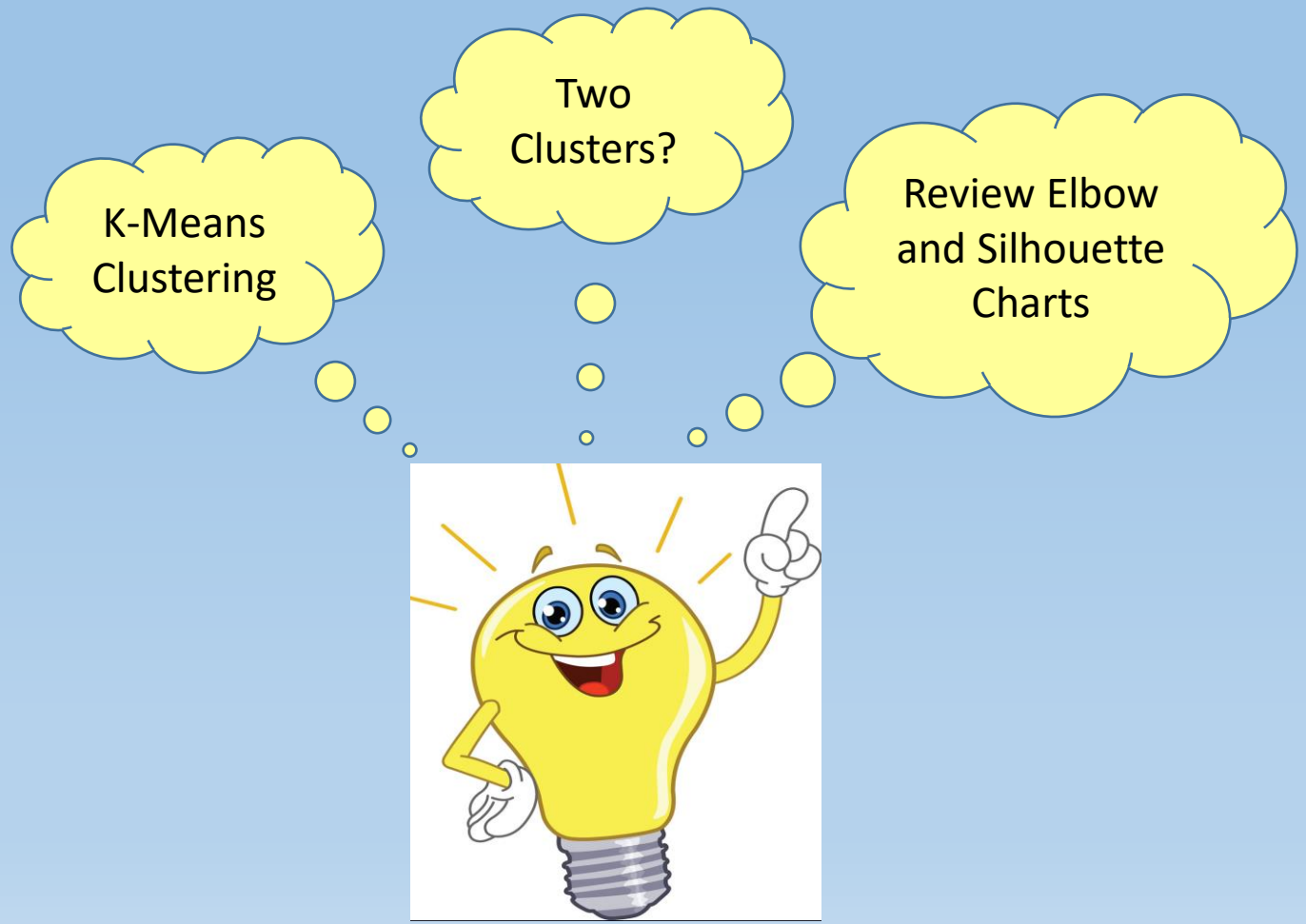
Can You See the Two Groups of Points?

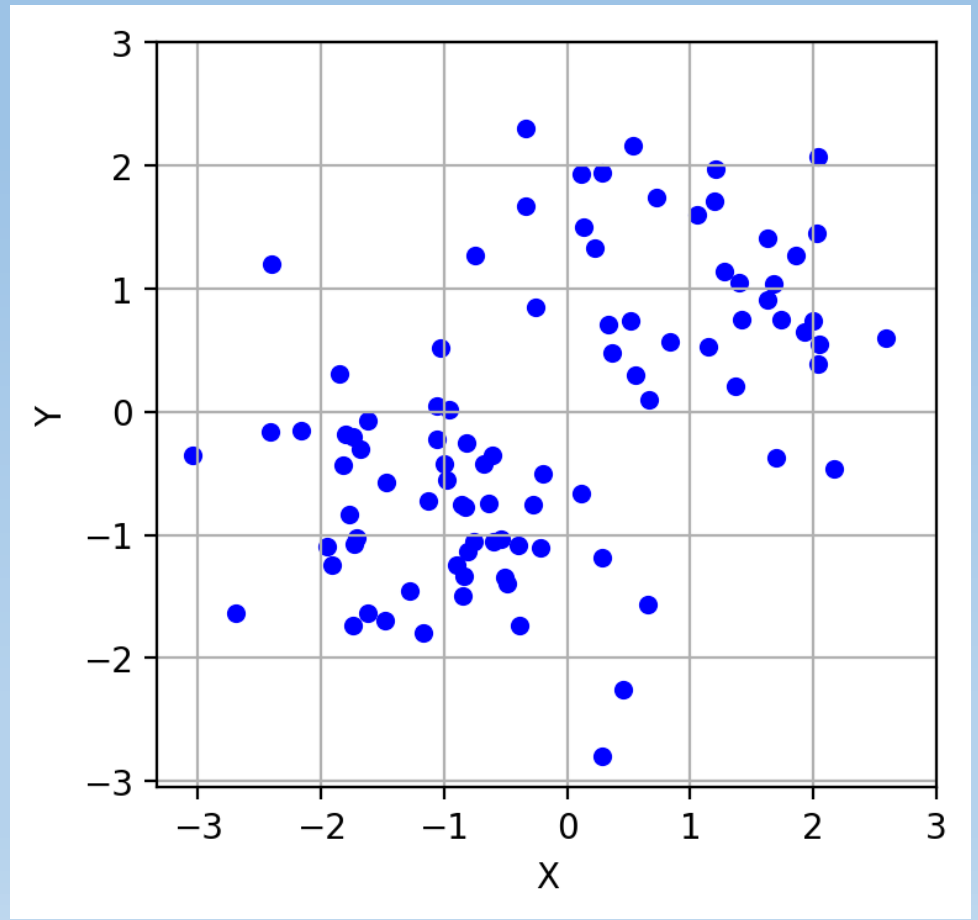
Ninety-Two Observations

Continuous Features X and Y



Assign Observations into Groups





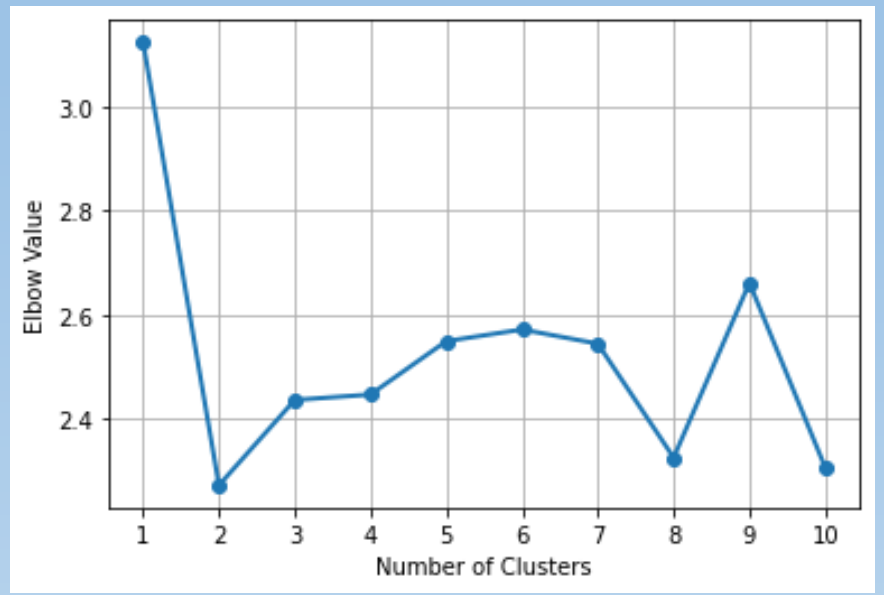
K-Means Cluster

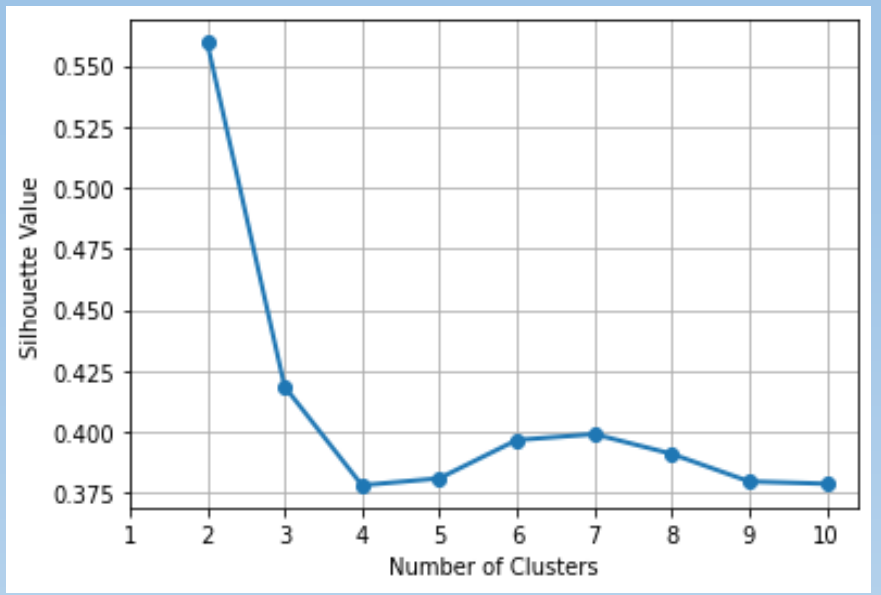
```
nObs = trainData.shape[0]
for c in range(maxNClusters):
   KClusters = c + 1
   nClusters[c] = KClusters

   kmeans = cluster.KMeans(n_clusters=KClusters, random_state=20191106).fit(trainData)

   if (1 < KClusters):
       Silhouette[c] = metrics.silhouette_score(trainData, kmeans.labels_)
   else:
       Silhouette[c] = numpy.NaN

   WCSS = numpy.zeros(KClusters)
   nC = numpy.zeros(KClusters)

   for i in range(nObs):
      k = kmeans.labels_[i]
      nC[k] += 1
      diff = trainData.iloc[i] - kmeans.cluster_centers_[k]
      WCSS[k] += diff.dot(diff)

   Elbow[c] = 0
   for k in range(KClusters):
      Elbow[c] += WCSS[k] / nC[k]
      TotalWCSS[c] += WCSS[k]
```

Week 10 Discover MVN.py

Discover The Two Groups

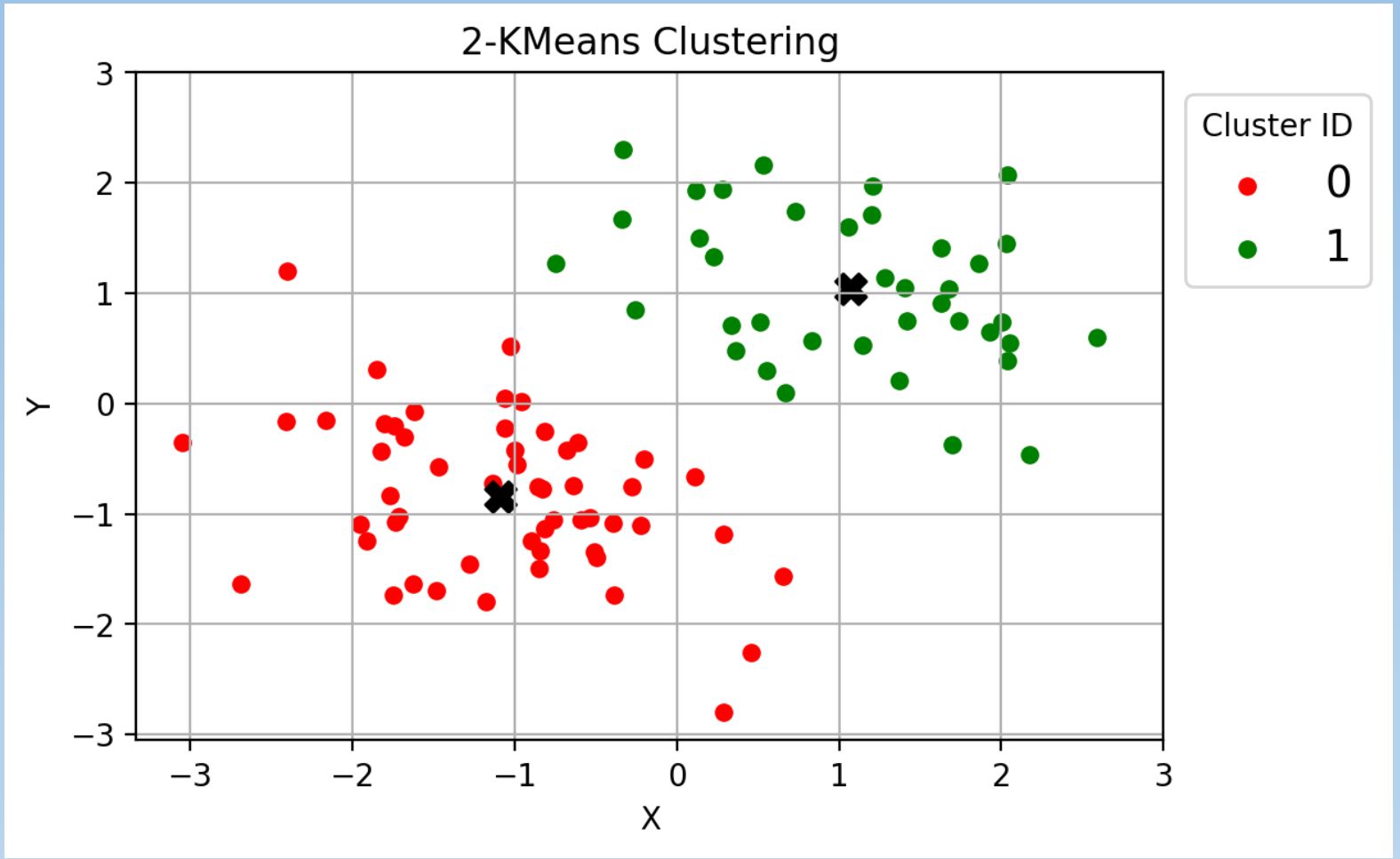
Number of Clusters	Elbow	Silhouette
1	3.1242	
2	2.2718	0.5594
3	2.4359	0.4185
4	2.4465	0.3781
5	2.5491	0.3810
6	2.5716	0.3966
7	2.5442	0.3991
8	2.3250	0.3910
9	2.6614	0.3797
10	2.3061	0.3787





Go with the Two-Cluster solution!

The Two-Cluster Solution



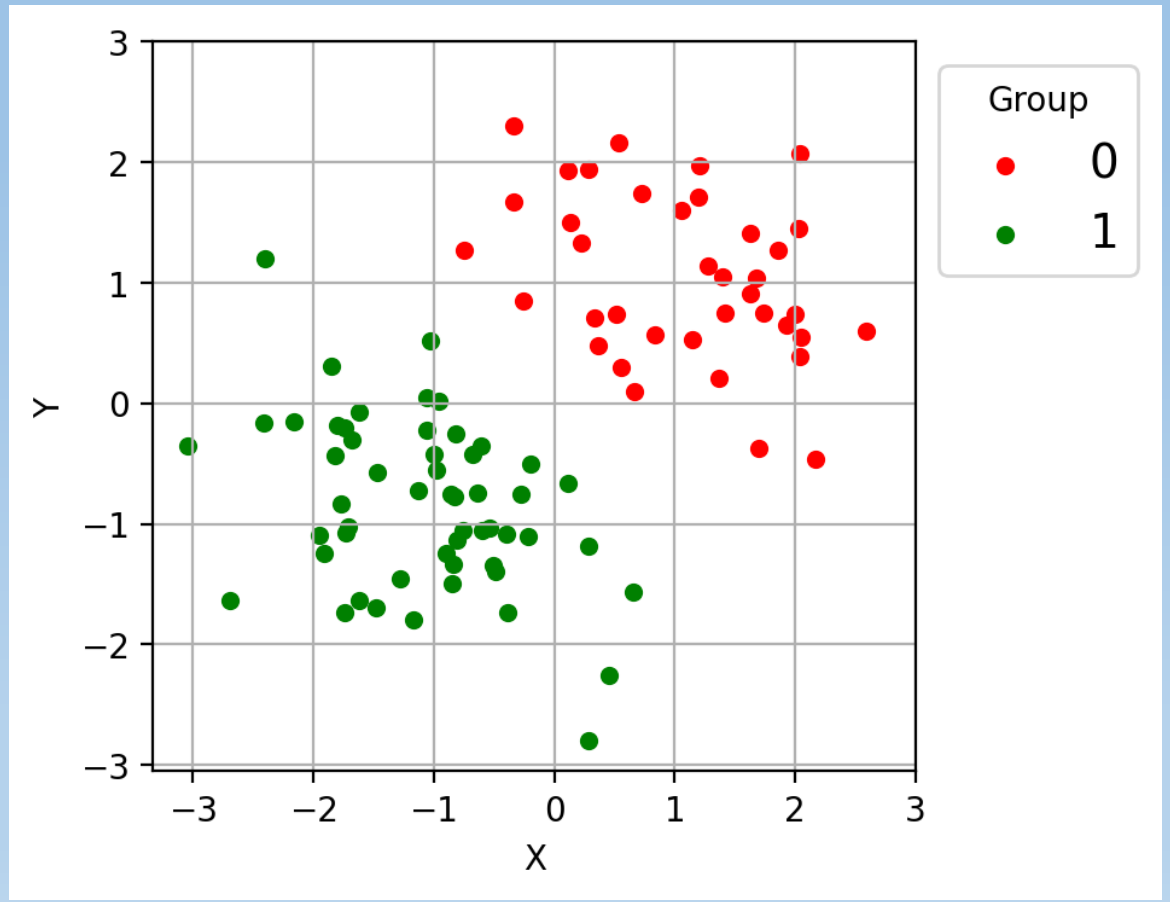
Identify The Two Groups

Nearest Neighbors

Decision Tree

Logistic Regression

Provided Prior Group Identifier



Nearest Neighbors

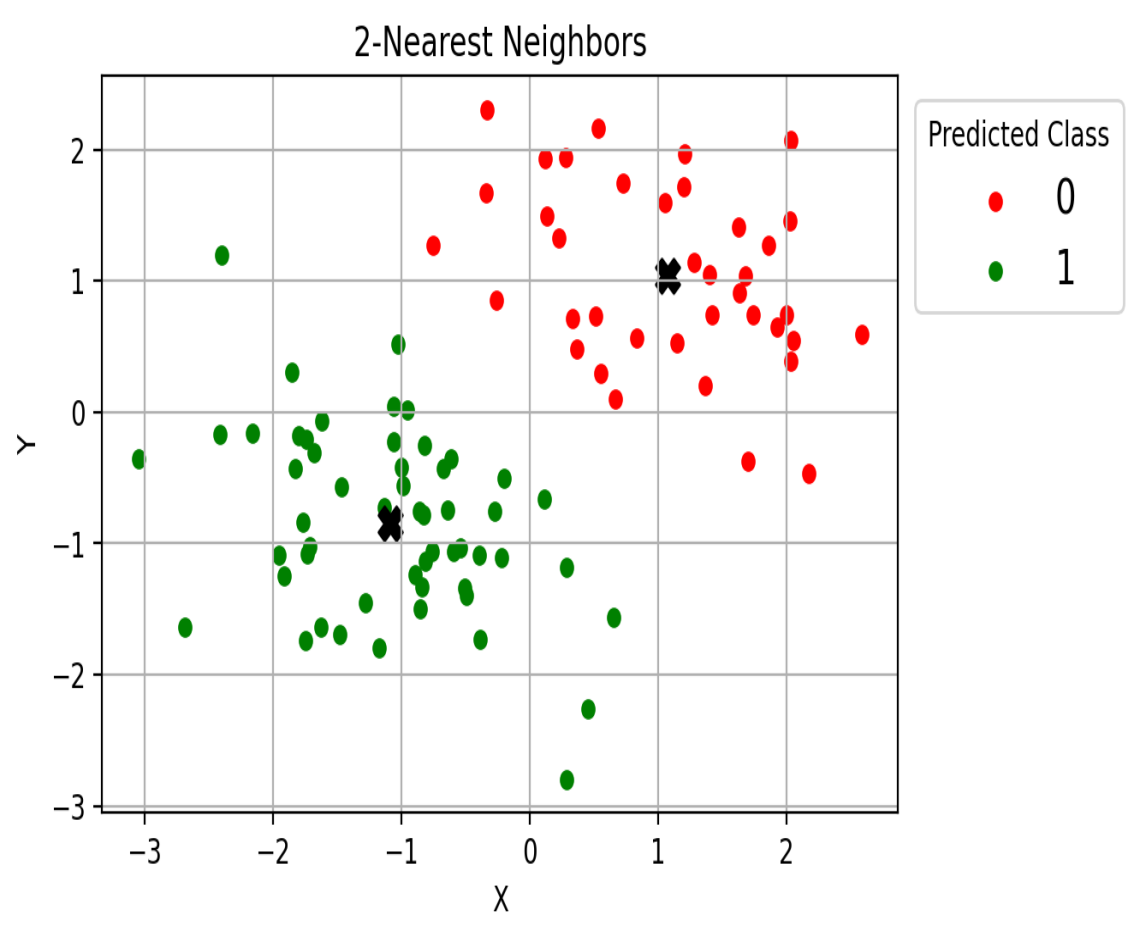
```
xTrain = trainData[['X','Y']]
yTrain = trainData['Group']
kNN_accuracy = numpy.zeros((20,2))
for k in numpy.arange(1,21,1):
    neigh = kNN.KNeighborsClassifier(n_neighbors=k, algorithm = 'brute', metric = 'euclidean')
    nbrs = neigh.fit(xTrain, yTrain)

    # See the classification result
    kNN_accuracy[k-1,0] = k
    kNN_accuracy[k-1,1] = neigh.score(xTrain, yTrain)

neigh = kNN.KNeighborsClassifier(n_neighbors=2, algorithm = 'brute', metric = 'euclidean')
nbrs = neigh.fit(xTrain, yTrain)
trainData['_PredictedClass_'] = nbrs.predict(xTrain)
kNN_Mean = trainData.groupby('_PredictedClass_').mean()
```

Nearest Neighbors

Number of Neighbors	Accuracy
1	100%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%
8	100%
9	100%
10	100%

Number of Neighbors	Accuracy
11	100%
12	100%
13	100%
14	100%
15	100%
16	100%
17	100%
18	100%
19	100%
20	100%

The 1-Neighbor solution is as good as the 20-Neighbors solution!



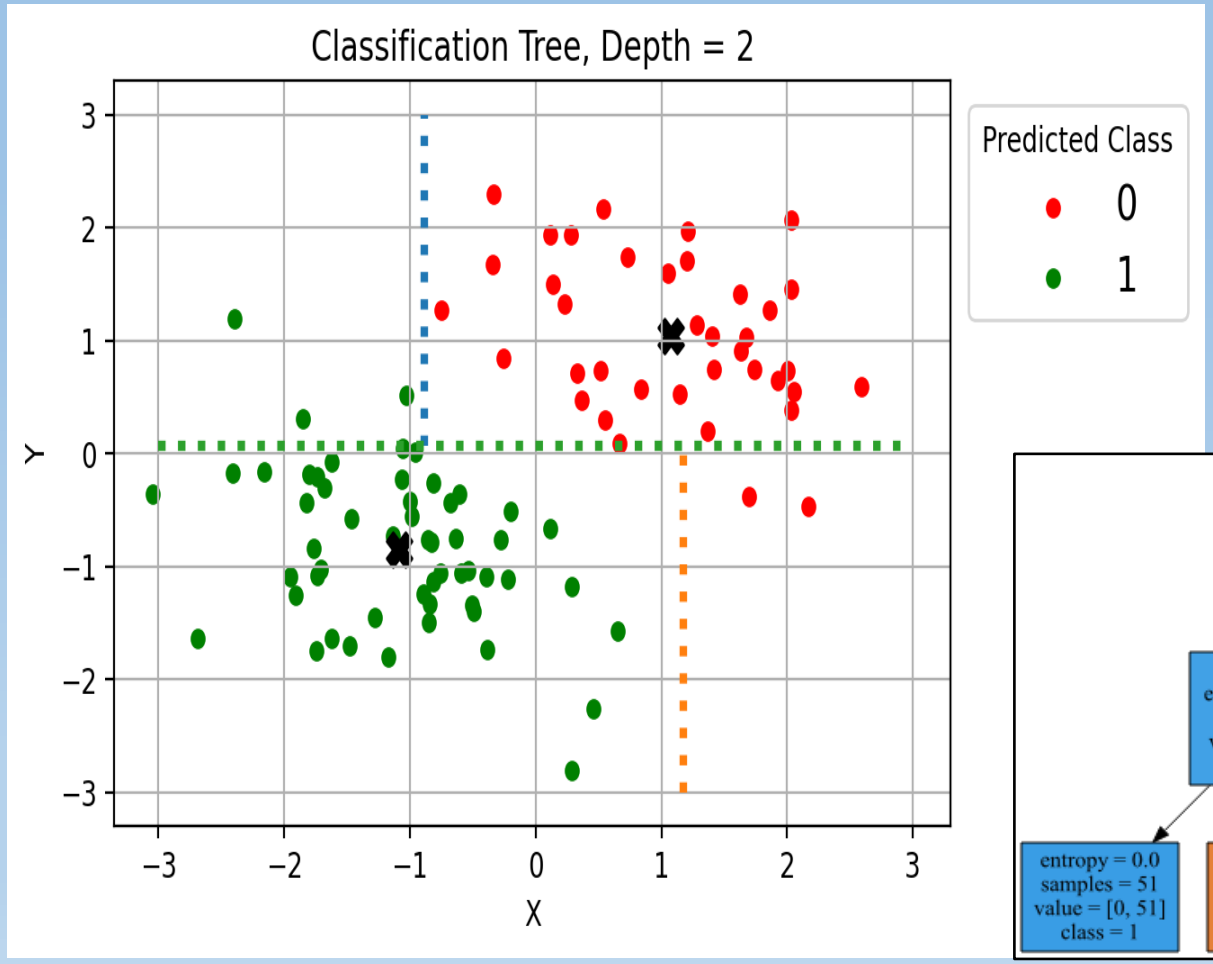
Classification Tree

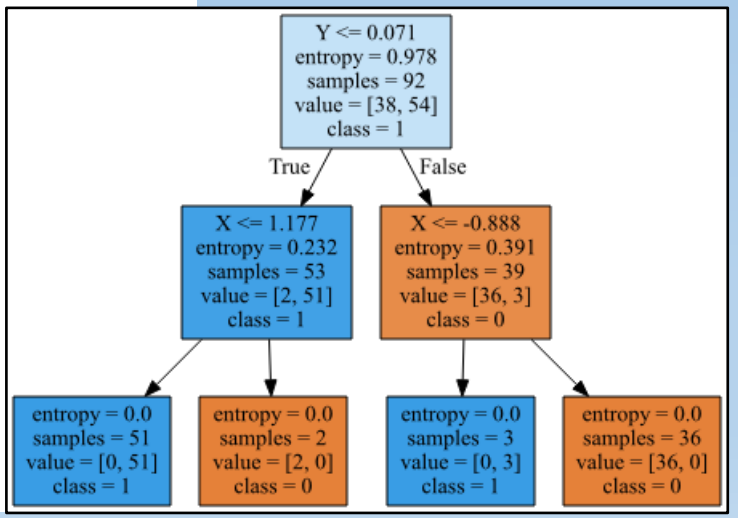
```
xTrain = trainData[['X','Y']]
yTrain = trainData['Group']
tree_accuracy = numpy.zeros((10,2))
for k in numpy.arange(1,11,1):
    objTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=k, random_state=20191106)
    thisFit = objTree.fit(xTrain, yTrain)
    tree_accuracy[k-1,0] = k
    tree_accuracy[k-1,1] = objTree.score(xTrain, yTrain)

objTree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=20191106)
thisFit = objTree.fit(xTrain, yTrain)
y_predProb = thisFit.predict_proba(xTrain)
trainData['_PredictedClass_'] = numpy.where(y_predProb[:,1] >= y_threshold, 1, 0)
tree_Mean = trainData.groupby('_PredictedClass_').mean()
```

Classification Tree

Maximum Depths	Accuracy
1	94.6%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%
8	100%
9	100%
10	100%

Go with the classification tree with depth two!





Logistic Regression Model (BFGS)

```
xTrain = trainData[['X','Y']]
yTrain = trainData['Group']
xTrain = sm.add_constant(xTrain, prepend=True)
logit = sm.MNLogit(yTrain, xTrain)
print("Name of Target Variable:", logit.endog_names)
print("Name(s) of Predictors:", logit.exog_names)

thisFit = logit.fit(method='bfgs', full_output = True, maxiter = 100, tol = 1e-8, retall = True)
print(thisFit.summary())

y_predProb = thisFit.predict(xTrain)
trainData['_PredictedClass_'] = numpy.where(y_predProb[1] >= y_threshold, 1, 0)
```

Logistic Regression Model (BFGS)

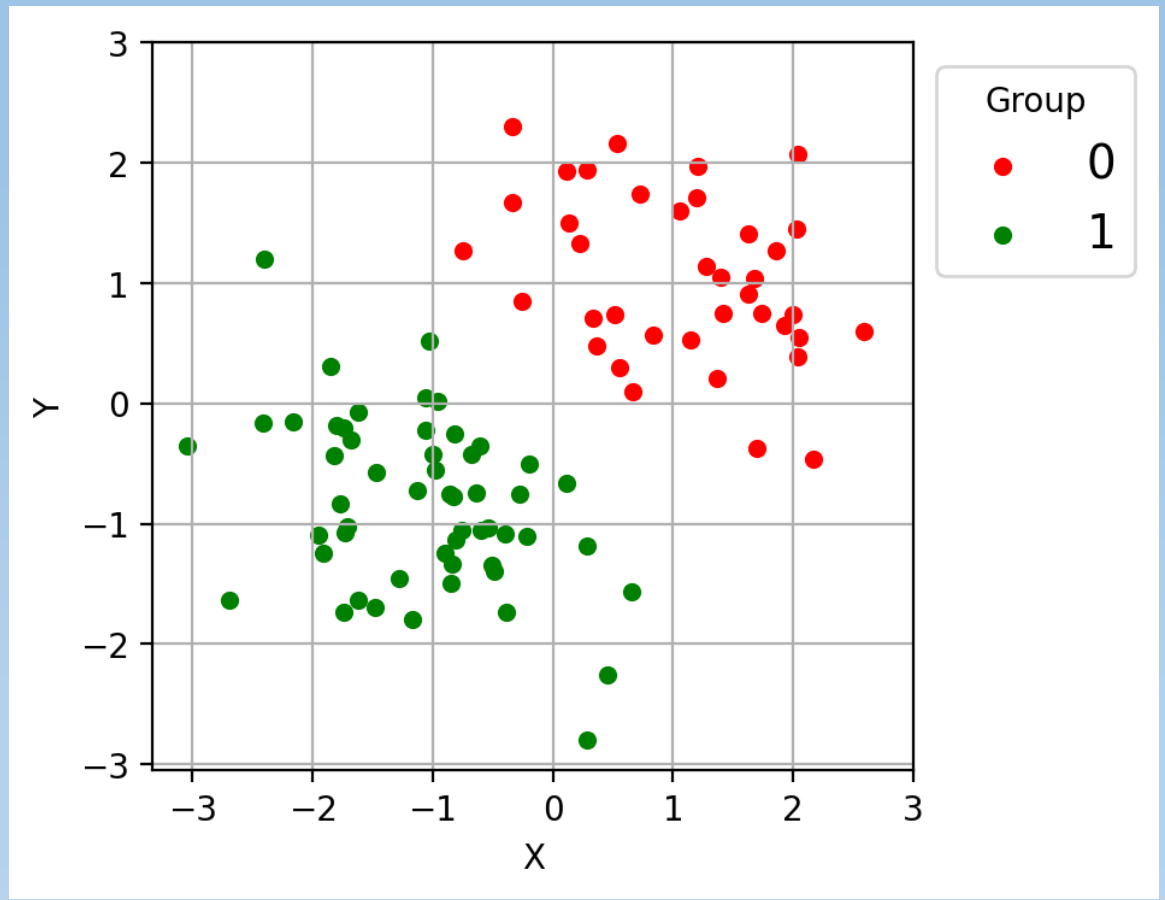
```
==============================================================================
Dep. Variable:                  Group   No. Observations:                   92
Model:                        MNLogit   Df Residuals:                       89
Method:                           MLE   Df Model:                            2
Date:                Thu, 25 Mar 2021   Pseudo R-squ.:                   1.000
Time:                        14:57:21   Log-Likelihood:            -0.0010684
converged:                       True   LL-Null:                      -62.371
Covariance Type:            nonrobust   LLR p-value:                8.185e-28
==============================================================================
   Group=1       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.6428     74.773      0.022      0.982    -144.911     148.196
X            -15.2530     91.423     -0.167      0.867    -194.438     163.932
Y            -16.1540     84.710     -0.191      0.849    -182.182     149.874
==============================================================================
```

Signs of a Complete Separation

1. Almost zero Log-Likelihood
2. Large coefficients and standard errors
3. Newton-Raphson method did not converge

0% Misclassification Rate

Two Groups Obviously



Personas of Algorithms

K-Means

A "*Follow the Leaders*" algorithm

- The leaders are the clusters' centroids
- Assign the observation to the cluster whose centroid is closest to the observation

Nearest Neighbor

A "*Crowd-sourcing*" algorithm

- The crowd consists of the neighbors
- An observation takes the majority opinion of the crowd

Personas of Algorithms

Decision Tree

A "*Divide and Conquer*" algorithm

- Recursively divide the observations into groups of similar target values
- An observation take the majority identity of the group

Logistic Regression

A "*Personality Test*" algorithm

- Each feature individually contributes to the log-odds of a particular target class
- An observation is assigned to the most-likely class

Pros and Cons of the Algorithms

Algorithm	Pros	Cons
K-Means	1. Only need the centroids for prediction	1. The predictions are reliable only if the number of clusters is correct. 2. Need to calculate the distance metric.
Nearest Neighbor	1. Only need to determine the number of neighbors 2. An instance/memory-based learning model	1. The predictions are reliable only if the number of neighbors is correct for every observation. 2. Need to calculate the distance metric. 3. Need to carry the whole training data for prediction.

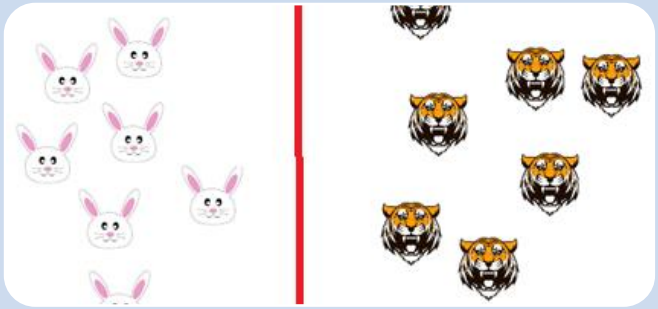
Pros and Cons of the Algorithms

Algorithm	Pros	Cons
Decision Tree	1. Only need to know the splitting criteria. 2. Involve only one feature for splitting.	1. The boundaries of the data segments are always parallel to an axis. 2. Need to know when to stop
Multinomial Logistic	1. Only need to know the parameter estimates (i.e., the weights). 2. Allows us to visualize the impacts of the attributes on the predictions.	1. Correlations among attributes have to be explicitly specified in the model. 2. May suffer from non-convergence and floating-point numerical problems.

Outcomes of Classification Algorithms

Assignment to a Target Class

Likelihood from a Target Class



Set the Boundary Between Target Classes

Set-the-Boundary Type Algorithm

Introduce the Support Vector Machine

Do

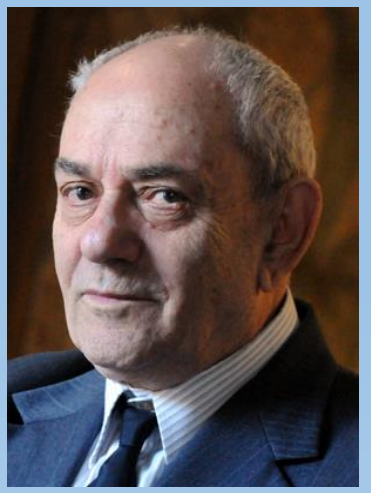
- Work well for future observations
- Need only simple arithmetic calculations for classification

Don't

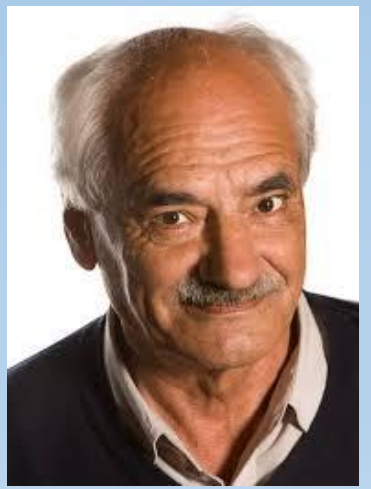
- Need to know the characteristics (e.g., means and standard deviations) of the target class populations
- Need to know the likelihood of belonging to a population
- Need to carry the training data everywhere for prediction
- Need to calculate the distance metric or the logit
- Need to go through iterations to obtain the parameter estimates

History of Support Vector Machines

- In 1963, Vladimir N. Vapnik and Alexey Y. Chervonenkis invented the original Support Vector Machines algorithm at the Institute of Control Sciences of the Russian Academy of Sciences, Moscow, Russia.
- In 1992, Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik improved the original support vector machine which allows for the creation of nonlinear classifiers.



Vapnik (1936-present)



Chervonenkis (1938-2014)

https://www.kdnuggets.com/2016/07/guyon-data-mining-history-svm-support-vector-machines.html
https://link.springer.com/chapter/10.1007%2F978-3-642-41136-6_3

Overview of Support Vector Machines

A machine is a device or an algorithm that performs a particular task with a definite goal

Can successfully solve many classification problems

A classifier defined by a hyperplane

A large-margin linear classifier

Matrix Algebra: Column Vector

- A **column vector** is an array of numbers that are listed as a column.
- Denote the column vector in boldface

- The **dimension** of a column vector is written as $k \times 1$ where $k \geq 1$ is the number of elements in the column vector.

Example

- $\mathbf{w} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 7 \end{bmatrix}$
- Dimension of $\mathbf{w}$ is 4×1

Matrix Algebra: Row Vector

- A **row vector** is an array of numbers that are listed as a row.
- Denote the row vector in boldface

- The **dimension** of a row vector is written as $1 \times k$ where $k \geq 1$ is the number of elements in the row vector.

Example

- $\mathbf{w} = [-1 \quad -2 \quad -3]$
- Dimension of $\mathbf{w}$ is 1×3

Matrix Algebra: Transpose Operator

- The **Transpose** operator transforms a column vector to a row vector, and vice versa
- Denote by a superscript *t*

- In general, the **Transpose** operator interchanges the row and the column dimensions

Example

- $\mathbf{w} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$
- $\mathbf{w}^t = [1 \quad 2 \quad 3]$

Matrix Algebra: Inner Product of Two Vectors

- Let $\mathbf{w}$ and $\mathbf{v}$ be two column vector
- Both vectors have the same dimension $k \times 1$

- $\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_k \end{pmatrix}$

- The inner product of $\mathbf{w}$ and $\mathbf{v}$ is denoted as $\langle \mathbf{w}, \mathbf{v} \rangle$ or $\mathbf{w}^t \mathbf{v}$, is the sum of the products of the elements
- $\langle \mathbf{w}, \mathbf{v} \rangle = \sum_{i=1}^{k} w_i v_i$

Matrix Algebra: Norm of a Vector

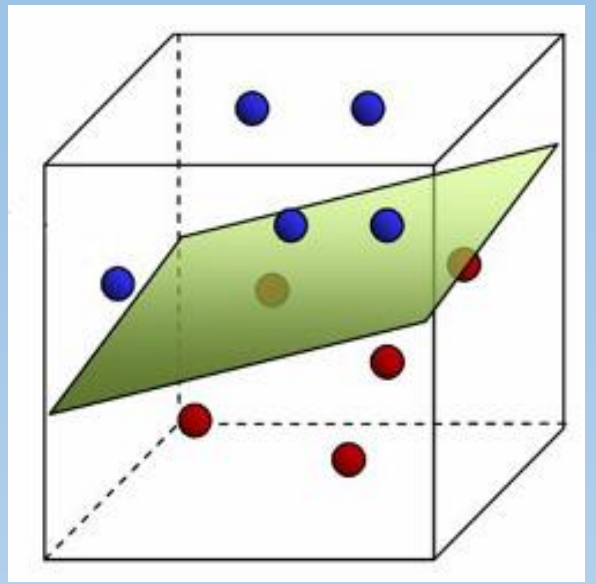
- Let $\mathbf{w}$ be a column vector of dimension $k \times 1$
- $\mathbf{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_k \end{pmatrix}$

- The transpose $\mathbf{w}^t = (w_1, \ldots, w_k)$

- The norm of a vector, denoted as $\|\mathbf{w}\|$, is the square root of the sum of squared of the elements
- $\|\mathbf{w}\| = \sqrt{\sum_{i=1}^{k} w_i^2} = \sqrt{\langle \mathbf{w}, \mathbf{w} \rangle}$

What is Hyperplane?

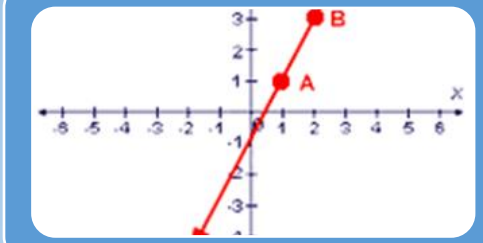
- Let $\mathbf{w}^t = (w_1, \dots, w_p)$ be a given vector of scalars and at least one of them is not zero.
- Let $\mathbf{x}^t = (x_1, \dots, x_p)$ be a general vector in the $\mathbb{R}^p$ the *p*-dimensional space of real numbers.
- The set $\{\mathbf{x} \in \mathbb{R}^p : \mathbf{w}^t\mathbf{x} = c\}$ for a constant c is a hyperplane. In other words, a hyperplane is a subspace of $\mathbb{R}^p$, the p^{th} dimensional number space.



Examples of a Hyperplane

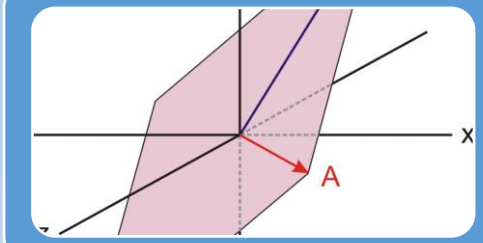
One-dimensional, $p = 1$

- It is a point $w_1 x_1 = c$, if $w_1 \neq 0$ then $x_1 = c/w_1$
- If $w_1 = 0$, then the hyperplane is undefined



Two-dimensional, $p = 2$

- It is a line $w_1 x_1 + w_2 x_2 = c$ when $w_1 \neq 0$ and $w_2 \neq 0$
- If $w_1 = 0$ or $w_2 = 0$ (but not both), then this hyperplane reduces to a point



Three-dimensional, $p = 3$

- It is a plane $w_1 x_1 + w_2 x_2 + w_3 x_3 = c$ when $w_1 \neq 0$, $w_2 \neq 0$, and $w_2 \neq 0$
- Reduce to a hyperplane in a lower dimension if some $w_i = 0$

What are the Support Vectors?

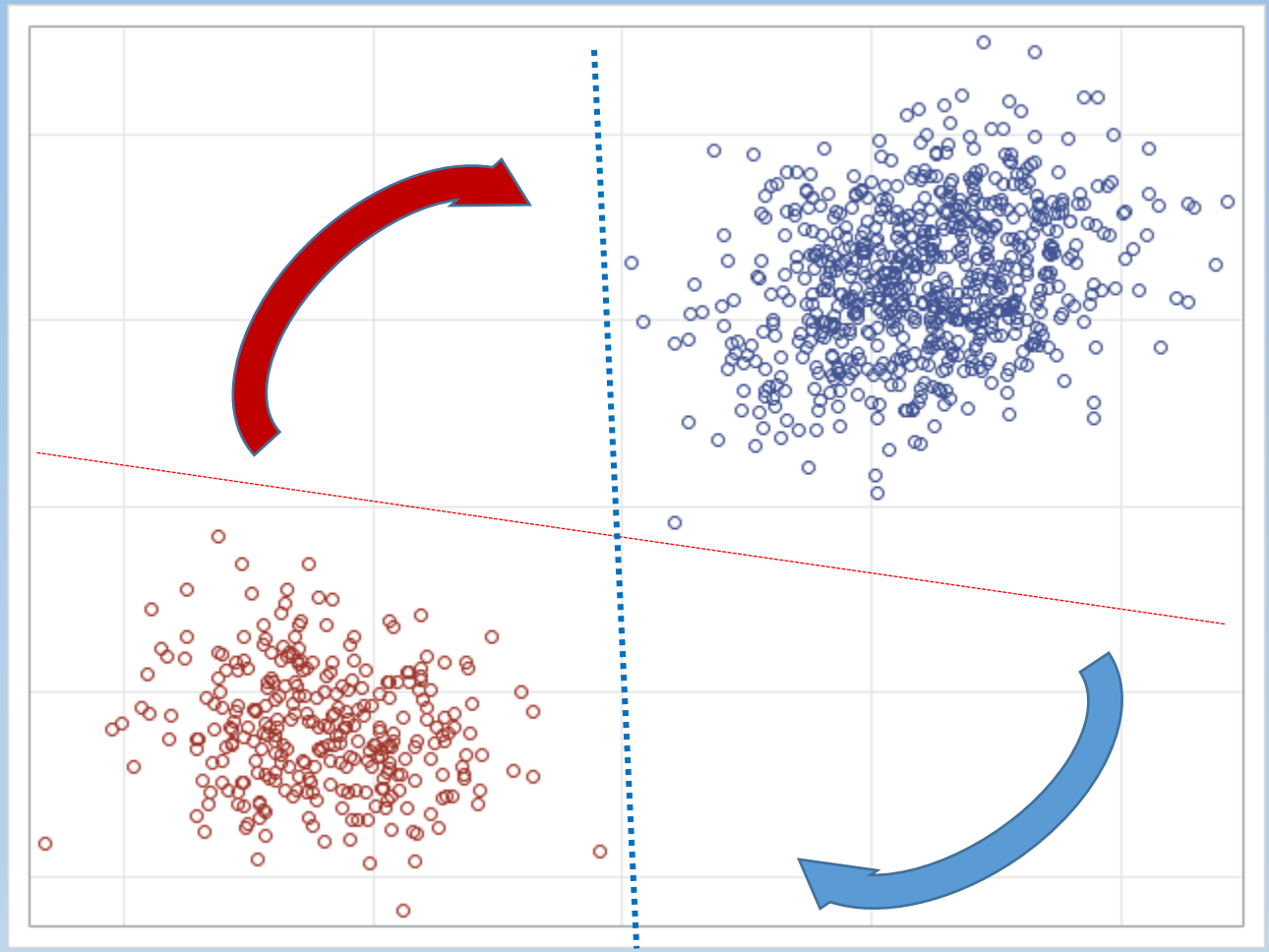
- Given a hyperplane, the support vectors are observations that are *nearest* to the hyperplane.
- Change the hyperplane, get a different set of support vectors
- They are the observations which have a high tendency to be misclassified.
- They have direct and often the highest influence on the weights (i.e., $w_1, \ldots, w_p$) of the hyperplane.

Perceptron: A Linear Separator

- The input to the hidden node is the weighted sum $\sum_{j=1}^{p} w_j x_j$
- If this weighted sum is greater than a *threshold value* c, then the output from the hidden node is 1. Otherwise, the output value is 0.
- The parameters of the perceptron are the weights $w_1, \ldots, w_p$ and the threshold value c.

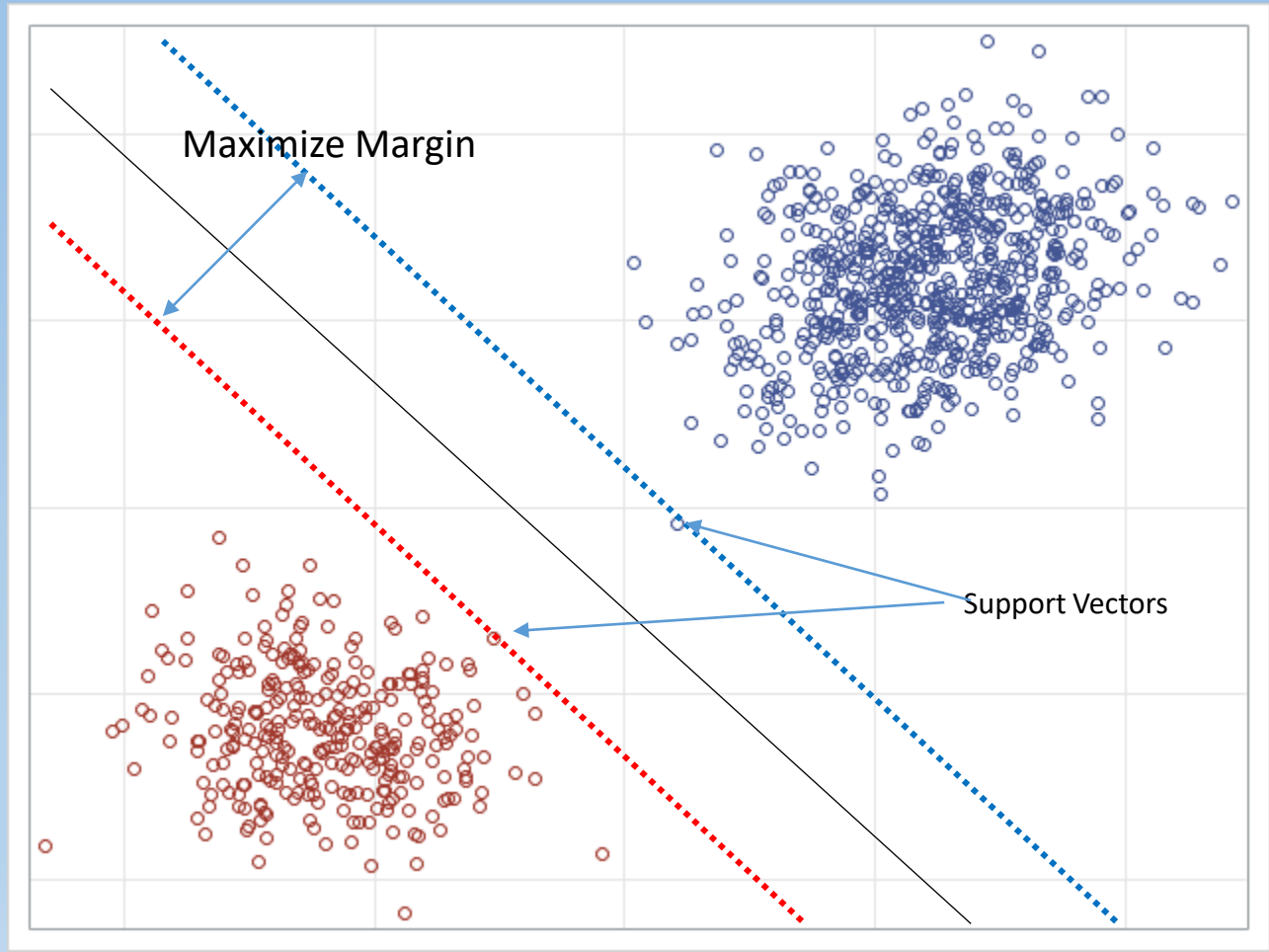
Which Linear Separator?

- In the graph on the right, any linear separator $w_1 x_1 + w_2 x_2 = c$ that lies between the red dashed line and the blue dashed line can separate the two obvious groups of observations.
- In other words, there are infinite choices of w_1, w_2, and c.



The Optimal Linear Separator

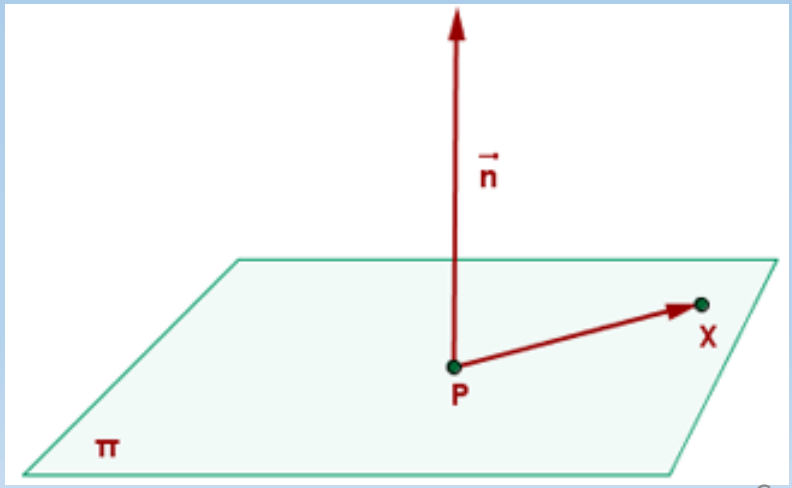
- SVM finds an optimal choice of w_1, w_2, and c such that it maximizes the margin between the hyperplane and the support vectors.
- As you can see, the hyperplane is fully specified by the support vectors and all other training observations are ignorable.



Distance from a Point to a Hyperplane

(Advanced Material)

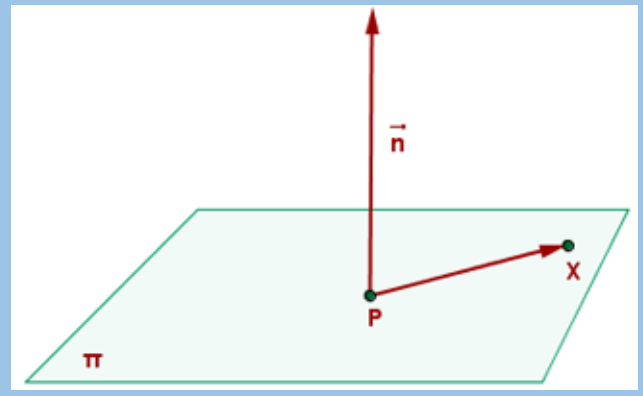
- Suppose the support vector is $\mathbf{x}_i$.
- The hyperplane is originally represented as $\mathbf{w}^t\mathbf{x} = c$.
  - By letting $w_0 = -c$, we can re-write the representation as $w_0 + \mathbf{w}^t\mathbf{x} = 0$.
- The shortest distance from a point P to a hyperplane is along the normal vector $\vec{\mathbf{n}}$ which is perpendicular to the plane.



Source: https://www.ditutor.com/space/normal_vector.html

Normal Vector of a Hyperplane

(Advanced Material)

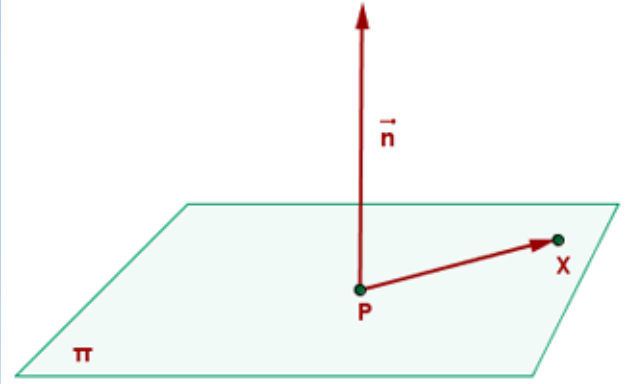


- Let us consider this particular hyperplane $\mathbf{w}^t\mathbf{x} = 0$.
- Obviously, the origin $\mathbf{P}^t = (0, \dots, 0)$ is on this hyperplane because of this identity $\mathbf{w}^t\mathbf{P} = 0$.
- The vector $\overrightarrow{\mathbf{PX}} = \vec{\mathbf{x}} - \vec{\mathbf{P}} = \vec{\mathbf{x}}$ is on the hyperplane
  - The overhead arrow indicates the direction.
- If $\vec{\mathbf{n}}$ is the normal vector to the hyperplane $\mathbf{w}^t\mathbf{x} = 0$, then the inner product between $\vec{\mathbf{n}}$ and $\overrightarrow{\mathbf{PX}}$ has to be zero.
- Mathematically speaking, $\langle \vec{\mathbf{n}}, \overrightarrow{\mathbf{PX}} \rangle = \langle \vec{\mathbf{n}}, \vec{\mathbf{x}} \rangle = \mathbf{n}^t\mathbf{x} = 0$
- Therefore, $\vec{\mathbf{n}} = \vec{\mathbf{w}}$. In other words, $\mathbf{n}^t = \mathbf{w}^t = (w_1, \dots, w_p)$.

Normal Vector of a Hyperplane

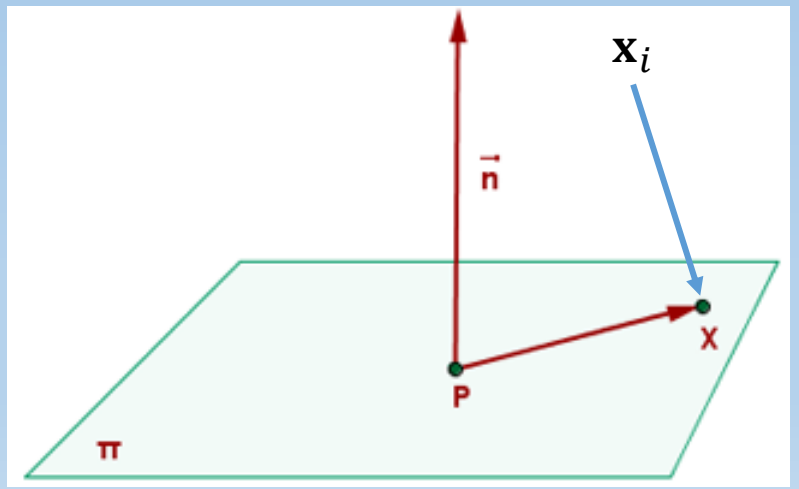
(Advanced Material)

- If $\vec{\mathbf{n}}$ is the normal vector to the hyperplane $\mathbf{w}^t\mathbf{x} = 0$, then $\vec{\mathbf{n}}$ is the also the normal vector to all hyperplanes which are parallel to this hyperplane.
- Since a hyperplane $w_0 + \mathbf{w}^t\mathbf{x} = 0$ for any w_0 is parallel to the hyperplane $\mathbf{w}^t\mathbf{x} = 0$, we can conclude that $\vec{\mathbf{n}} = \vec{\mathbf{w}}$ is also the normal vector to the hyperplane $w_0 + \mathbf{w}^t\mathbf{x} = 0$.



Distance from the Support Vector to a Hyperplane

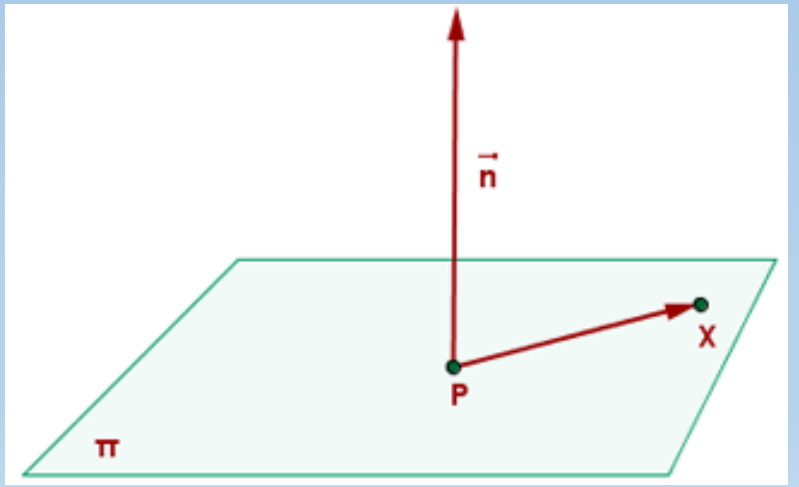
- Suppose the support vector is $\mathbf{x}_i$.
- The vector from the support vector to a point $\mathbf{x}$ on the hyperplane $w_0 + \mathbf{w}^t\mathbf{x} = 0$ is $\overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}$.
- Projecting $\overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}$ onto the normal vector $\vec{\mathbf{n}}$ can give the distance D from the support vector to the hyperplane.
- Theorem: $\langle \overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}, \vec{\mathbf{n}} \rangle = \|\overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}\| \|\vec{\mathbf{n}}\| \cos(\theta)$ where θ is the angle between this vector $\overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}$ and the normal vector $\vec{\mathbf{n}}$.
- $\|\overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}\| \cos(\theta)$ is the distance D.



Distance from the Support Vector to a Hyperplane

- There $D = \frac{|\langle \overrightarrow{\mathbf{x}_i} - \vec{\mathbf{x}}, \vec{\mathbf{n}} \rangle|}{\|\vec{\mathbf{n}}\|} = \frac{|\mathbf{n}^t(\mathbf{x}_i - \mathbf{x})|}{\sqrt{\mathbf{n}^t\mathbf{n}}} = \frac{|\mathbf{w}^t(\mathbf{x}_i - \mathbf{x})|}{\sqrt{\mathbf{w}^t\mathbf{w}}}$
- Since $\mathbf{w}^t\mathbf{x} = -w_0$, $\mathbf{w}^t(\mathbf{x}_i - \mathbf{x}) = \mathbf{w}^t\mathbf{x}_i - (-w_0) = w_0 + \mathbf{w}^t\mathbf{x}_i$.
- Thus, the distance from the support vector to the hyperplane is

$$D = \frac{|w_0 + \mathbf{w}^t\mathbf{x}_i|}{\sqrt{\mathbf{w}^t\mathbf{w}}}$$



Source: https://www.ditutor.com/space/normal_vector.html

From Perceptron to Support Vector Machines

- A binary classification (i.e., the label or the target variable has only two distinct values) can be considered as the task of separating *p*-dimensional observations in a $\mathbb{R}^p$ space.
- Use this hyperplane $w_0 + \sum_{j=1}^{p} w_j x_j = 0$ as the "knife" that divides the observations in a $\mathbb{R}^p$ space into two subsets.
- Instead of values 0 or 1, the output from the classifier is -1 or 1.
  - Subset 0: $w_0 + \sum_{j=1}^{p} w_j x_j \leq -1$, output $y = -1$.
  - Subset 1: $w_0 + \sum_{j=1}^{p} w_j x_j \geq +1$, output $y = +1$.
- Collectively, $y(w_0 + \mathbf{w}^t \mathbf{x}) \geq +1$

Maximizing the Margin

- Since the support vectors must satisfy $y(w_0 + \mathbf{w}^t\mathbf{x}) \geq +1$, we can rewrite $|w_0 + \mathbf{w}^t\mathbf{x}_i| = y(w_0 + \mathbf{w}^t\mathbf{x}_i)$. It is because
  - Subset 0: $w_0 + \mathbf{w}^t\mathbf{x}_i \leq -1$, the output is $y = -1$ and $y(w_0 + \mathbf{w}^t\mathbf{x}_i) = -1 \times (w_0 + \mathbf{w}^t\mathbf{x}_i) = |w_0 + \mathbf{w}^t\mathbf{x}_i|$.
  - Subset 1: $w_0 + \mathbf{w}^t\mathbf{x}_i \geq +1$, the output is $y = +1$ and $y(w_0 + \mathbf{w}^t\mathbf{x}_i) = +1 \times (w_0 + \mathbf{w}^t\mathbf{x}_i) = |w_0 + \mathbf{w}^t\mathbf{x}_i|$.
- Our goal is to find the largest $\rho > 1$ such that $\frac{y(w_0+\mathbf{w}^t\mathbf{x})}{\sqrt{\mathbf{w}^t\mathbf{w}}} \geq \rho$ for all observations $\mathbf{x}$ in the training data. In other words, we would like to maximize ρ with respect to w_0 and $\mathbf{w}$.

Minimizing the Norm of the Weights

- However, we can rescale the weights either up or down, and we still can get the same ρ.
  - For example, $\frac{y(2w_0+2\mathbf{w}^t\mathbf{x})}{\sqrt{(2\mathbf{w})^t(2\mathbf{w})}} = \frac{y(w_0+\mathbf{w}^t\mathbf{x})}{\sqrt{\mathbf{w}^t\mathbf{w}}} = \frac{y(0.5w_0+0.5\mathbf{w}^t\mathbf{x})}{\sqrt{(0.5\mathbf{w})^t(0.5\mathbf{w})}}$
- Therefore, we are only interested in the weights $\mathbf{w}$ such that $\rho\|\mathbf{w}\| = \rho\sqrt{\mathbf{w}^t\mathbf{w}} = 1$.
- This transforms our original optimization task of maximizing ρ to a new optimization task of minimizing $\|\mathbf{w}\|$.
  - In practice, we will minimize $\|\mathbf{w}\|^2 = \mathbf{w}^t\mathbf{w}$ indeed for ease of manipulation arithmetically.

The Unconstrained Minimization Problem

- The new optimization task is to minimize $\mathbf{w}^t\mathbf{w}$ subject to $y(w_0 + \mathbf{w}^t\mathbf{x}) \geq +1$ for all observation $\mathbf{x}$.
- Let $(\mathbf{x}_i, y_i), i = 1, \ldots, N$ denotes the pairs of the features and the target variable.
- This constrained optimization task can be transformed into an unconstrained optimization task by using the Lagrange multipliers $\alpha_i \geq 0, i = 1, \ldots, N$ as follows:
- Minimize $L = \frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{N} \alpha_i(y_i(w_0 + \mathbf{w}^t\mathbf{x}_i) - 1)$ with respect to w_0, $\mathbf{w}$, and $\alpha_1, \ldots, \alpha_N$.

The Unconstrained Minimization Problem

- Minimize $L = \frac{1}{2}\mathbf{w}^t\mathbf{w} - \sum_{i=1}^{N} \alpha_i(y_i(w_0 + \mathbf{w}^t\mathbf{x}_i) - 1)$ with respect to w_0, $\mathbf{w}$, and $\alpha_1, \ldots, \alpha_N$.
- Expand the objective function as

$$L = \frac{1}{2}\mathbf{w}^t\mathbf{w} - w_0\sum_{i=1}^{N}\alpha_i y_i - \mathbf{w}^t\sum_{i=1}^{N}\alpha_i y_i\mathbf{x}_i + \sum_{i=1}^{N}\alpha_i$$

- Set the partial derivatives to zero
  - $\frac{\partial L}{\partial w_0} = -\sum_{i=1}^{N}\alpha_i y_i = 0$ or $\sum_{i=1}^{N}\alpha_i y_i = 0$
  - $\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i\mathbf{x}_i = 0$ or $\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i\mathbf{x}_i$

The Unconstrained Minimization Problem

- Applying $\sum_{i=1}^{N} \alpha_i y_i = 0$ and $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$ into the objective function, we get

$$L = \frac{1}{2}\mathbf{w}^t\mathbf{w} - 0 - \mathbf{w}^t\mathbf{w} + \sum_{i=1}^{N} \alpha_i = -\frac{1}{2}\mathbf{w}^t\mathbf{w} + \sum_{i=1}^{N} \alpha_i$$

- Expressing the objective function in term of the Lagrange multipliers

$$L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^t \mathbf{x}_j$$

The Unconstrained Minimization Problem

(End of Advanced Material)

- The final minimization problem is:
  - The objective function $L = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^t \mathbf{x}_j$
  - With respect to $\alpha_i \geq 0, i = 1, \ldots, N$
  - Subject to $\sum_{i=1}^{N} \alpha_i y_i = 0$.
- This can be solved using quadratic programming method.
  - This method minimizes or maximizes a quadratic function of variables subject to linear constraints on these variables.
  - The LIBLINEAR library (https://www.csie.ntu.edu.tw/~cjlin/liblinear/) that is used by sklearn.svm.SVC implemented the conjugate gradient algorithm.

sklearn.svm.SVC() Function

- **kernel = 'linear'**
- **max_iter = -1** to specify no limit on the number of iterations
- The Intercept w_0 can be retrieved from the **intercept_** object
- The array of coefficients $\mathbf{w}$ can be retrieved from the **coef_** object
- The hyperplane is constructed from the equation $w_0 + \mathbf{w}^t\mathbf{x} = \mathbf{0}$

From General Form to Slope-Intercept Form

- When $p = 2$, we can visualize the boundary
- The 2-dimensional number space $\mathbb{R}^2 = \{(x_1, x_2): -\infty < x_1, x_2 < \infty\}$
- General form: $w_0 + w_1 x_1 + w_2 x_2 = 0$
- If $w_2 \neq 0$, then $w_2 x_2 = -w_0 - w_1 x_1 \Leftrightarrow x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} x_1$
- Slope-Intercept form: $x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} x_1$
- Slope is $-\frac{w_1}{w_2}$ and Intercept is $-\frac{w_0}{w_2}$

Support Vector Machine

```
xTrain = trainData[['X','Y']]
yTrain = trainData['Group']

svm_Model = svm.SVC(kernel = 'linear', random_state = 20191106, max_iter = -1)
thisFit = svm_Model.fit(xTrain, yTrain)
y_predictClass = thisFit.predict(xTrain)
print('Mean Accuracy = ', metrics.accuracy_score(yTrain, y_predictClass))
trainData['_PredictedClass_'] = y_predictClass
svm_Mean = trainData.groupby('_PredictedClass_').mean()

# get the separating hyperplane
w = thisFit.coef_[0]
a = -w[0] / w[1]
xx = numpy.linspace(-3, 3)
yy = a * xx - (thisFit.intercept_[0]) / w[1]

# plot the parallels to the separating hyperplane that pass through the
# support vectors
b = thisFit.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])

b = thisFit.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

Week 12 Identify MVN.py

Support Vector Machine

- The Intercept w_0 = 0.0063753244948
- The array of coefficients
  $\mathbf{w}$ = [-1.1963594311056, -1.4961953600765]
- The hyperplane is represented by this general form equation:
  0.0063753 - 1.1963594 * x - 1.4961954 * y = 0
- The slope-intercept form: y = 0.0042610 - 0.7996011 * x

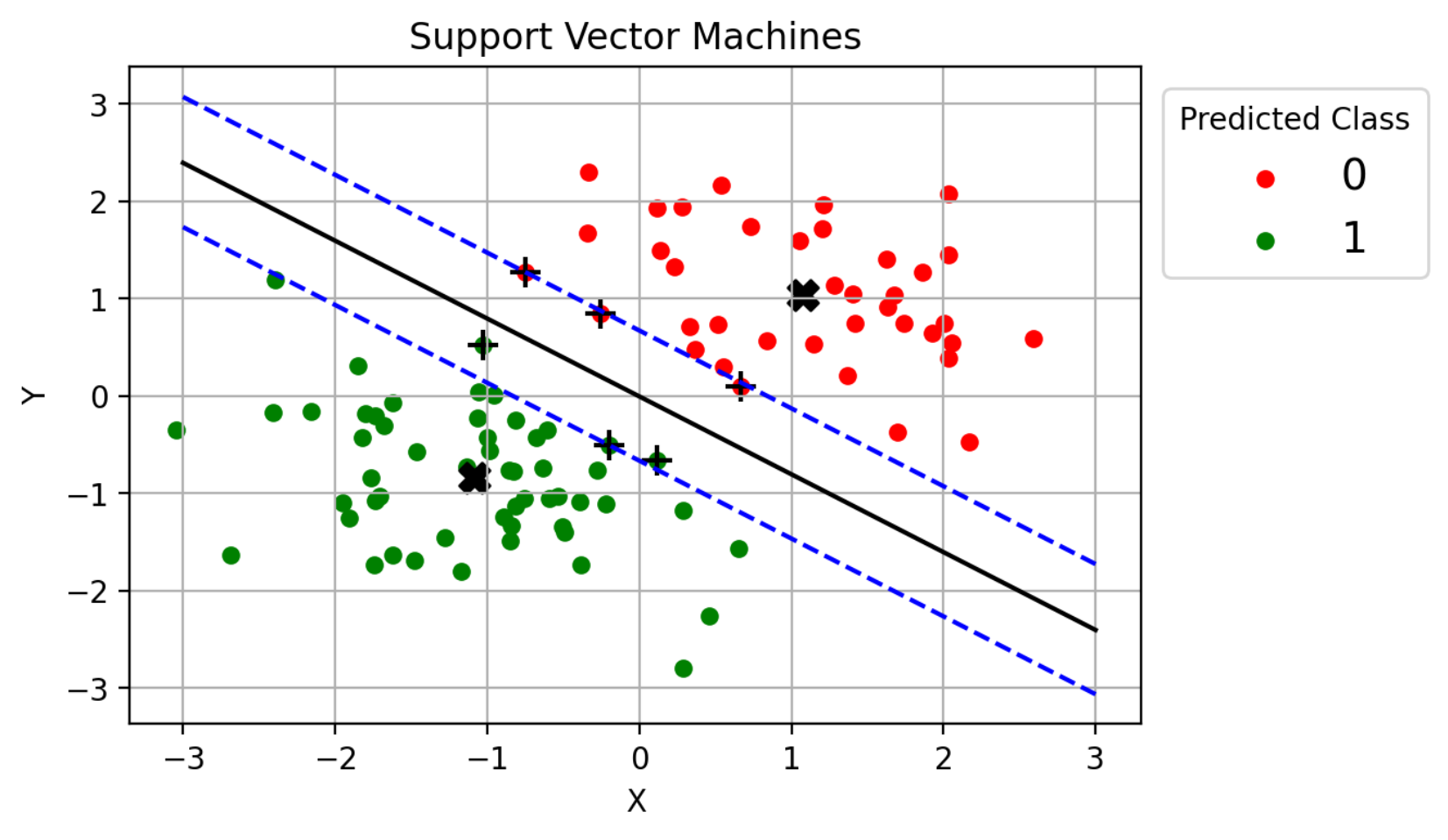
Zero misclassification rate!

Week 10 Identify MVN.py

Support Vectors

x	y	0.0063753 - 1.1963594 * x - 1.4961954 * y
-0.7489152474284	1.2714563756114	-0.999999986
-0.2554720190218	0.8506655660570	-0.960750189
0.6666807932735	0.0964977293203	-0.935593985
-1.0270341093332	0.5210185662266	0.455531706
0.1146322116875	-0.6592369607678	0.855581279
-0.1985435673937	-0.5053452368473	0.999999992

Support Vectors Marked by "+"



sklearn.svm.LinearSVC() Function

- **dual = False** if n_samples > n_features
- **random_state = 0** to disable random number generation
- The Intercept w_0 can be retrieved from the **intercept_** object
- The array of coefficients $\mathbf{w}$ can be retrieved from the **coef_** object
- The hyperplane is constructed from the equation $w_0 + \mathbf{w}^t\mathbf{x} = \mathbf{0}$

Feature Selection

- Enter one feature sequentially
- Enter the feature that yields the lowest misclassification rate
- Stop if the misclassification rate does not decrease

- HMEQ data
- Target is BAD
- Categories: 0 and 1

HMEQ Data: Step 1

Week 10 HMEQ SVM.py

- Included Features is empty
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG, NINQ, YOJ

```
hmeq = pandas.read_csv('C:\\IIT\\Machine Learning\\Data\\HMEQ.csv',delimiter=',')

y_name = 'BAD'

# Set dual = False because n_samples > n_features

accuracyResult = pandas.DataFrame()
includeVar = []
X_name = ['CLAGE','CLNO','DELINQ','DEROG','NINQ','YOJ']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]], ignore_index = True)
```

HMEQ Data: Step 1

- Included Features is empty
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG, NINQ, YOJ

Feature	N Samples	Misclassification
YOJ	5445	0.79357
NINQ	5450	0.79431
CLNO	5738	0.80202
DEROG	5252	0.80236
CLAGE	5652	0.80343
DELINQ	5380	0.81152

HMEQ Data: Step 2

- Included Feature: YOJ
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG, NINQ

```
accuracyResult = pandas.DataFrame()
includeVar = ['YOJ']
X_name = ['CLAGE','CLNO','DELINQ','DEROG','NINQ']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]],
                                           ignore_index = True)
```

HMEQ Data: Step 2

- Included Feature: YOJ
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG, NINQ

Feature	N Samples	Misclassification
YOJ, NINQ	5127	0.79013
YOJ, CLNO	5340	0.79719
YOJ, CLAGE	5271	0.79928
YOJ, DEROG	4914	0.80362
YOJ, DELINQ	5059	0.80945

HMEQ Data: Step 3

- Included Feature: YOJ, NINQ
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG

```
accuracyResult = pandas.DataFrame()
includeVar = ['YOJ', 'NINQ']
X_name = ['CLAGE','CLNO','DELINQ','DEROG']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]],
                                           ignore_index = True)
```

HMEQ Data: Step 3

- Included Feature: YOJ, NINQ
- Candidate Features: CLAGE, CLNO, DELINQ, DEROG

Feature	N Samples	Misclassification
YOJ, NINQ, CLNO	5127	0.79091
YOJ, NINQ, CLAGE	5058	0.79439
YOJ, NINQ, DEROG	4837	0.80132
YOJ, NINQ, DELINQ	4983	0.81617

HMEQ Data: Step 4

- Included Feature: YOJ, NINQ, CLNO
- Candidate Features: CLAGE, DELINQ, DEROG

```
accuracyResult = pandas.DataFrame()
includeVar = ['YOJ', 'NINQ', 'CLNO']
X_name = ['CLAGE','DELINQ','DEROG']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]],
                                           ignore_index = True)
```

HMEQ Data: Step 4

- Included Feature: YOJ, NINQ, CLNO
- Candidate Features: CLAGE, DELINQ, DEROG

Feature	N Samples	Misclassification
YOJ, NINQ, CLNO, CLAGE	5058	0.79439
YOJ, NINQ, CLNO, DELINQ	4983	0.81698
YOJ, NINQ, CLNO, DEROG	4837	0.80091

HMEQ Data: Step 5

- Included Feature: YOJ, NINQ, CLNO, CLAGE
- Candidate Features: DELINQ, DEROG

```
accuracyResult = pandas.DataFrame()
includeVar = ['YOJ', 'NINQ', 'CLNO', 'CLAGE']
X_name = ['DELINQ','DEROG']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]],
                                           ignore_index = True)
```

HMEQ Data: Step 5

- Included Feature: YOJ, NINQ, CLNO, CLAGE
- Candidate Features: DELINQ, DEROG

Enter DEROG

Feature	N Samples	Misclassification
YOJ, NINQ, CLNO, CLAGE, DEROG	4768	0.80998
YOJ, NINQ, CLNO, CLAGE, DELINQ	4914	0.82011

HMEQ Data: Step 6

- Included Feature: YOJ, NINQ, CLNO, CLAGE, DEROG
- Candidate Features: DELINQ

```
accuracyResult = pandas.DataFrame()
includeVar = ['YOJ', 'NINQ', 'CLNO', 'CLAGE', 'DEROG']
X_name = ['DELINQ']

for ivar in X_name:
    inputData = hmeq[includeVar + [y_name, ivar]].dropna()
    X = inputData[includeVar + [ivar]]
    y = inputData[y_name].astype('category')
    svm_Model = svm.LinearSVC(verbose = 1, dual = False, random_state = None, max_iter = 10000)
    thisFit = svm_Model.fit(X, y)
    y_predictClass = thisFit.predict(X)
    y_predictAccuracy = metrics.accuracy_score(y, y_predictClass)
    accuracyResult = accuracyResult.append([[includeVar + [ivar], inputData.shape[0], y_predictAccuracy]],
                                           ignore_index = True)
```

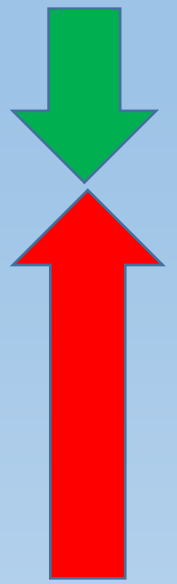
HMEQ Data: Step 6

- Included Feature: YOJ, NINQ, CLNO, CLAGE, DEROG
- Candidate Features: DELINQ

Feature	N Samples	Misclassification
YOJ, NINQ, CLNO, CLAGE, DEROG, DELINQ	4748	0.83151

HMEQ Data: Step Summary

Step	Feature	N Samples	Misclassification
1	YOJ	5445	0.79357
2	YOJ, NINQ	5127	0.79013
3	YOJ, NINQ, CLNO	5127	0.79091
4	YOJ, NINQ, CLNO, CLAGE	5058	0.79439
5	YOJ, NINQ, CLNO, CLAGE, DEROG	4768	0.80998
6	YOJ, NINQ, CLNO, CLAGE, DEROG, DELINQ	4748	0.83151



- Consider to stop at Step #2 or Step #3
- My choice of the optimal feature set: {YOJ, NINQ}

The Non-separable Case

- There is no hyperplane to separate the two populations.
- We define the slack variables $\xi_i, \geq 0\ i = 1, \ldots, N$ which represent the deviation from the margin.
- There are two types of deviations:
  1. Lie on the wrong side of the hyperplane and is misclassified
  2. Lie on the correct side but inside the margin area, i.e., not sufficiently away from the hyperplane.

The Non-separable Case

- Instead of strictly requiring $y(w_0 + \mathbf{w}^t\mathbf{x}) \geq +1$, we relax it as $y(w_0 + \mathbf{w}^t\mathbf{x}) \geq 1 - \xi$.
- Three scenarios:
  - $\xi = 0$. The observation $\mathbf{x}$ is on the correct side and lies outside the margin.
  - $0 < \xi < 1$. The observation $\mathbf{x}$ is on the correct side and lies inside the margin. This observation is non-separable.
  - $\xi > 1$. The observation $\mathbf{x}$ is on the wrong side. This observation is misclassified.
- Define the Soft Error as the $\sum_{i=1}^{N} \xi_i$. We add this Soft Error as a penalty term to the objective function *L*.

The Non-separable Case

- Define the Hinge Loss as the estimated Soft Error.
  - It is zero if $y(w_0 + \mathbf{w}^t\mathbf{x}) \geq +1$ (i.e., $\xi = 0$)
  - It is $1 - y(w_0 + \mathbf{w}^t\mathbf{x})$ otherwise (i.e., $0 < \xi < 1$ or $\xi > 1$)
- Specifying the Hinge Loss will make the margin narrower than the optimal, and to re-position the hyperplane in the margin away from the support vectors.

Multi-Class Target Variable

If the target variable has more than two categories, the classification can be done in

1. OVO: One class versus another class. One SVM classifier for each pair of classes.
2. OVR: One class versus all the other classes. One SVM classifier for each pair of a class versus the otherwise.

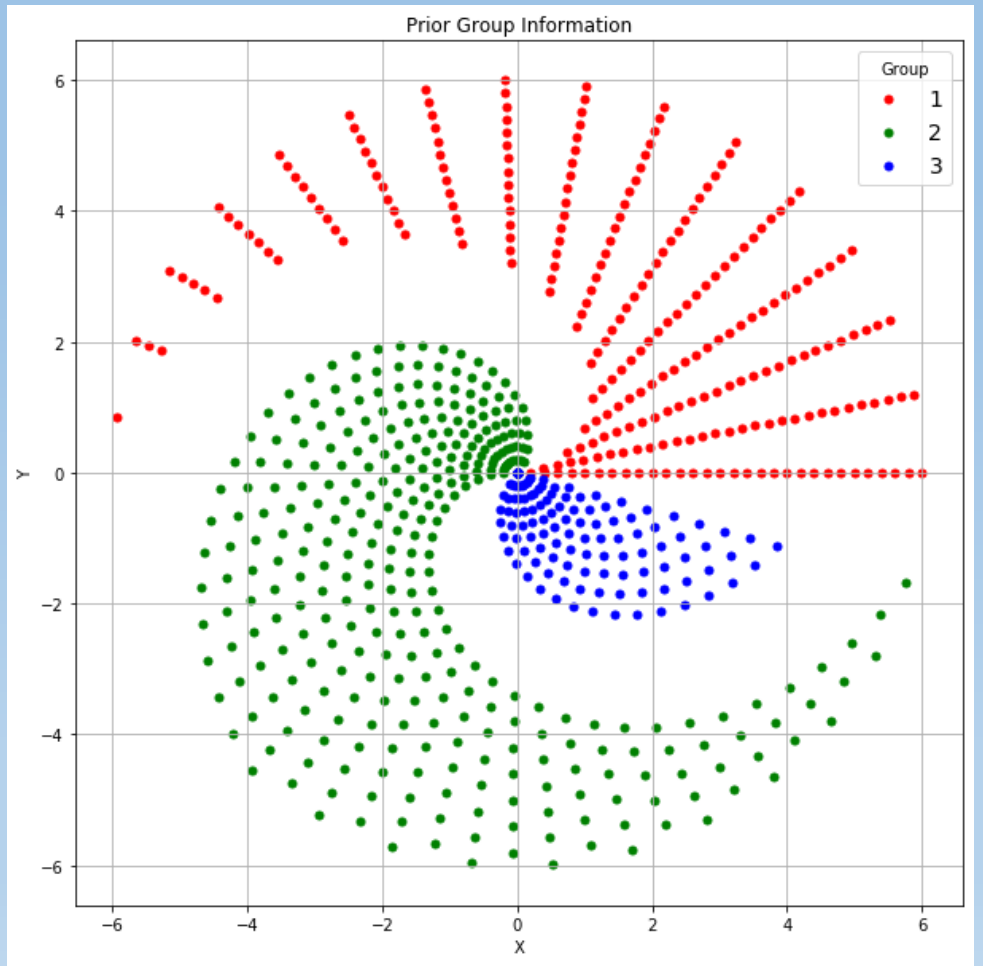
The SVC() function recommends OVR because the OVO method has been deprecated.

Three Shells Example

Three Target Classes

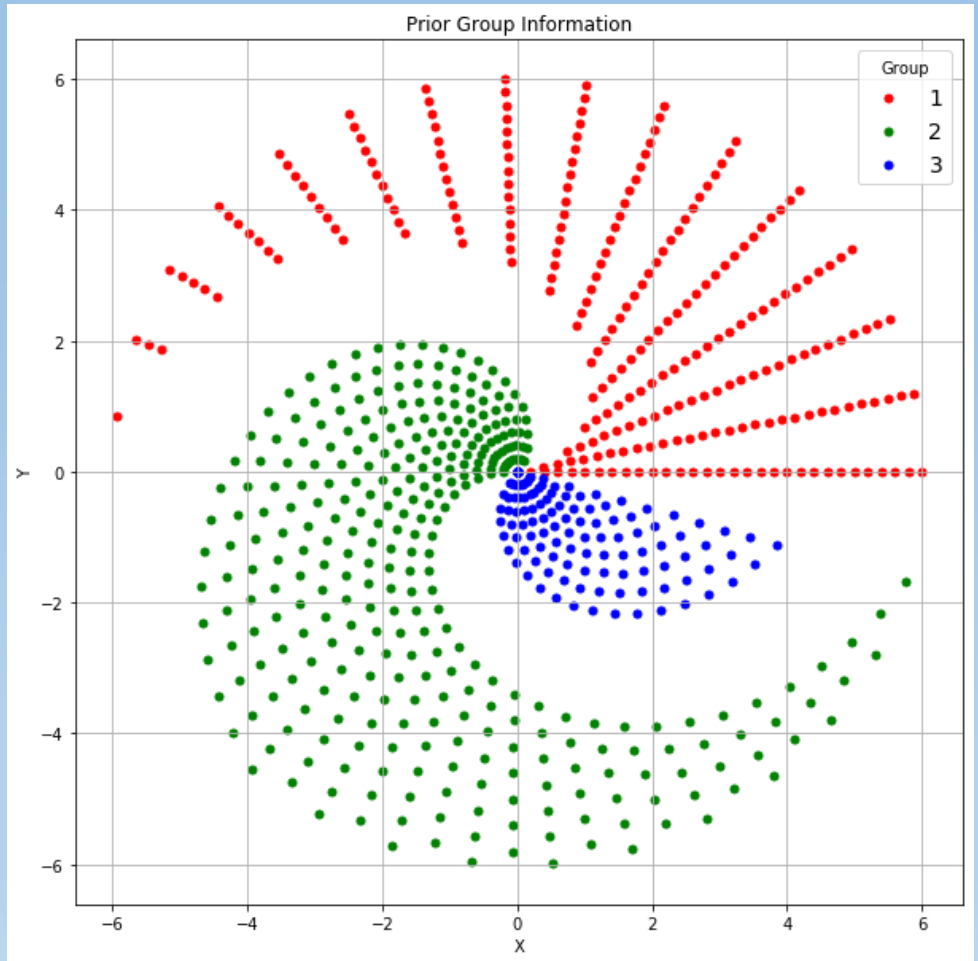
The three shells *meet* at the origin (0,0)

Week 10 SVM Three Segment.py



Use SVM to Determine Linear Separator?

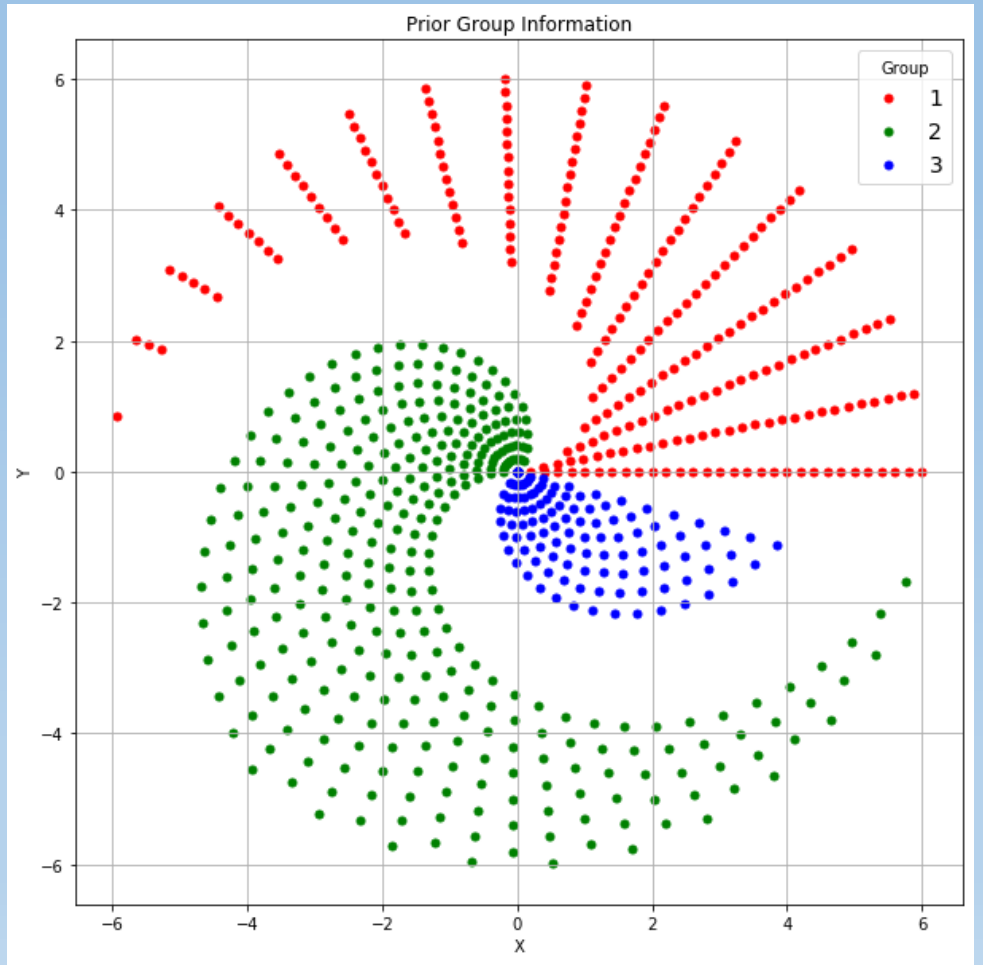
```
# Build Support Vector Machine classifier
xTrain = trainData[['X','Y']]
yTrain = trainData['Group']

svm_Model = svm.SVC(kernel = 'linear',
    decision_function_shape = 'ovr',
    random_state = 20191106, max_iter = -1)
thisFit = svm_Model.fit(xTrain, yTrain)
y_predictClass = thisFit.predict(xTrain)

print('Mean Accuracy = ', metrics.accuracy_score(yTrain,
y_predictClass))
trainData['_PredictedClass_'] = y_predictClass

print('Intercept = ', thisFit.intercept_)
print('Coefficients = ', thisFit.coef_)

# get the separating hyperplane
xx = numpy.linspace(-6, 6)
yy = numpy.zeros((len(xx),3))
for j in range(3):
    w = thisFit.coef_[j,:]
    a = -w[0] / w[1]
    yy[:,j] = a * xx - (thisFit.intercept_[j]) / w[1]
```



Three Shells Example

- One Class Over the Rest (OVR) performs the SVM three times.
  1. Group 1 versus Group 2 & 3
  2. Group 2 versus Group 1 & 3
  3. Group 3 versus Group 1 & 2
- Each SVM generates one hyperplane
- Score an observation based on each hyperplane.
- Assign to the highest score group

Hyperplane	Group 1	Group 2	Group 3
1 vs (2 & 3)	+1	+0.5	+0.5
2 vs (1 & 3)	+0.5	+1	+0.5
3 vs (1 & 2)	+0.5	+0.5	+1

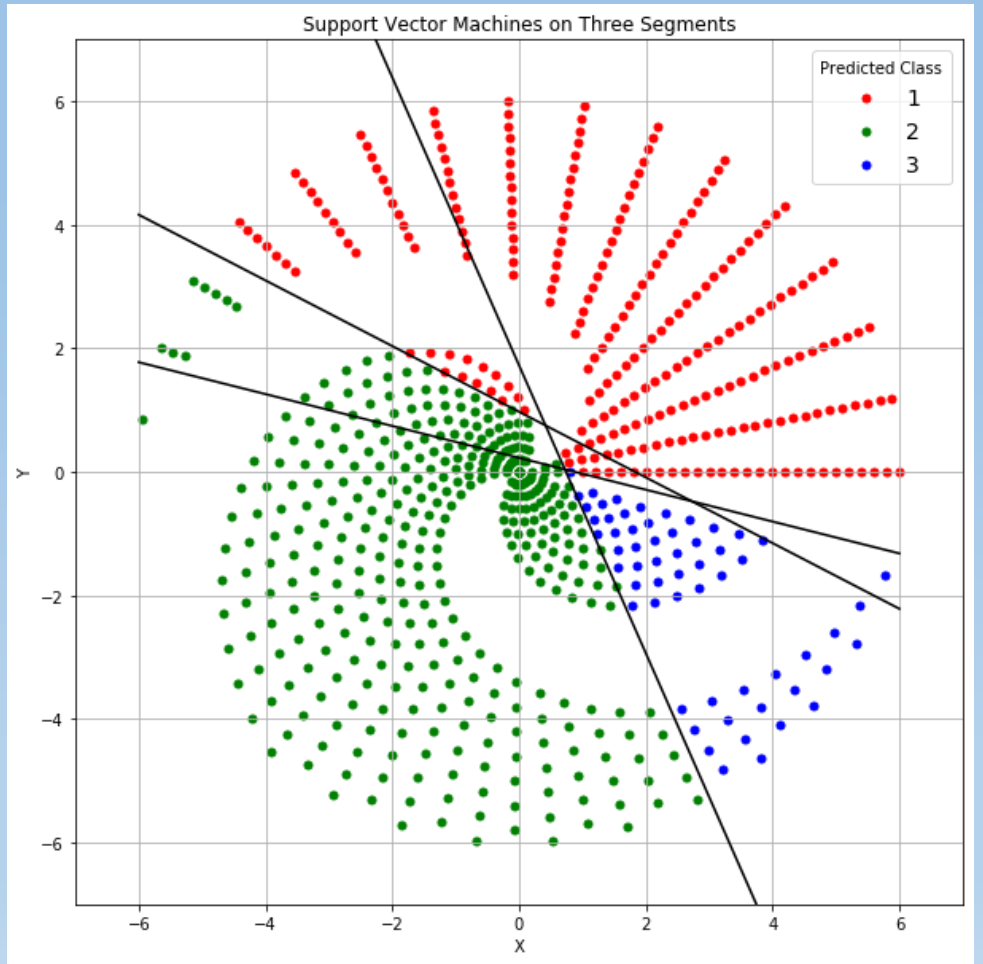


Three Shells Example

- Accuracy is 0.8148
- The three hyperplanes are:

Hyperplane	Intercept	X	Y
0	-1.33201658	0.72885719	1.37117503
1	-0.78031386	0.89138290	3.45622392
2	0.50821464	-0.68709132	-0.29486694

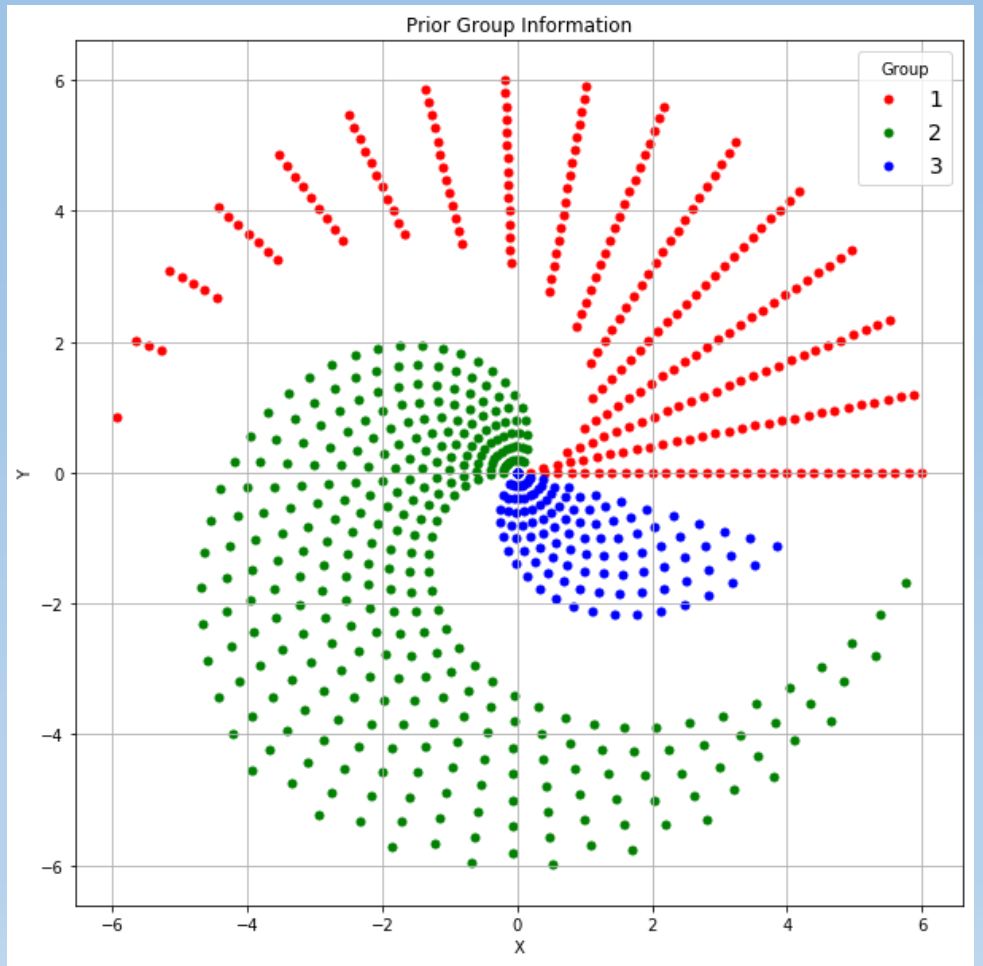
- Obviously, the hyperplanes do not classify the three shells well!



Three Shells Example

Polar Coordinates?

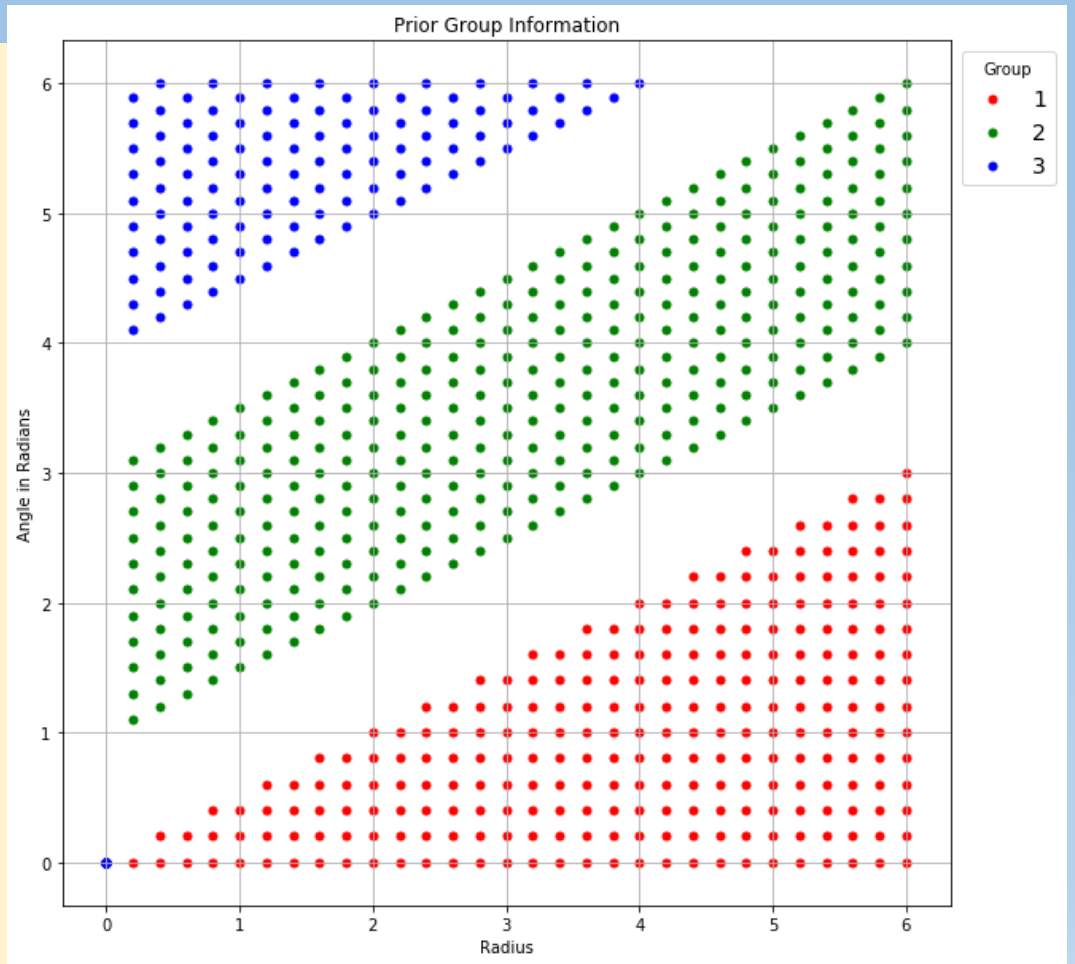
Transform a curve into a straight line



Convert to Polar Coordinates

```
# Convert to the polar coordinates
trainData['radius'] = numpy.sqrt(trainData['X']**2 +
                          trainData['Y']**2)
trainData['theta'] = numpy.arctan2(trainData['Y'],
                         trainData['X'])

# ArcTan2 gives angle from -Pi to Pi
# Make the angle from 0 to 2*Pi
def customArcTan (z):
    theta = numpy.where(z < 0.0, 2.0*numpy.pi+z, z)
    return (theta)

trainData['theta'] = trainData['theta'].apply(customArcTan)

# Build Support Vector Machine classifier
xTrain = trainData[['radius','theta']]
yTrain = trainData['Group']
```

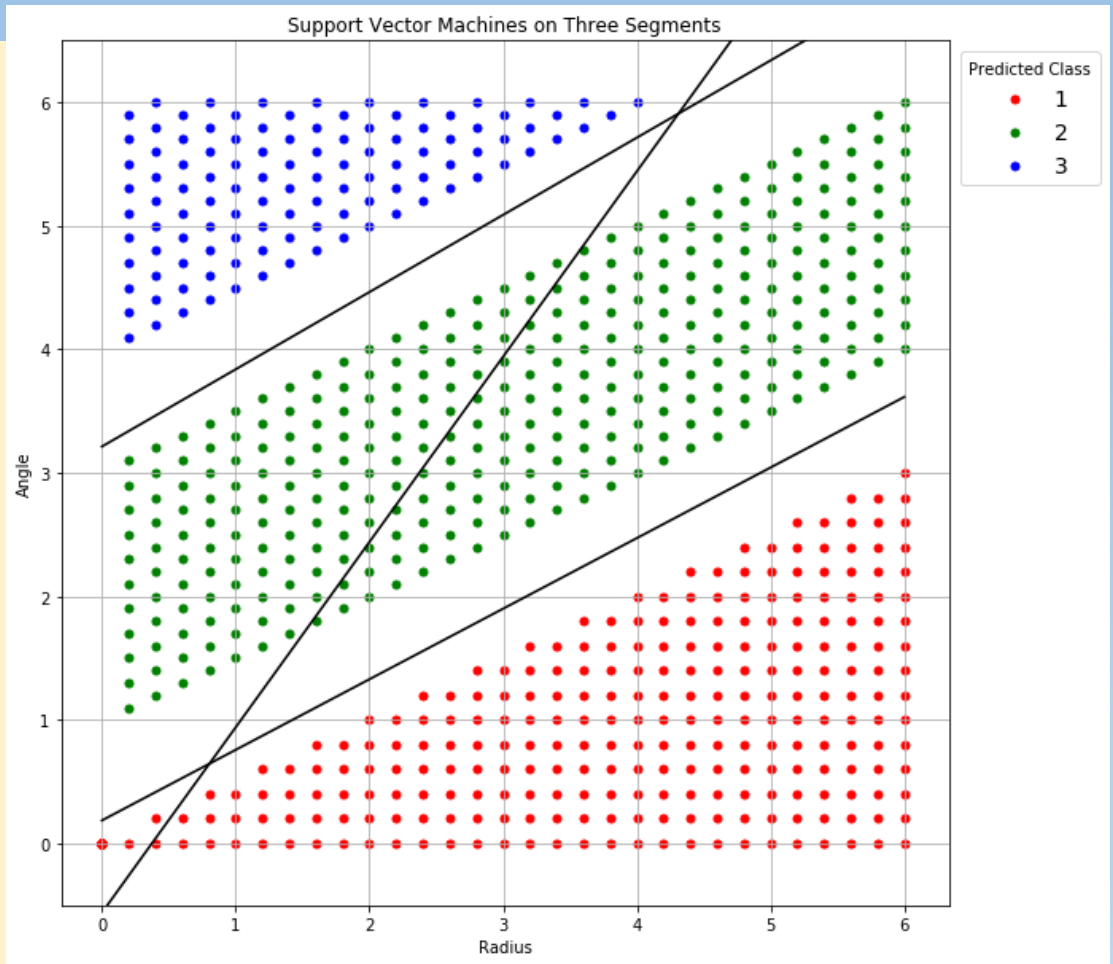
Clearly, there are three separable segments under the Polar Coordinates



Convert to Polar Coordinates

```
xTrain = trainData[['radius','theta']]
yTrain = trainData['Group']

svm_Model = svm.SVC(kernel = 'linear',
decision_function_shape = 'ovr',
                    random_state = 20191106, max_iter = -1)
thisFit = svm_Model.fit(xTrain, yTrain)
y_predictClass = thisFit.predict(xTrain)

print('Mean Accuracy = ', metrics.accuracy_score(yTrain,
y_predictClass))
trainData['_PredictedClass_'] = y_predictClass

print('Intercept = ', thisFit.intercept_)
print('Coefficients = ', thisFit.coef_)
```

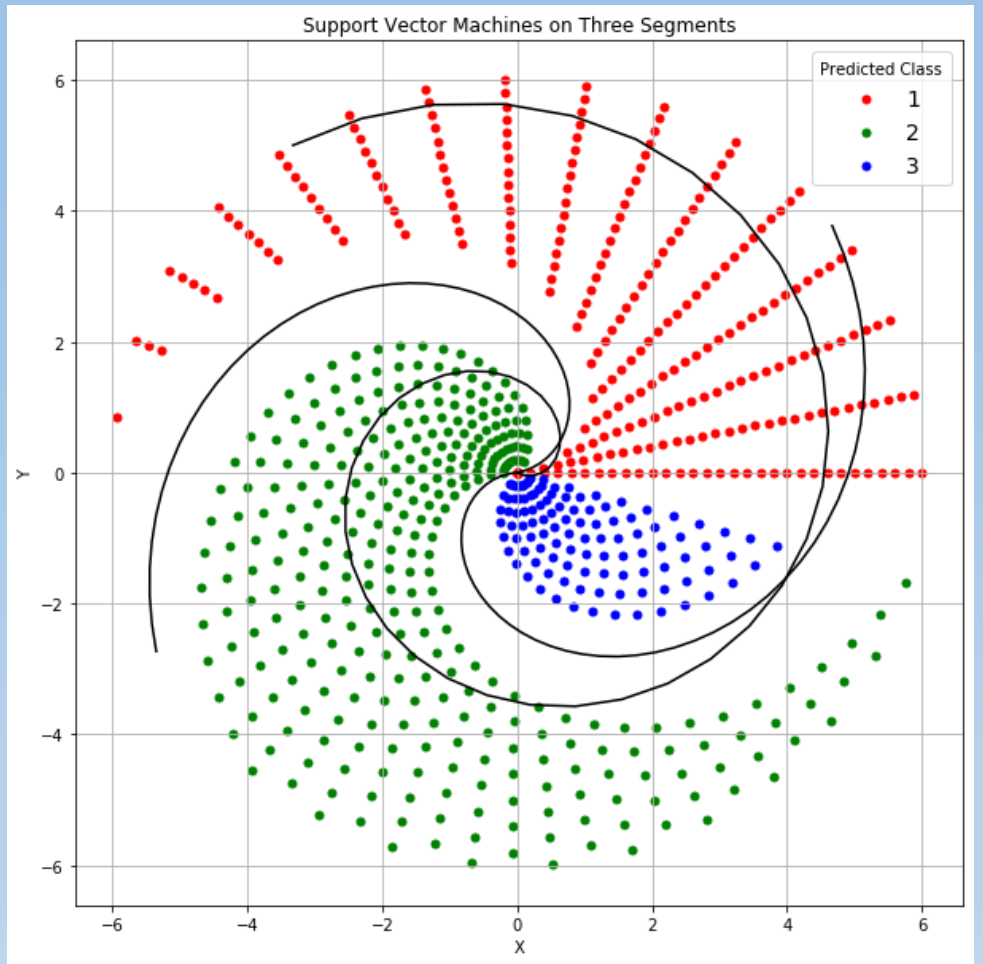
```
Mean Accuracy = 0.9693593314763231
Intercept = [ 0.44928429 -1.          4.21469585]
Coefficients = [[ 1.37802885 -2.41217043]
                [ 2.67692307 -1.78461539]
                [ 0.8200091  -1.31201103]]
```



Back to Cartesian Coordinates

```
# Back-transform the hyperplane from the Polar coordinates
to the Cartesian coordinates

h0_xx = xx * numpy.cos(yy[:,0])
h0_yy = xx * numpy.sin(yy[:,0])

h1_xx = xx * numpy.cos(yy[:,1])
h1_yy = xx * numpy.sin(yy[:,1])

h2_xx = xx * numpy.cos(yy[:,2])
h2_yy = xx * numpy.sin(yy[:,2])
```

Except for the points near the origin, all the points are correctly classified.



Final Remarks

- Support Vector Machines are primarily for classifying a binary target variable.
- In addition to classification, Support Vector Machines provide the line in the sand. The line ensures that the margin between the line and the support vectors are widest. This helps the prediction of new data.
- Support Vector Machines work well for groups of compact observations. However, it is likely to fail for groups of connected observations.
- Consider transforming the data to another coordinate system such that the segments are more separable

Assignment 4

- Due at 11:59 pm on Sunday, April 4, 2021
- Submit your answers as a PDF file
- Submit your Python codes as .py files, otherwise liable for 50% deduction
- No more than two attempts
- Only grade the most recently submission