Fourth Industrial Summer School

Day 5

Discriminant Classification

Session Objectives

- ✓ Discriminative Classification
  - Linear Discriminant Analysis
    - How it works
    - Python Example
    - Try it your self
  - Support Vector Machines(SVM)
    - Hard Margin
    - Soft Margin
    - Kernel tricks
  - Hands on
- ✓ Semi-supervised Learning
  - When to use
  - Python Example
  - Hands on

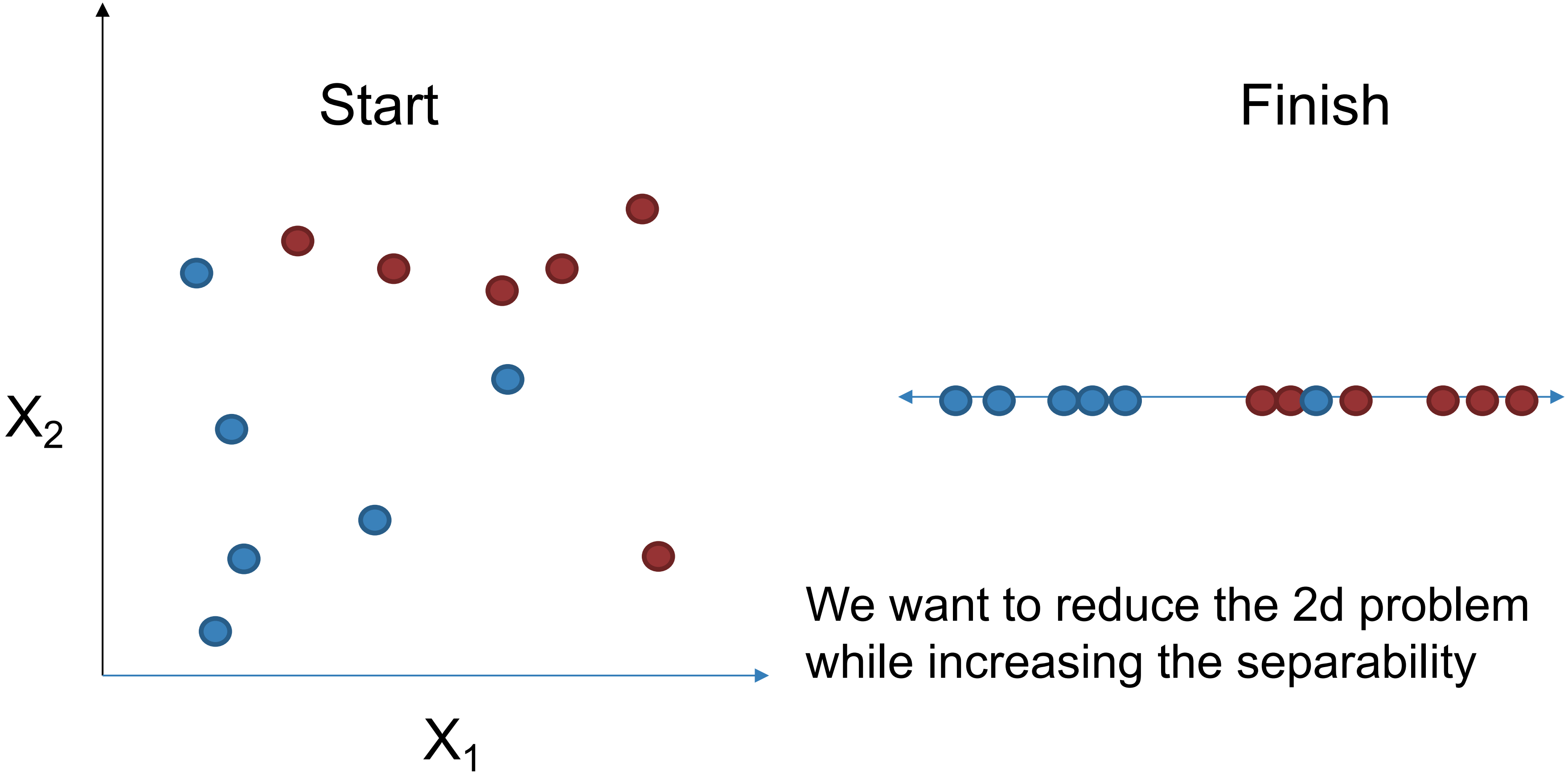
Linear Discriminant analysis

- Linear Discriminant analysis or LDA is a classification method
- Developed in 1936 by R. A. Fisher
- It is a mathematically robust and often produces models whose accuracy is as good as complex ones.

Main idea

- The main idea of LDA is to reduce the dimensionality of the data while considering maximum separability.
- This is not similar to PCA that reduces the dimensionality while focusing on a dimension with the most variations.
- Linear Discriminant Analysis (LDA) is like PCA, but it focuses on maximizing the separability among known classes.

Contd

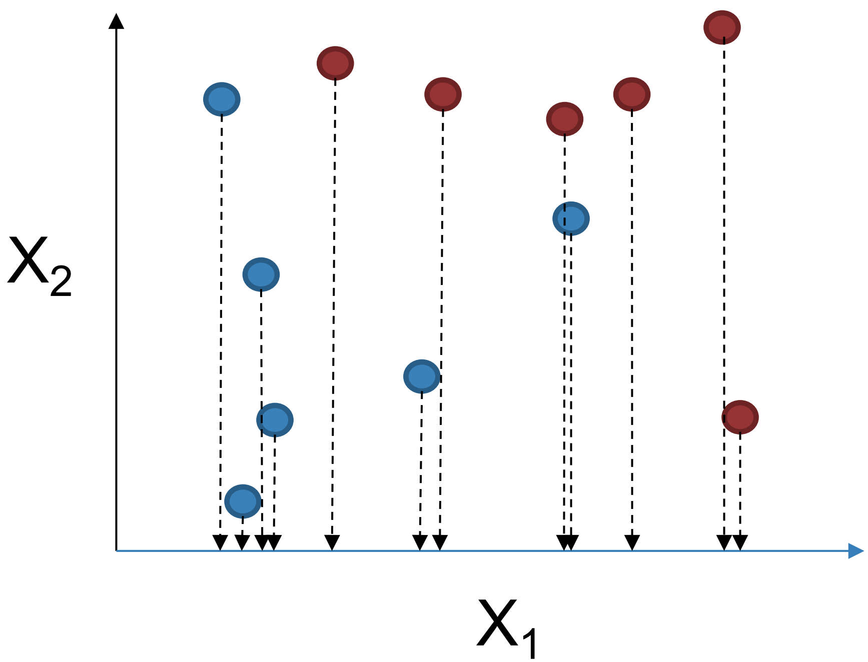


We want to reduce the 2d problem while increasing the separability

What is the best way to do that?

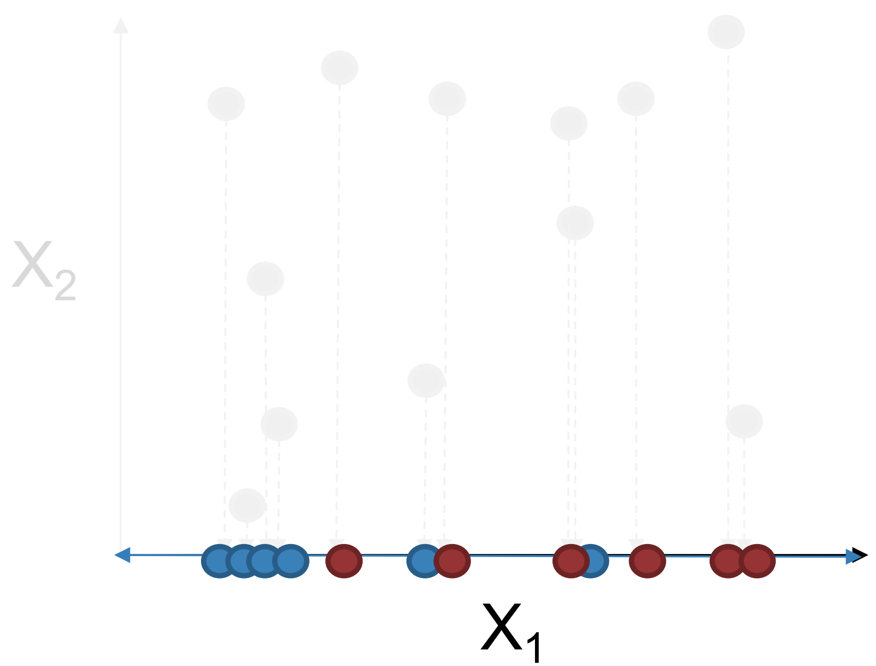
Contd

- One option is to ignore X_2 and focus on X_1

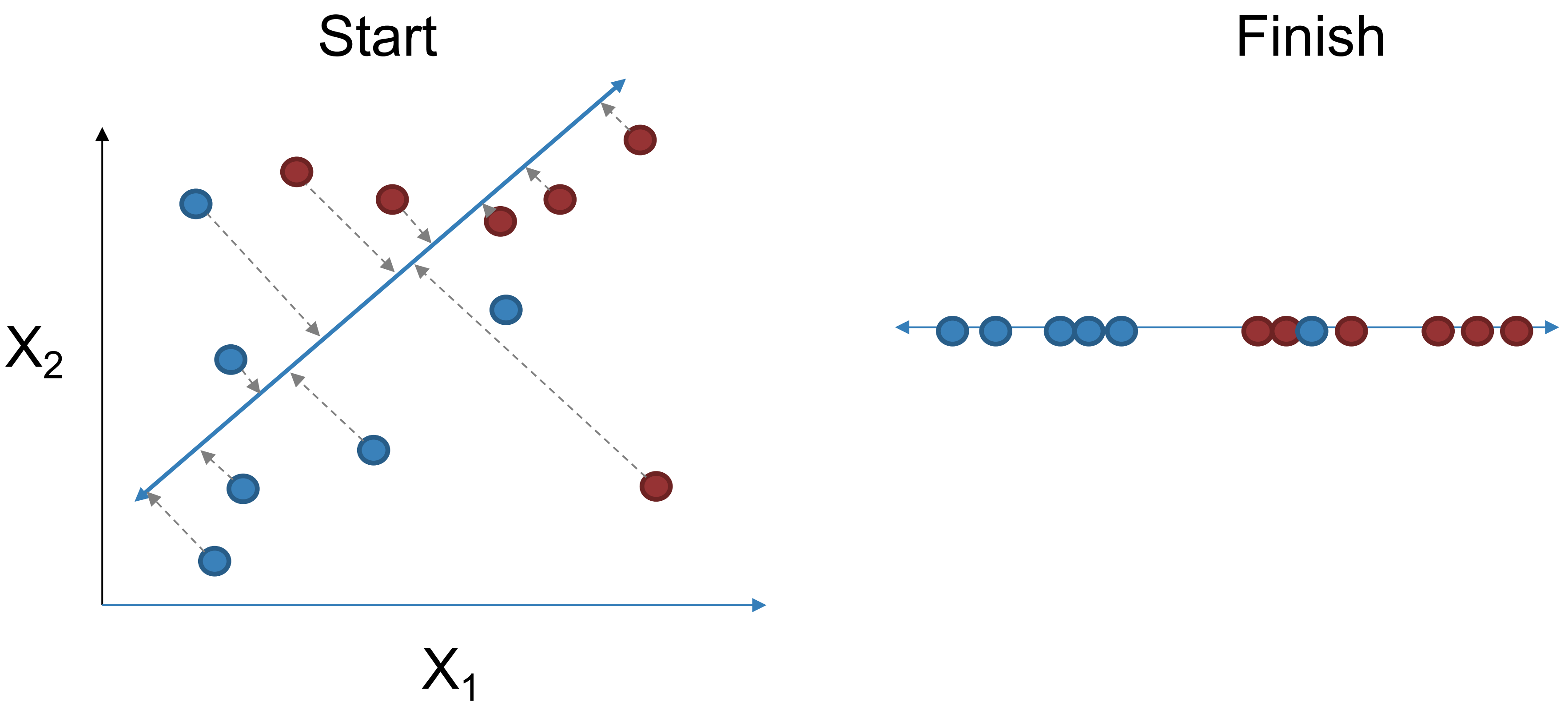


Contd

- One option is to ignore X_2 and focus on X_1



Contd.



Contd

- LDA is doing that reduction by utilizing the information of all dimensions. It finds a new dimension that satisfies two conditions
  - Maximizing the distance between the classes centroids

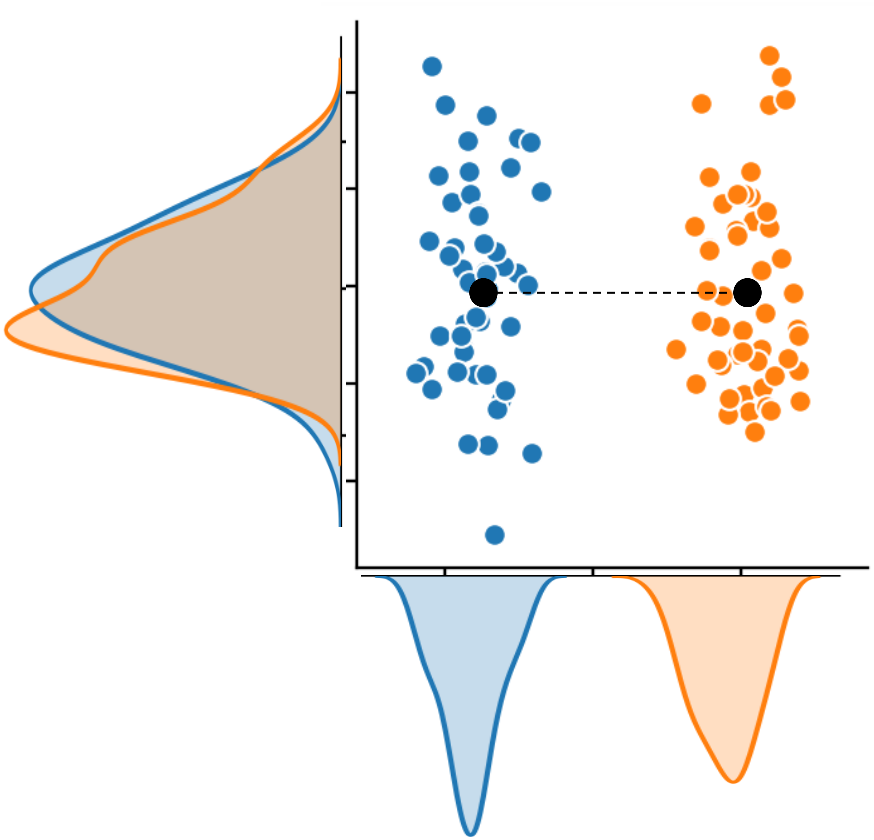
$$d = (\mu_1 - \mu_2)^2$$

  - Minimizing the speared within each class.

$$Sw = s_1^2 + s_2^2$$

Assessing Discrimination of the model

- One way to assess the effectiveness of the discrimination is to compute the Mahalanobis distance between the groups!



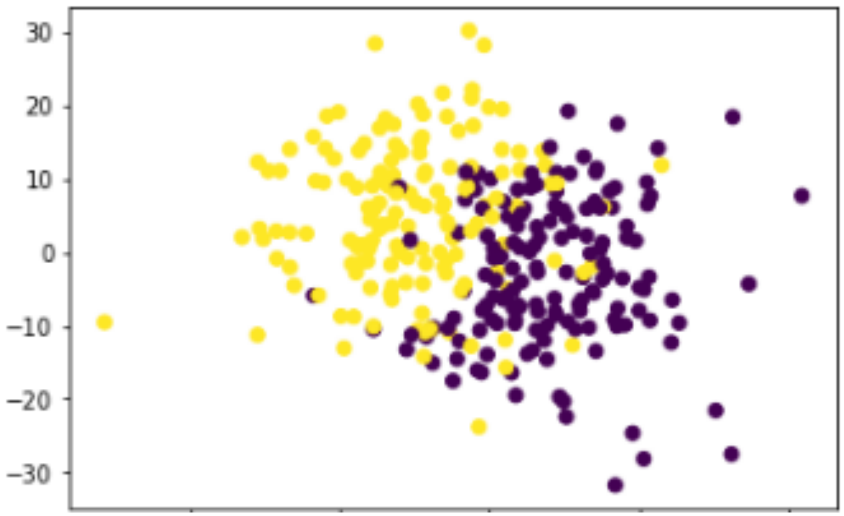
LDA in steps

1. Compute the frequency of each class $p(c_1), p(c_2)$
2. Compute the mean of each group $\mu_1,\ \mu_2$
3. Compute the covariance matrix of each group $\Sigma_1,\ \Sigma_2$
4. Compute the pooled covariance matrix: $\Sigma_p = \frac{1}{n_1+n_2}(n_1\Sigma_1 + n_2\Sigma_2)$
5. Compute the inverse of Σ_p^{-1}
6. Compute the linear combination parameters: $\beta = \Sigma_p^{-1}(\mu_1 - \mu_2)$
7. Apply the decision rule on the new data to classify

$$D(X) = \begin{cases} Class\ 1: & \beta^T\left(x - \left(\frac{\mu_1 + \mu_2}{2}\right)\right) > \log\left(\frac{p(c_1)}{p(c_2)}\right) \\ Class\ 2: & \beta^T\left(x - \left(\frac{\mu_1 + \mu_2}{2}\right)\right) \leq \log\left(\frac{p(c_1)}{p(c_2)}\right) \end{cases}$$

Example(Make data)

```
import matplotlib.pyplot as plt
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=300, centers=2, #will define classes
                           n_features=20, cluster_std=10,
                           random_state =6)
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



Copy

```
# making some dataa (Linearly separable)
import matplotlib.pyplot as plt
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=300, centers=2, #will define classes
                           n_features=20, cluster_std=10,
                           random_state =6)
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```

Example (Split data)

```
# Split the data into training and test
from sklearn.model_selection import train_test_split
X2, X_test, y2, y_test = train_test_split(X, y, test_size=0.1, shuffle=True)
X_train, X_Validate, y_train, y_Validate = train_test_split(X2, y2, test_size=0.2, shuffle=True)

print('Training set shape:', X_train.shape,
      '\nValidatation set shape:', X_Validate.shape,
      '\nTesting set shape:', X_test.shape )
```

```
Training set shape: (216, 20)
Validatation set shape: (54, 20)
Testing set shape: (30, 20)
```

Copy


```
# Split the data into training and test
from sklearn.model_selection import train_test_split
X2, X_test, y2, y_test = train_test_split(X, y, test_size=0.1, shuffle=True)
X_train, X_Validate, y_train, y_Validate = train_test_split(X2, y2,
test_size=0.2, shuffle=True)

print('Training set shape:', X_train.shape,
      '\nValidatation set shape:', X_Validate.shape,
      '\nTesting set shape:', X_test.shape )
```

Example(LDA)

```
# Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Using it as classifier, I don't need to specify parameters
lda = LinearDiscriminantAnalysis(n_components=2).fit(X_train, y_train)

print('Train accuracy:%0.2f'% lda.score(X_train, y_train))
print('Validation accuracy:%0.2f'% lda.score(X_Validate, y_Validate))
print('Tesing accuracy:%0.2f'% lda.score(X_test, y_test))
```

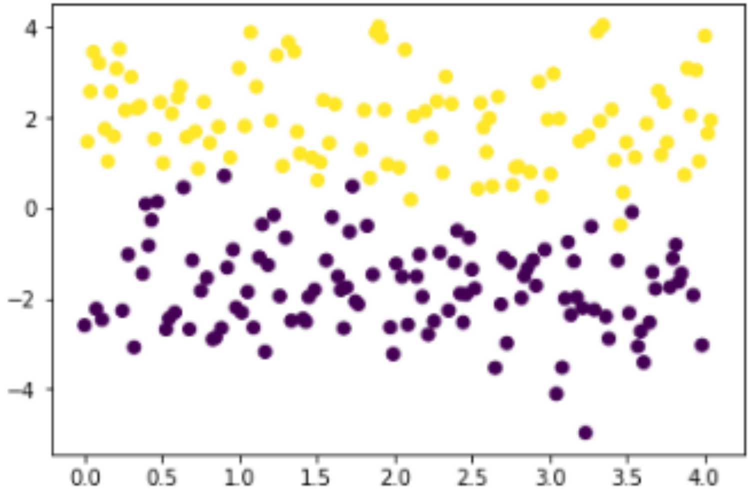
```
Train accuracy:0.98
Validation accuracy:0.93
Tesing accuracy:0.93
```

Copy


```
# Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Using it as classifier, I don't need to specify parameters
lda = LinearDiscriminantAnalysis(n_components=2).fit(X_train, y_train)

print('Train accuracy:%0.2f'% lda.score(X_train, y_train))
print('Validation accuracy:%0.2f'% lda.score(X_Validate, y_Validate))
print('Tesing accuracy:%0.2f'% lda.score(X_test, y_test))
```

Example (Feature Reduction)

```
# in case of two classes: show how it separates the two classes
newX = lda.transform(X_train)
newX.shape
plt.scatter(np.linspace(0,max(newX),len(newX)), newX, c=y_train )
plt.show()
```

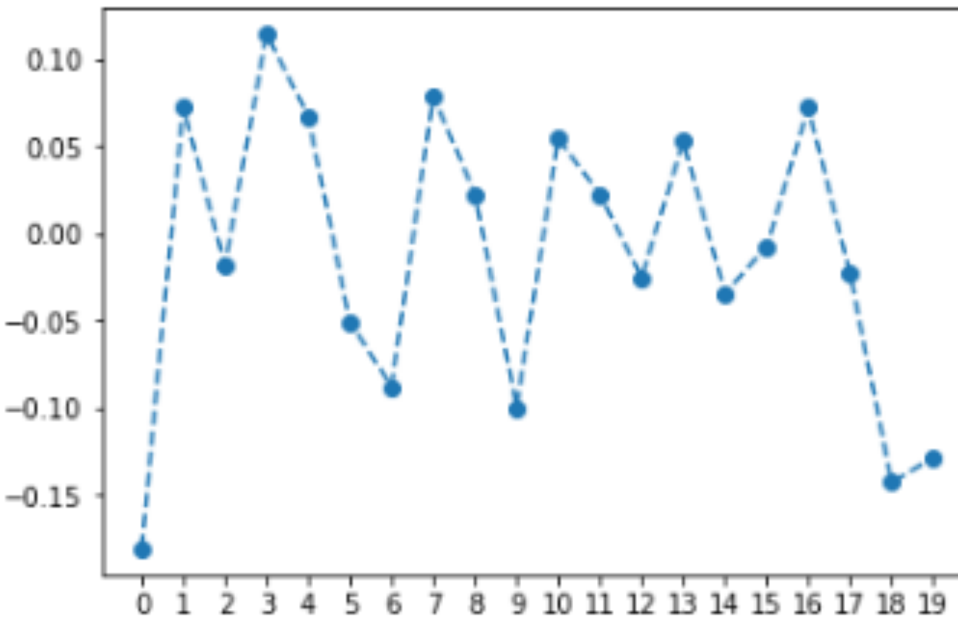


Copy

```
# in case of two classes: show how it separates the two classes

# To transform any new data into LDA space, use .transform(data)
newX = lda.transform(X_train)

plt.scatter(np.linspace(0,max(newX),len(newX)), newX, c=y_train )
plt.show()
```

Feature Contribution

```
# data properties
# The large the coef of a feature, the more important its role in the discrimination
plt.plot(np.arange(0,lda.coef_.size), lda.coef_[0], 'o--')
plt.xticks(np.arange(0,lda.coef_.size))
plt.show()
```



Copy

# data properties
# The large the coef of a feature, the more important its role in the discrimination
plt.plot(np.arange(0,lda.coef_.size), lda.coef_[0], 'o--')
plt.xticks(np.arange(0,lda.coef_.size))
plt.show()

Try it out

- You may generate more features say 50 and experiment LDA
- Increase number of classes
- Increase the cluster std
- Try different data shapes (circles or moons)

Pros and Cons

- Derived Linearity rather than assumed
- Considers the ratio (between groups SS)/(within groups SS) as large as possible.
- Can play a role in feature selection by looking on the coefficients

- Assumes conditional densities of clusters are approximately normal.
- Coefficients may not indicate Feature Significance in case of more than two outcomes problems.

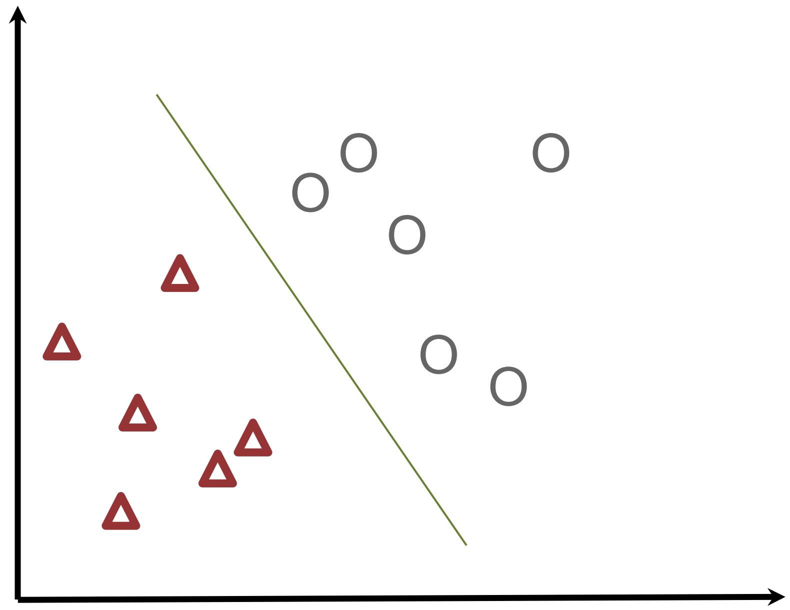
Support Vector Machines

Introduction: history

- SVM was first introduced in **1963** by V. Vapnik and A. Chervonenkis
- It dose really well with linear decision surfaces
- It becomes popular because of its success in modeling many machine learning problems, after its generalization in **1992** by Vapnik et. al (kernel trick)
- The basic idea is to separate a two class data points using a line (in 2-d space)

How it works?

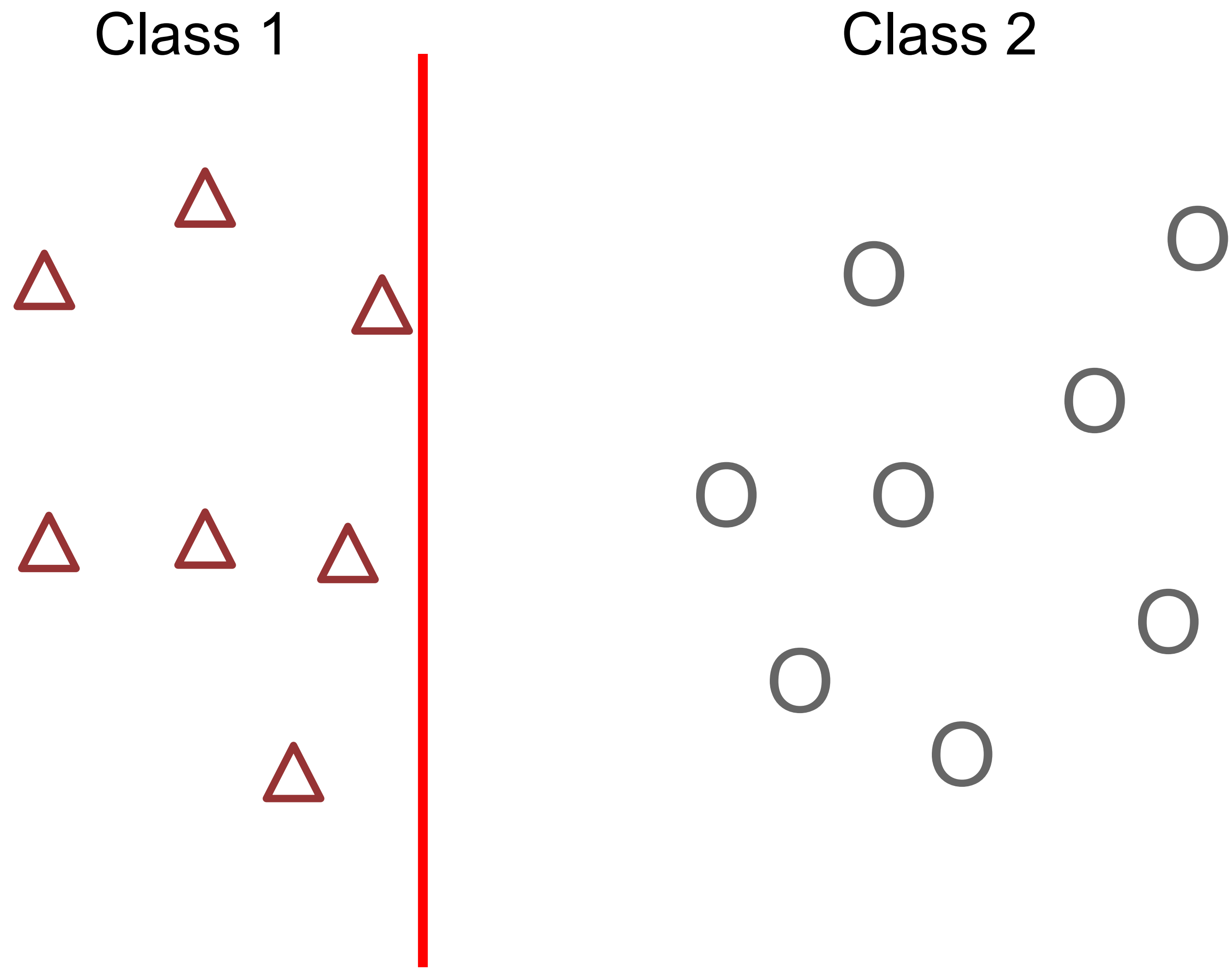
- Assume linear separability for now (and relax this later)



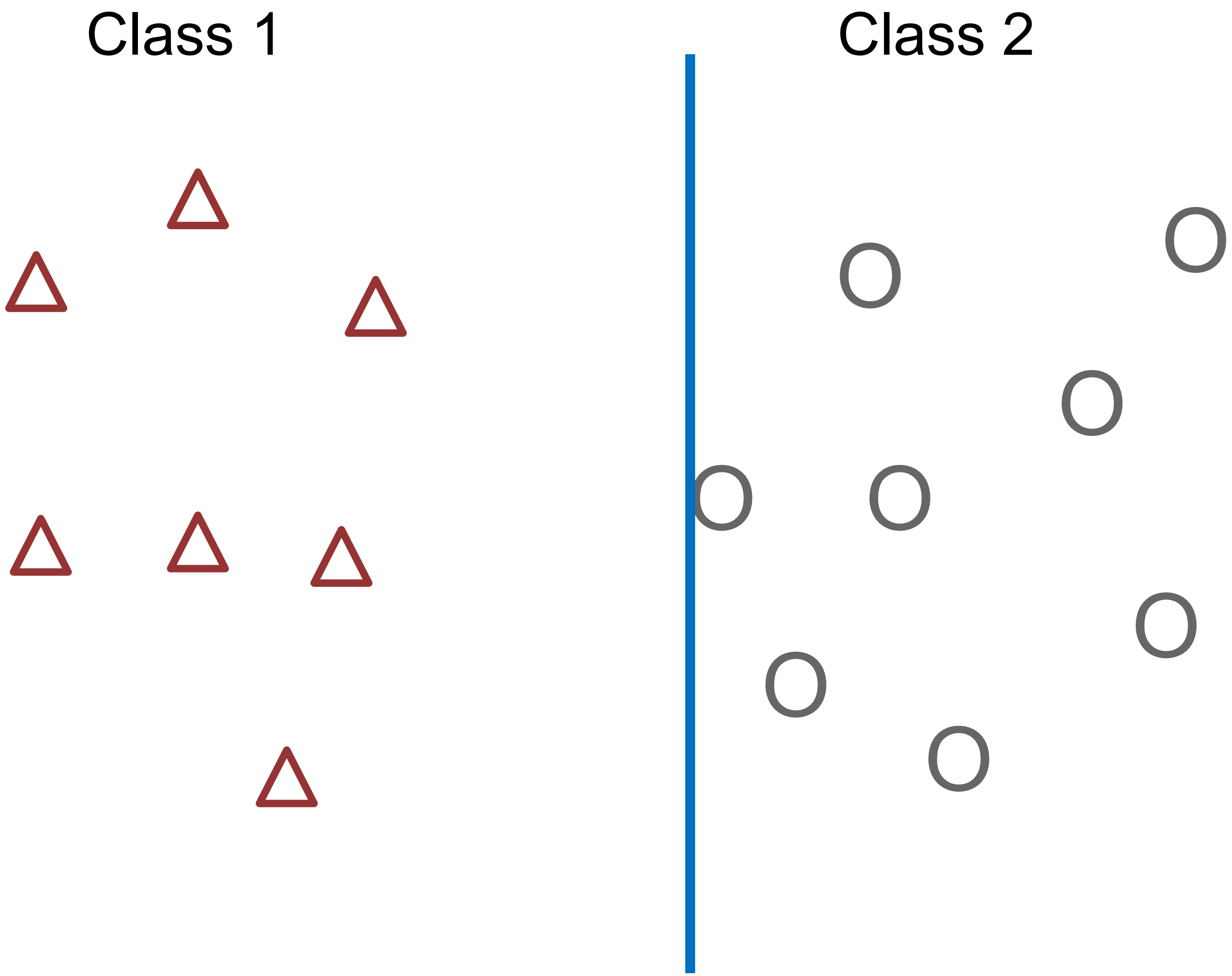
- Given such data points, we want to draw a line that separates the two classes.

$$w_1x_1 + w_2x_2 + b = 0$$

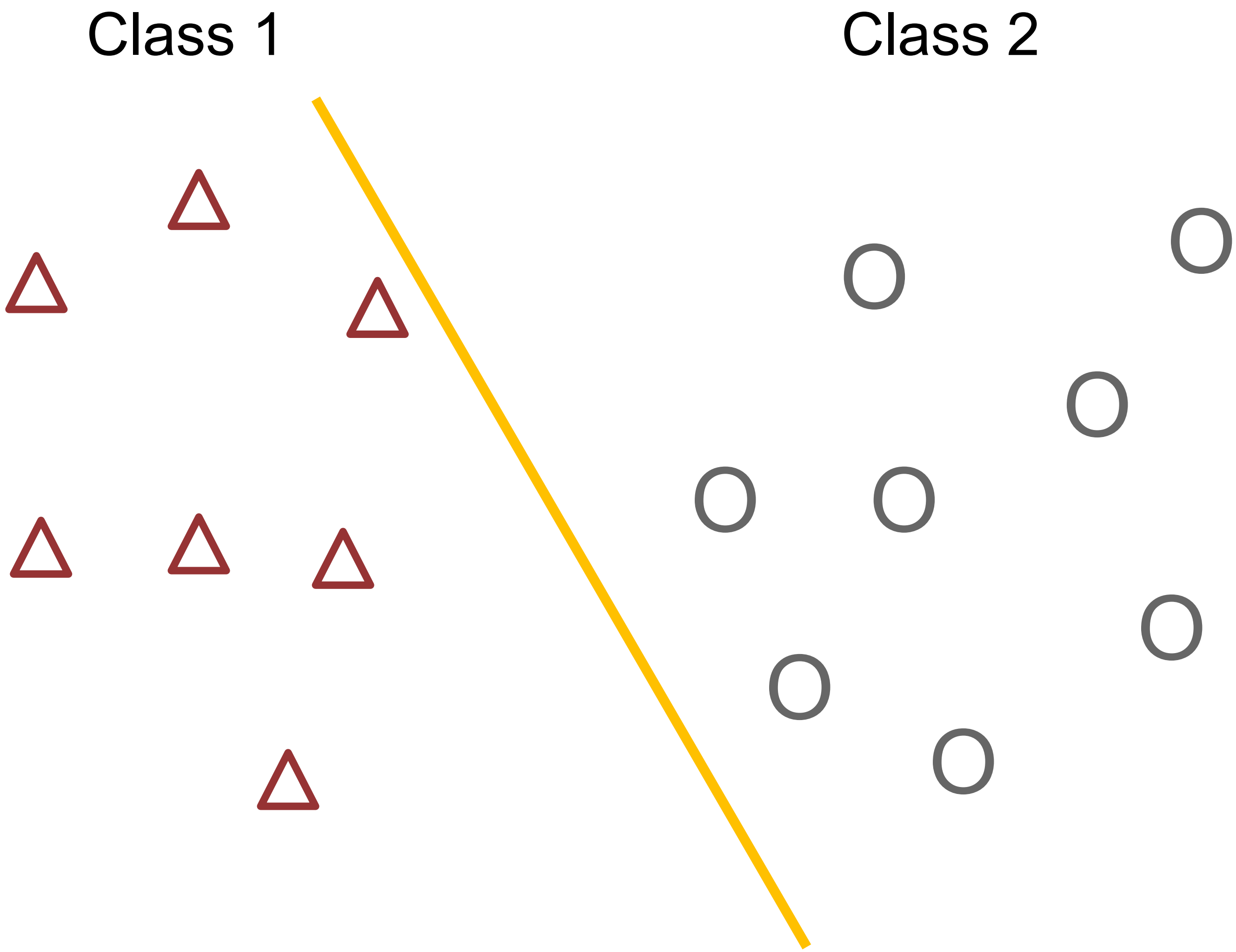
Contd.



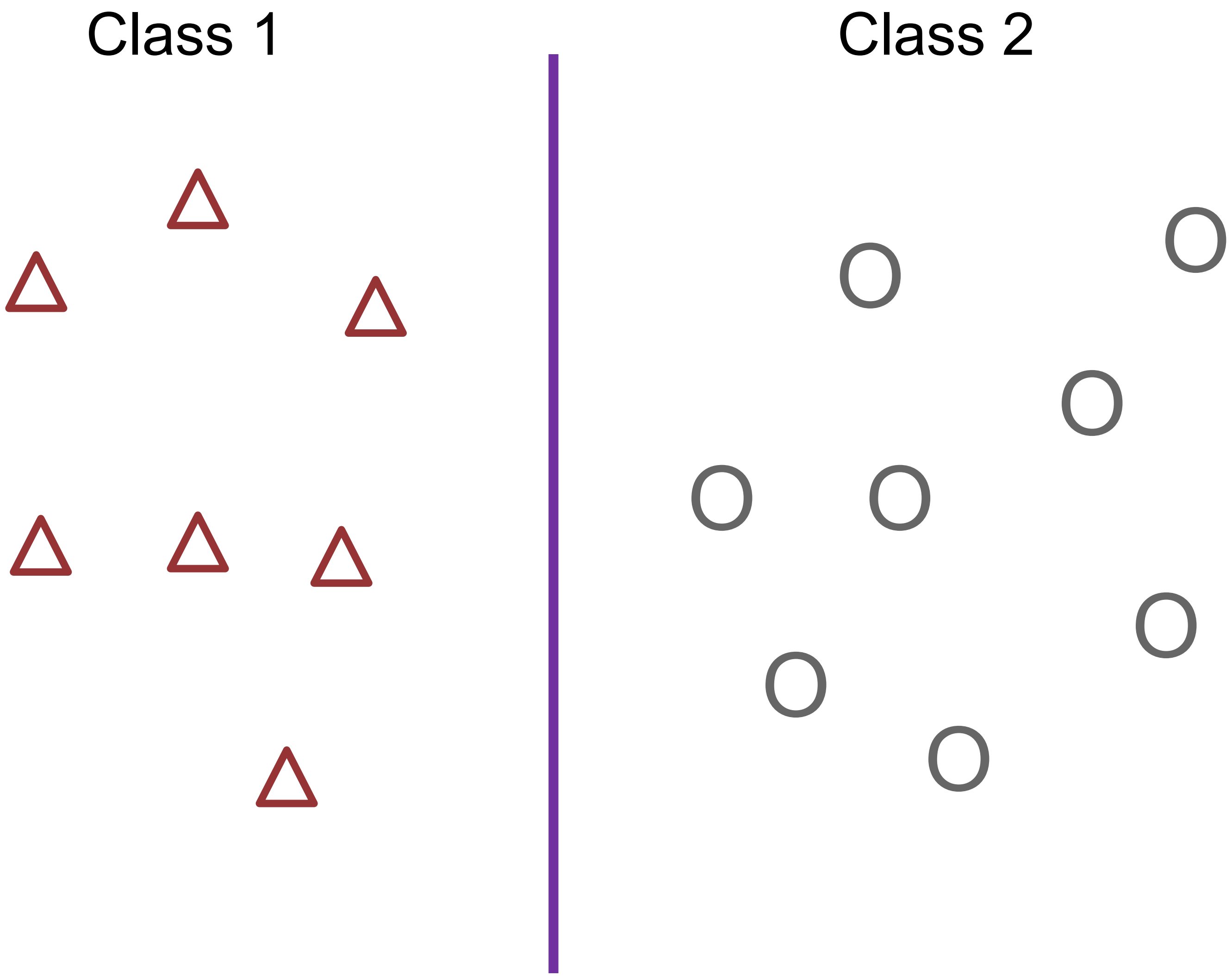
Contd.



Contd.



Optimal decision boundary

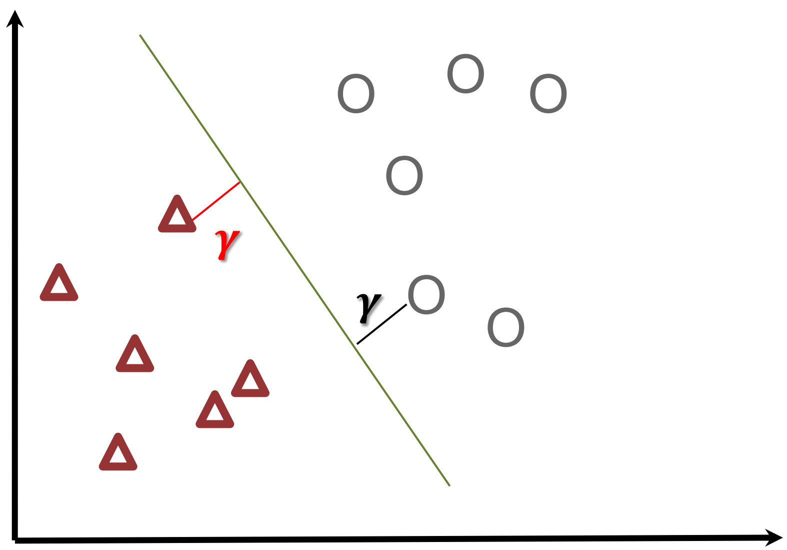


Optimal decision boundary

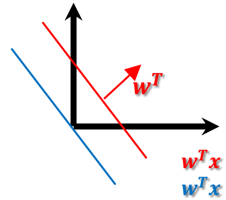
We're looking for a hyperplane that best separates the classes

How it works?

- Instead, we want to maximize the margin. In other words, we want to have a decision boundary far away from both classes as possible!

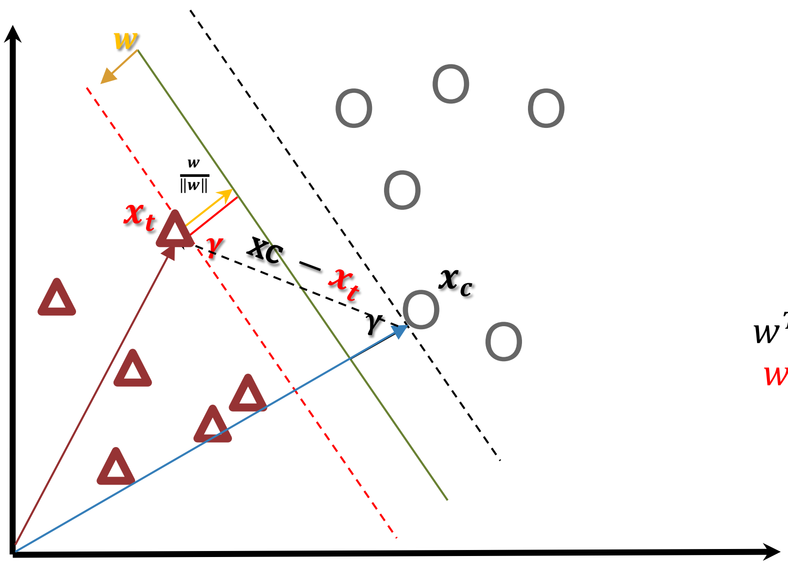


Define a margin



$w^T x + b = 0$, equate b=0
$w^T x = 0$, hence w is ⊥ on x

- SVM wants to maximize the margin between classes γ



$w^T x_c + b \geq +1$
$w^T x_t + b \leq -1$

***max γ*:**

$$(x_c - x_t) . \frac{w}{\|w\|}$$

$$(w. x_c - w. x_t) \frac{1}{\|w\|}$$

$$((1\text{-b}) - (\text{-1-b})) \frac{1}{\|w\|} = \frac{2}{\|w\|}$$

$$\max \frac{2}{\|w\|} \approx min\|w\| \approx \min \frac{1}{2} \|w\|^2$$

- So maximizing the margin γ, implicitly minimizing the vector **w**
- **Note both constraints have to be achieved.**

$$\max \gamma \approx \min \frac{1}{2} \|w\|^2 \ \ under \ y_i (w. x_i + b) \geq \ +1,$$

Hard margin

- To find the extremum, partial derivative is computed with respect to w and b
  - $\frac{\delta L}{\delta w}$: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
  - $\frac{\delta L}{\delta b}$: $\sum_{i=1}^{n} \alpha_i y_i = 0$
- By plug the value of w back in the Lagrangain function and simplify:

$$maximize: \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j x_i^t . x_j$$

Subject to: $\sum_{i=1}^{n} \alpha_i y_i = 0, \;\; \alpha_i \geq \; 0 \; \forall i$

- Now if you want to make predictions just use:

$$\left(\sum_{i=1}^{n} \alpha_i y_i x_i . \boldsymbol{u}_i + b \right) \geq 1 \, , then\ circles\ otherwise\ triagnles$$

Example: SVM

```
from sklearn import svm
X, y = datasets.make_blobs(n_samples=300, centers=2, #will define classes
                           n_features=20, cluster_std=1,
                           random_state =6)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
svmL= svm.SVC(kernel='linear',
              C=1e20,
              tol= 1e-3,
              max_iter = 1000,
              random_state=None).fit(X_train, y_train)
svmL.score(X_test,y_test )
```

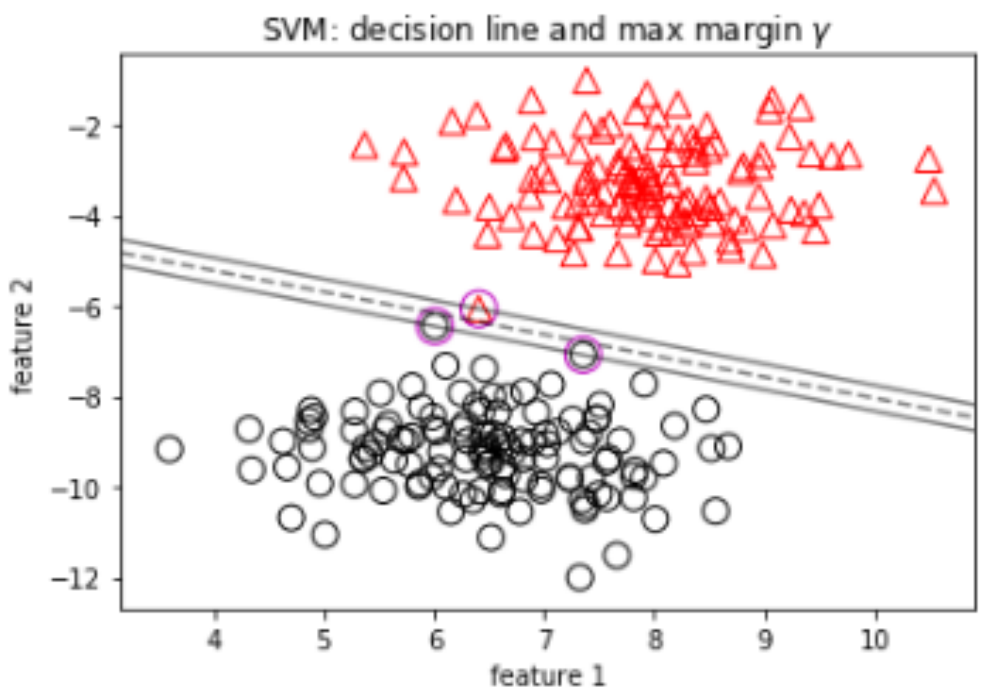
```
1.0
```

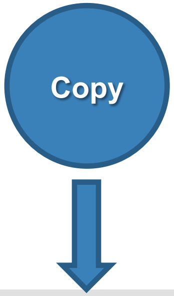
Copy


```
from sklearn import svm
X, y = datasets.make_blobs(n_samples=300, centers=2, #will define classes
                           n_features=20, cluster_std=1,
                           random_state =6)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
svmL= svm.SVC(kernel='linear',
              C=1e20,
              tol= 1e-3,
              max_iter = 1000,
              random_state=None).fit(X_train, y_train)
svmL.score(X_test,y_test )
```

Example: SVM(hard margin)

```
plot_SVM_decisionline2(svmL, X_train, y_train)
```

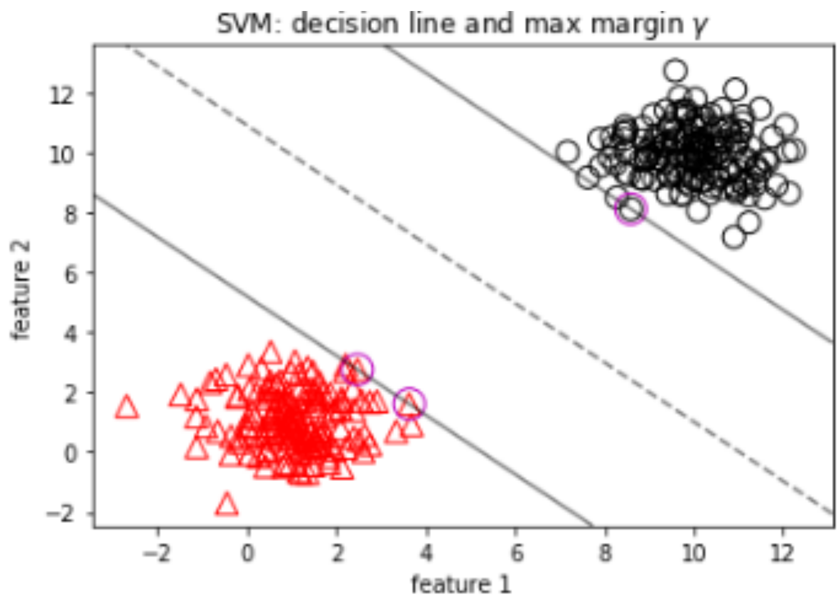





```
plot_SVM_decisionline2(svmL, X_train, y_train)
```

Example: SVM(hard margin)

```
X, y = datasets.make_blobs(n_samples=300, centers=[[1,1], [10,10]],
                           n_features=2, cluster_std=1,
                           random_state =6)

# train the model on this new data
svmL.fit(X,y )

plot_SVM_decisionline2(svmL, X, y)
```



What if one point landed in the other class

Copy

```
X, y = datasets.make_blobs(n_samples=300, centers=[[1,1], [10,10]],
                           n_features=2, cluster_std=1,
                           random_state =6)

# train the model on this new data
svmL.fit(X,y )

plot_SVM_decisionline2(svmL, X, y)
```

Example: SVM(hard margin)

```
# get a list of indecies of class 1
ind= np.where(y==1)[0] # tuple
y[ind[np.random.randint(10)]] = 0

# train the model on this new data
svmL.fit(X,y )

plot_SVM_decisionline2(svmL, X, y)
```

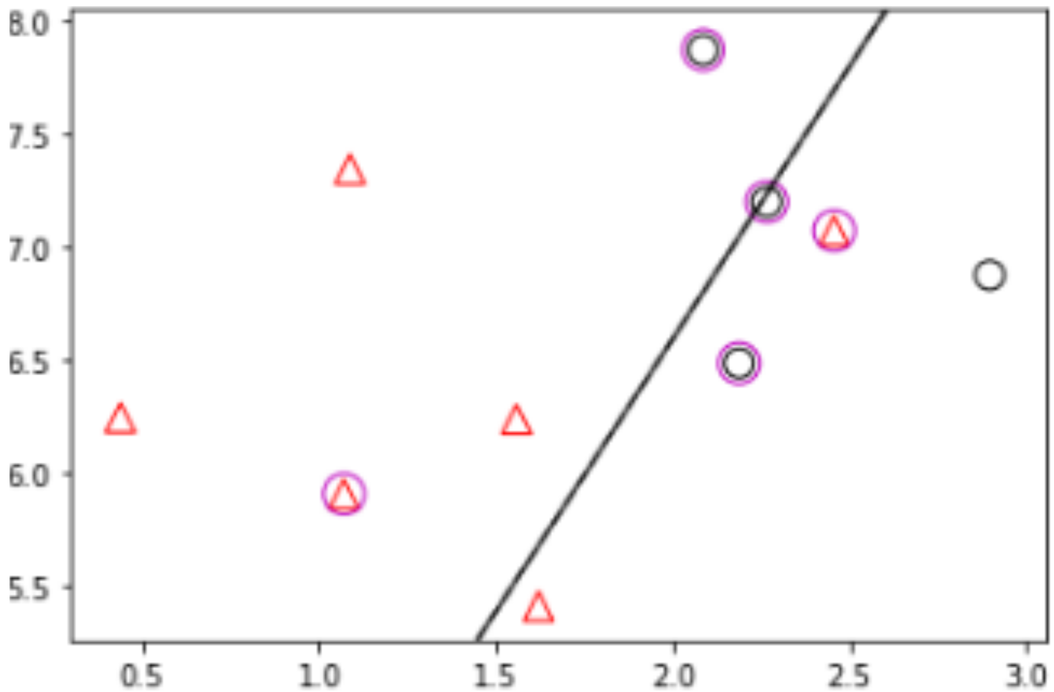
Copy


```
# get a list of indecies of class 1
ind= np.where(y==1)[0] # tuple
y[ind[np.random.randint(10)]] = 0

# train the model on this new data
svmL.fit(X,y )

plot_SVM_decisionline2(svmL, X, y)
```

SVM
Soft-Margin

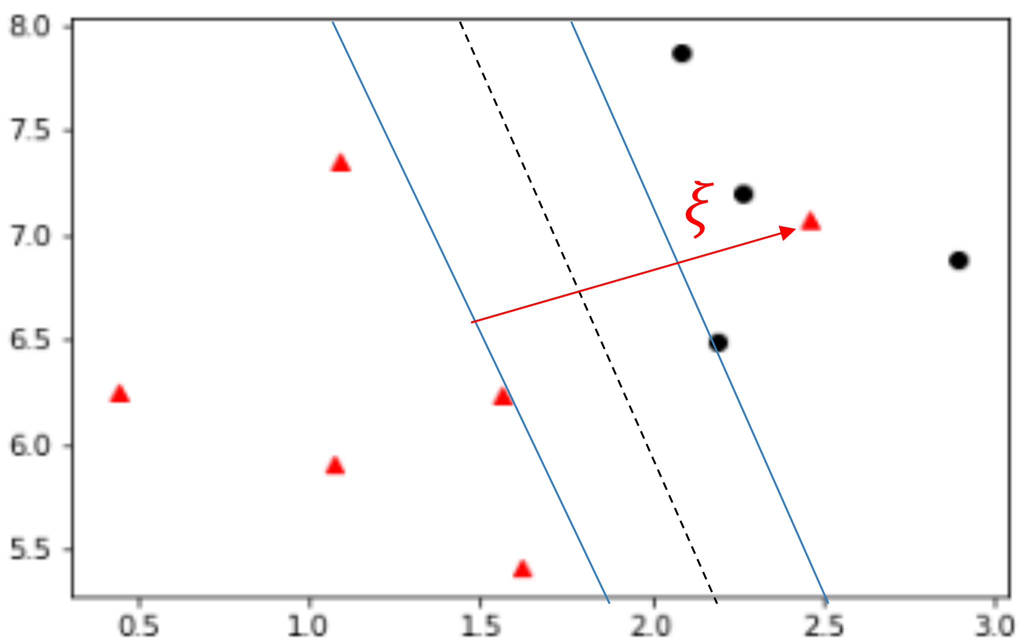
Soft Margin

- In reality, datasets suffer overlapping between class samples.
- Using SVM with Hard-margin optimization won't work!



Soft Margin

- To cope with this issue, the constraints are relaxed by introducing a slack variable ξ controlled by a weight parameter C
- The optimization and constraints becomes:
  - $\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$
  - $y_i(w^T x + b) \geq 1 - \xi\,, \quad \xi \geq 0$



Soft Margin

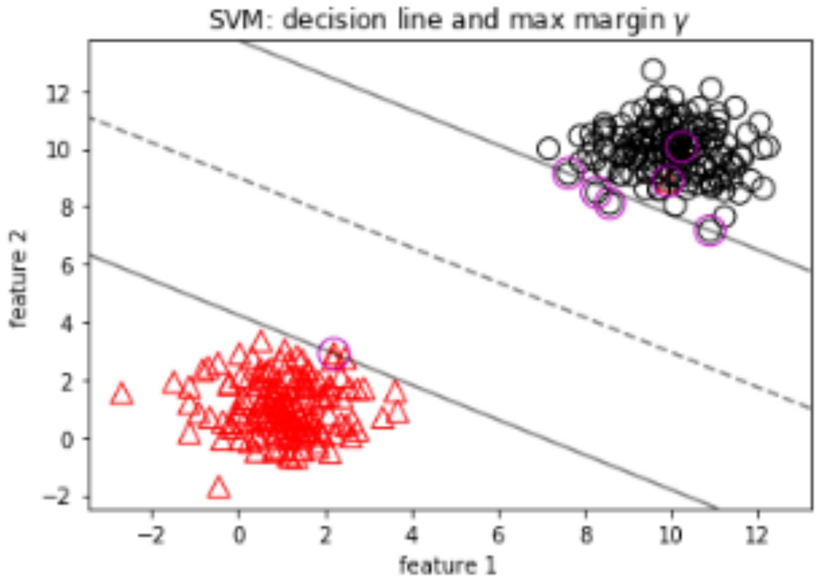
- By solving the new optimization problem using Lagrange multiplier :
- The partial derivatives are computed with respect to w, b and ξ
  - $\frac{\delta L}{\delta w}$: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$
  - $\frac{\delta L}{\delta b}$: $\sum_{i=1}^{n} \alpha_i y_i = 0$
  - $\frac{\delta L}{\delta \xi}$: $\sum_{i=1}^{n} C - \alpha_i - \beta_i, \quad C = \alpha_i + \beta_i, \quad \beta_i = C - \alpha_i$

$$maximize: \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j x_i^t . x_j$$

.

Example: SVM(hard margin)

```
svmL= svm.SVC(kernel='linear',
              C=0.3,
              tol= 1e-3,
              max_iter = 1000,
              random_state=None).fit(X_train, y_train)

# train the model on this new data
svmL.fit(X,y )

plot_SVM_decisionline2(svmL, X, y)
```



Copy

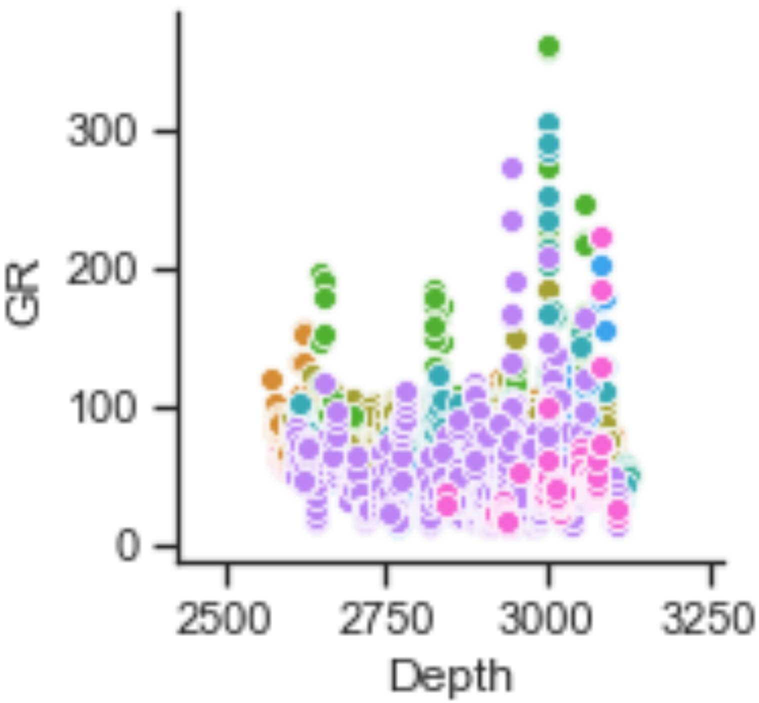
svmL= svm.SVC(kernel='linear',
C=0.3,
tol= 1e-3,
max_iter = 1000,
random_state=None).fit(X_train, y_train)

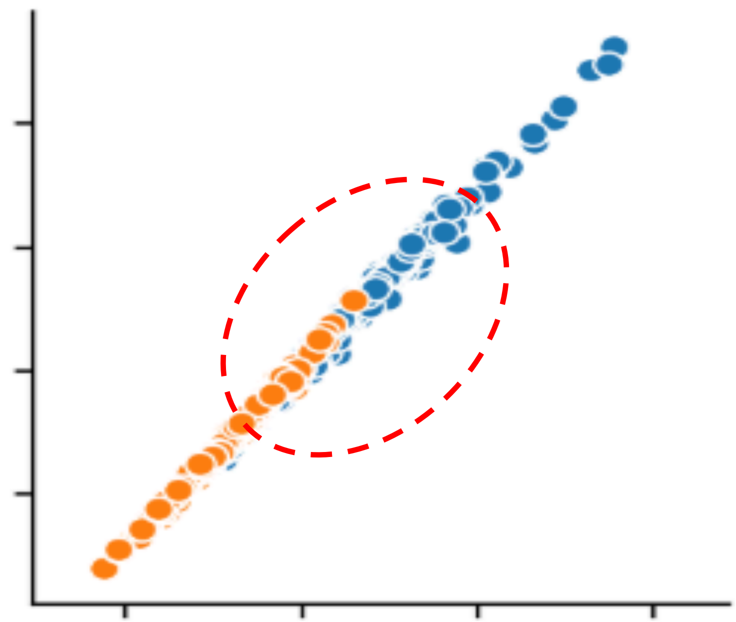
# train the model on this new data
svmL.fit(X,y)

plot_SVM_decisionline2(svmL, X, y)

Real problems

- Real problems are not linearly separable in nature.
- For example, Facies dataset or Cancer dataset (Scikit learn)

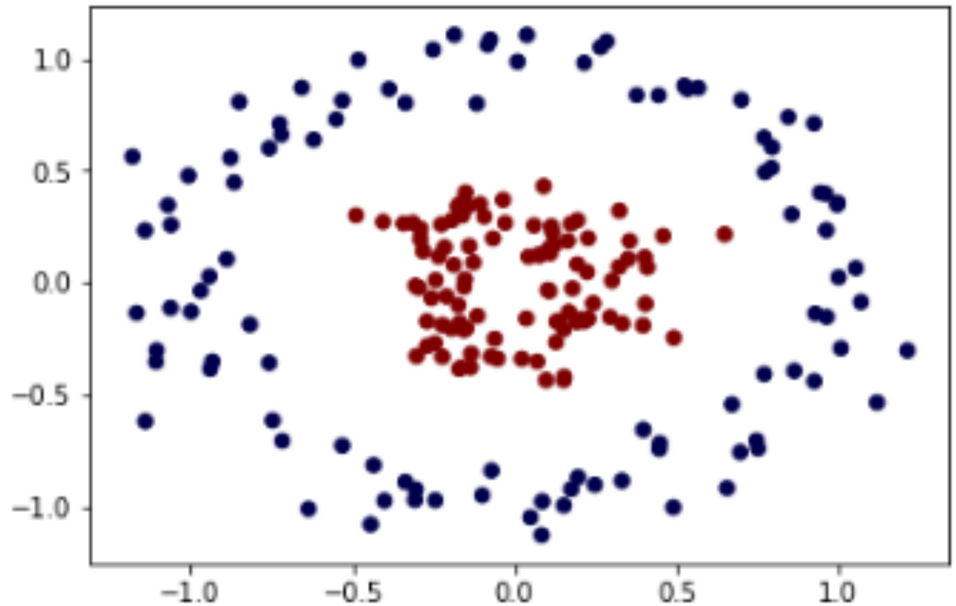




Kernel Trick

Kernel Trick

- Kernel trick is a data transformation that allow non-linearly separable data becomes separable in the another dimension.



Kernel Trick

$$maximize: \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \, y_i y_j x_i^t . x_j$$

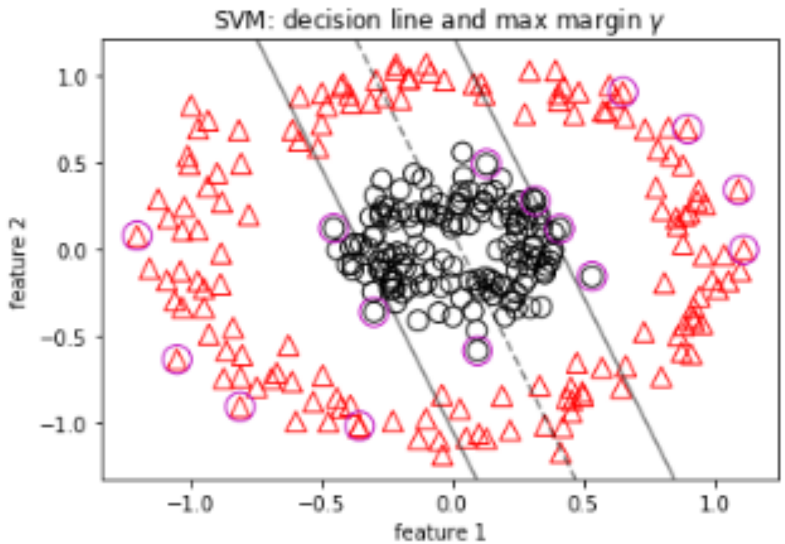
- The dot-product is targeted to transform the data.

$$K(x_i, x_j) = \phi(x_i).\phi(x_j)$$

- Common Kernels are:
  - Polynomial $(\boldsymbol{x_i}.\boldsymbol{x_j})^{\boldsymbol{n}}$
  - Radial Basis Function (RBF) $\mathbf{e}^{-\frac{|x_i - x_j|}{2\sigma^2}}$, depends on the value of sigma
  - Sigmoid-like: $\mathbf{tanh}(\mathbf{c}\mathbf{x_i^T}\mathbf{x_j} + \mathbf{h})$

Example: SVM(linear)

```
# Try linear SVM on Circles dataset
svmL= svm.SVC(kernel='linear',
              C=1e20,
              tol= 1e-3,
              max_iter = 1000,
              random_state=None).fit(Xc, yc)

# train the model on this new data
svmL.fit(Xc,yc )

plot_SVM_decisionline2(svmL, Xc, yc)
```

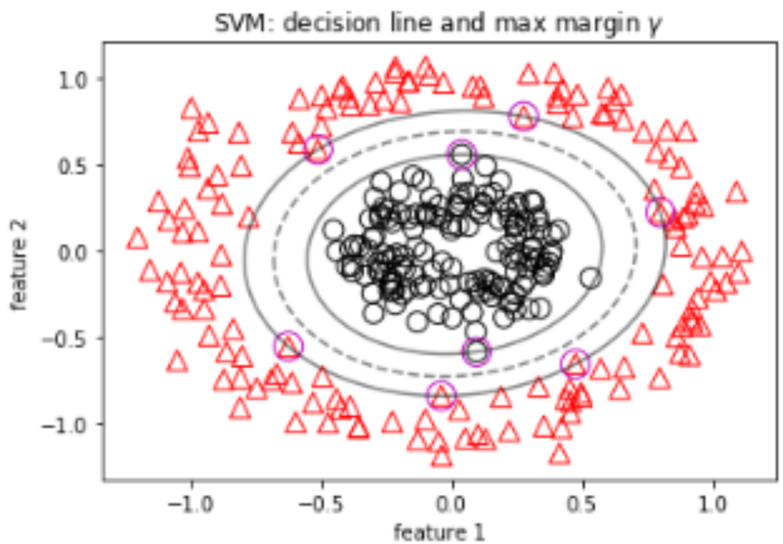


Copy


```
# Try linear SVM on Circles dataset
svmL= svm.SVC(kernel='linear',
        C=1e20,
        tol= 1e-3,
        max_iter = 1000,
        random_state=None).fit(Xc, yc)

# train the model on this new data
svmL.fit(Xc,yc )

plot_SVM_decisionline2(svmL, Xc, yc)
```

Example: SVM(linear)

```
# Try RBF SVM on Circles dataset
svmL= svm.SVC(kernel='rbf',
              gamma =0.2,
              C=1e20,
              tol= 1e-3,
              max_iter = 1000,
              random_state=None).fit(Xc, yc)

# train the model on this new data
svmL.fit(Xc,yc )

plot_SVM_decisionline2(svmL, Xc, yc)
```



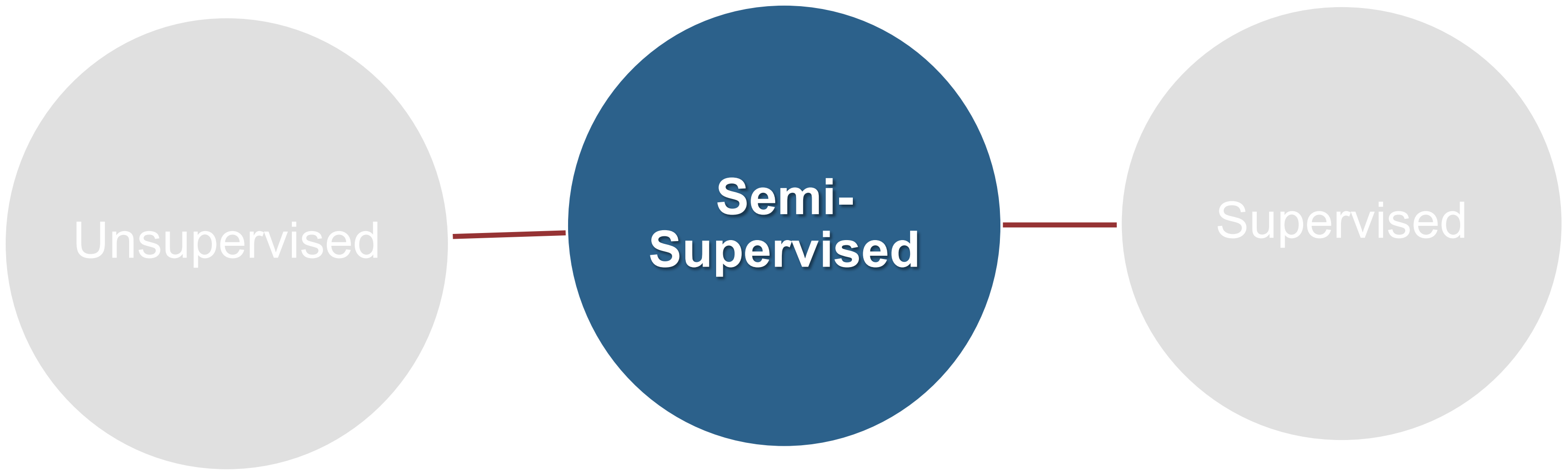
Copy


```
# Try RBF SVM on Circles dataset
svmL= svm.SVC(kernel='rbf',
        gamma =0.2,
        C=1e20,
        tol= 1e-3,
        max_iter = 1000,
        random_state=None).fit(Xc, yc)

# train the model on this new data
svmL.fit(Xc,yc )

plot_SVM_decisionline2(svmL, Xc, yc)
```

Semi-Supervised

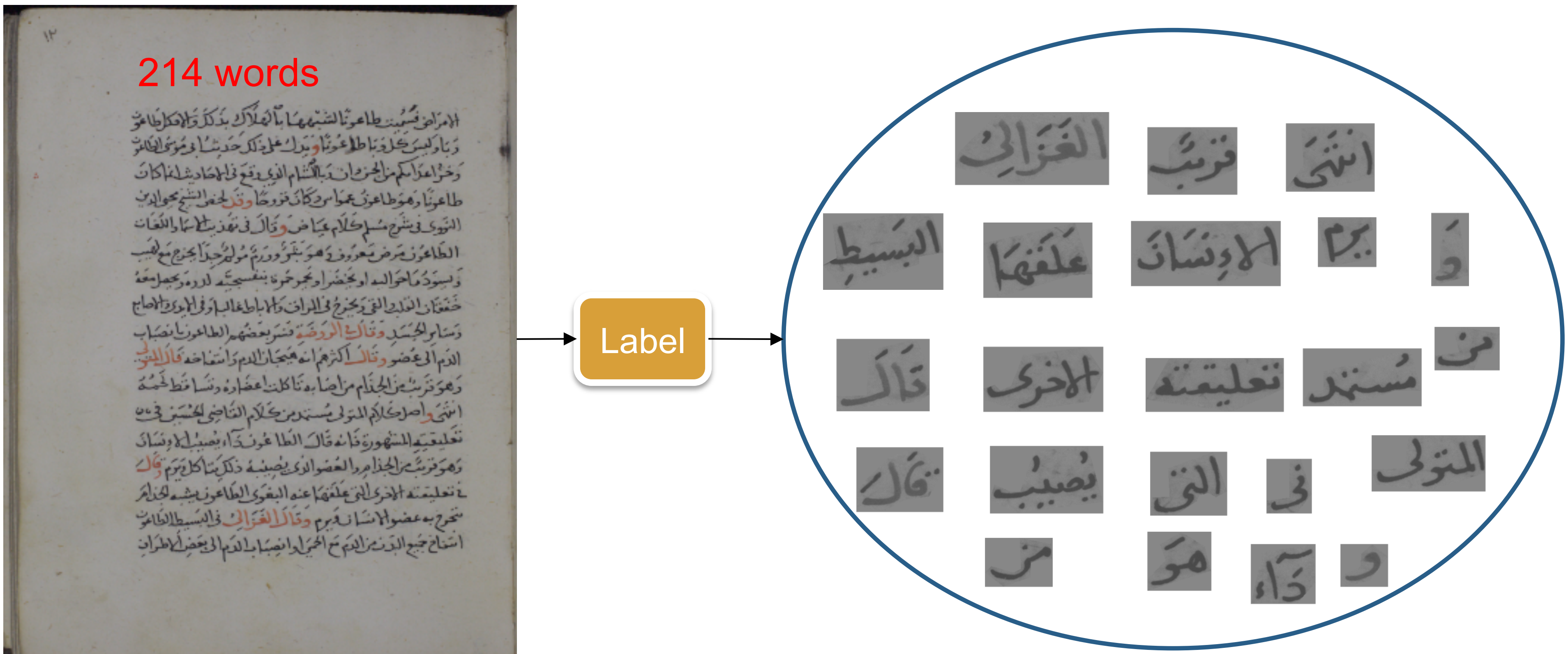
Introduction

- Semi-supervised is a middle ground between supervised and un-supervised.
- It means we have some sort of labeled and unlabeled data.
- Labeled data should help to understand the unlabeled data



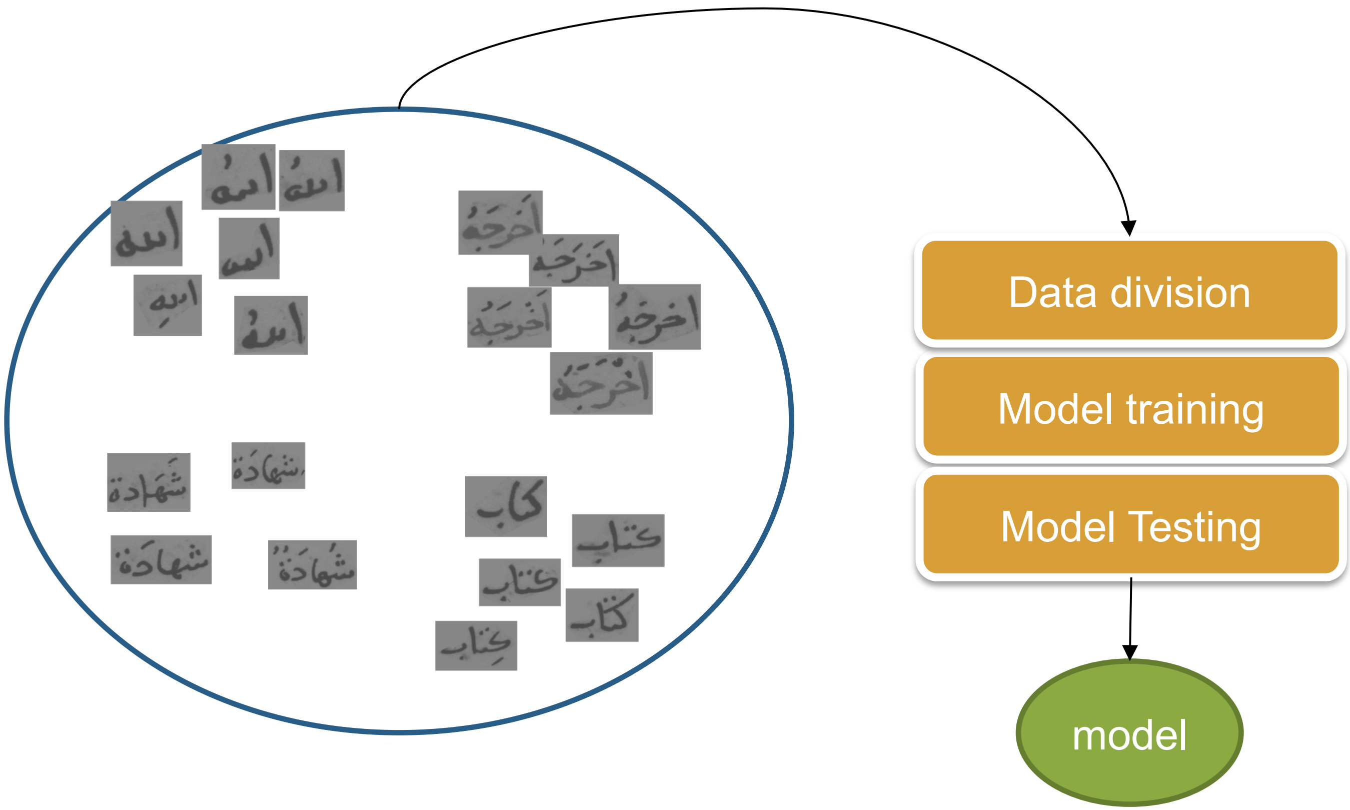
Example

- One example of semi-supervised learning is **Pseudo-labeling**.
- Suppose you have a large unlabeled dataset, and you want to label them for generating a good model for future predictions.



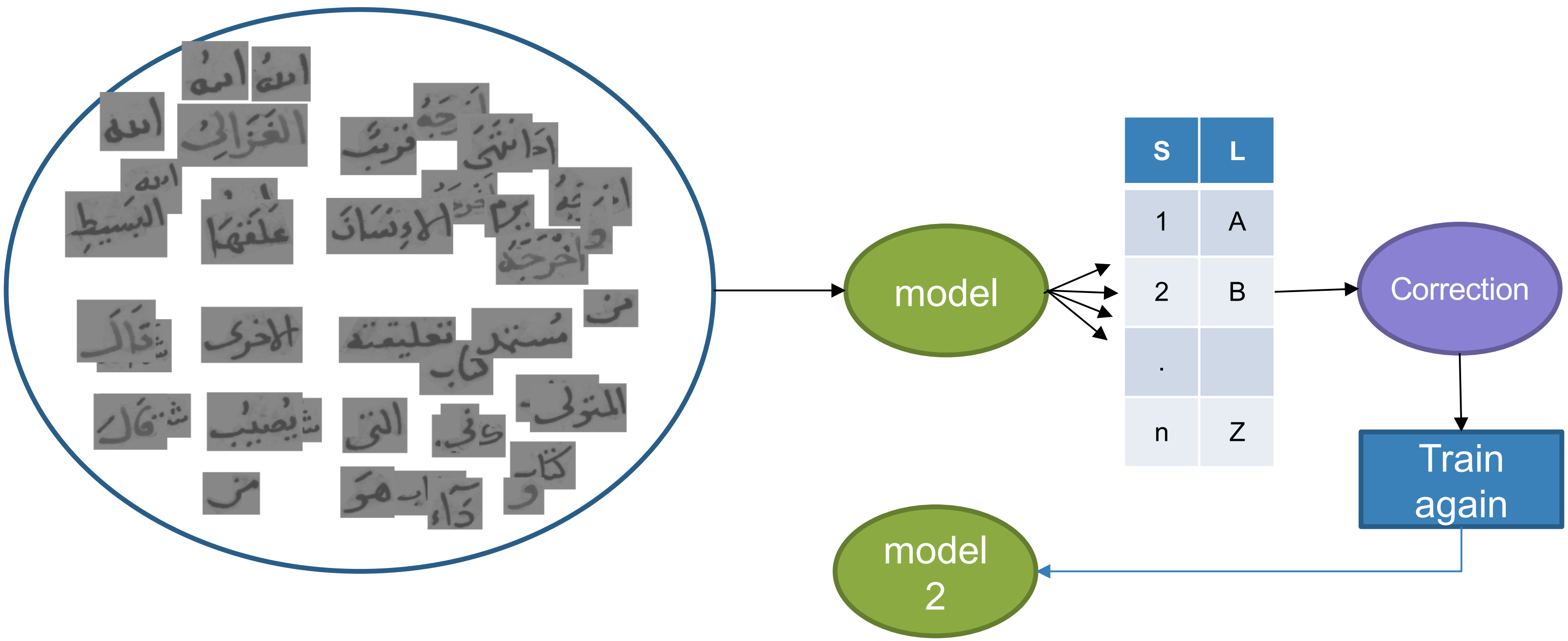
Pseudo-labeling

- Then, this labeled set is used to train and generate a machine model.



Pseudo-labeling

- Now, this model is used to label the larger dataset



Labeling

- Labeling is expensive and difficult

- Labeling is unreliable process
  - Such as Segmentation applications
  - May require experts to do that!

- Unlabeled data
  - Easy to obtain in large numbers
  - Ex. Sensors readings, text documents, images, etc.

Example

```python
# load label_propagation class
from sklearn.semi_supervised import label_propagation

# initialize a model
Prop_model = label_propagation.LabelPropagation(kernel='knn',

                                                )

Spread_model = label_propagation.LabelSpreading(kernel='rbf',
                                                gamma=0.3,
                                                )
#{'knn', 'rbf'} in case of rbf use gamma = value
```

Copy

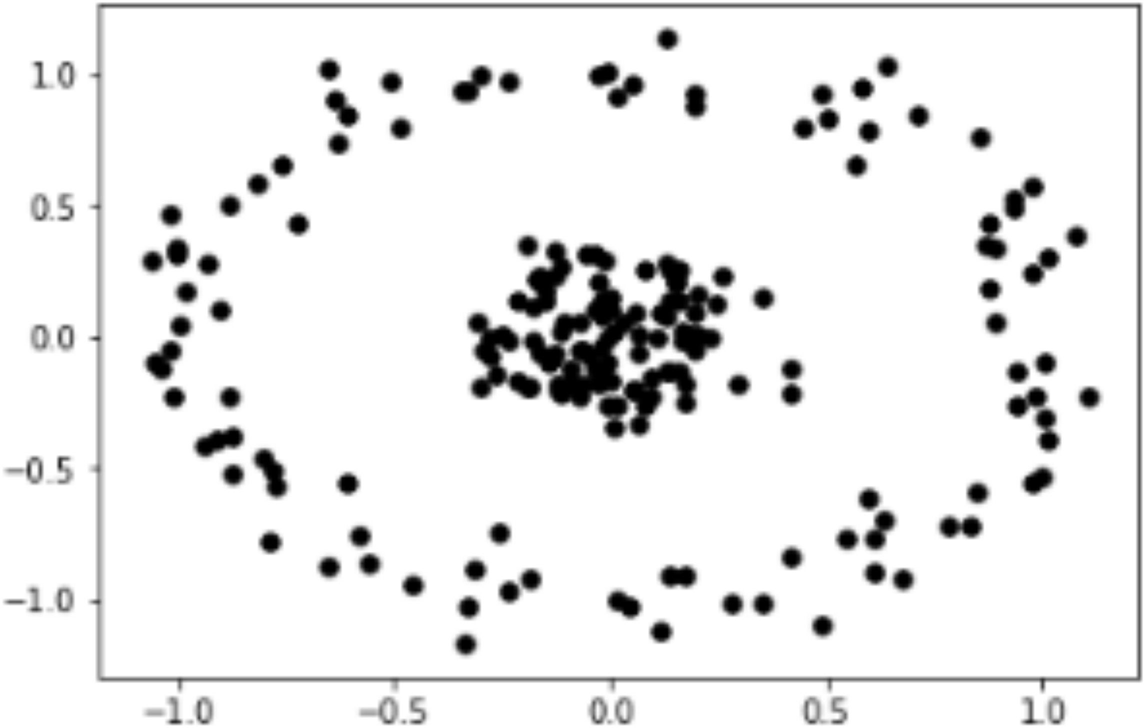
# load label_propagation class
from sklearn.semi_supervised import label_propagation

# initialize a model
Prop_model = label_propagation.LabelPropagation(kernel='knn',
)

Spread_model = label_propagation.LabelSpreading(kernel='rbf',
gamma=0.3,
)
#{'knn', 'rbf'} in case of rbf use gamma = value

Example

```python
# let us use circles dataset
Xc, yc = make_circles( 200, factor=.2, noise=.1)
plt.scatter(Xc[:,0], Xc[:,1], c='k', s=30)
plt.show()
```



Copy

```python
# let us use circles dataset
Xc, yc = make_circles( 200, factor=.3, noise=.1)
plt.scatter(Xc[:,0], Xc[:,1], c='k', s=30)
plt.show()
```

Example

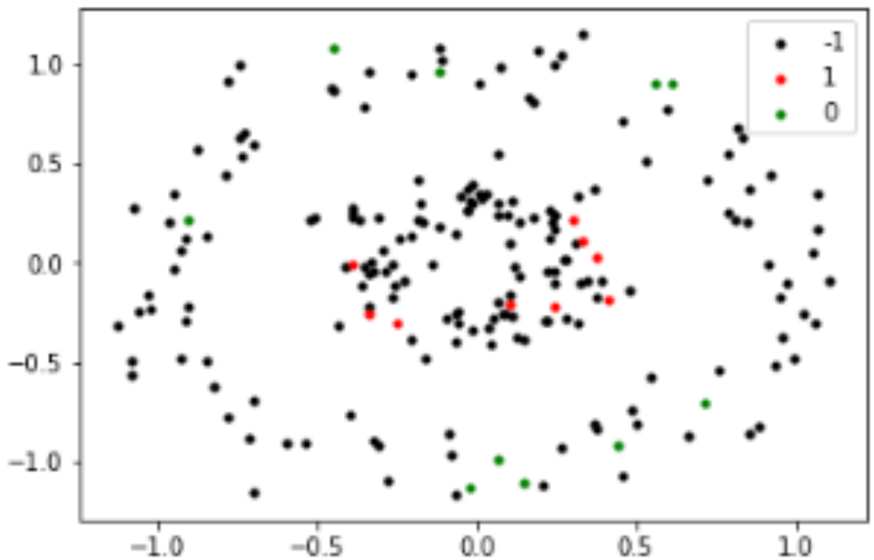
```python
# Changing labels to (unknown =-1)
rng = np.random.RandomState(42)
random_unlabeled = rng.rand(len(yc)) < 0.9
lbls = np.copy(yc)
lbls[random_unlabeled] = -1
```

Copy

```python
# Changing labels to (unknown =-1)
rng = np.random.RandomState(42)
random_unlabeled = rng.rand(len(yc)) < 0.7
lbls = np.copy(yc)
lbls[random_unlabeled] = -1
```

Example

```
plt.scatter(Xc[lbls==-1,0], Xc[lbls==-1,1], c='k', s=10,facecolors='none', label='unlabel'  )
plt.scatter(Xc[lbls==1,0], Xc[lbls==1,1], c='r', s=10,facecolors='none' , label='1')
plt.scatter(Xc[lbls==0,0], Xc[lbls==0,1], c='g', s=10,facecolors='none' , label='0' )
plt.legend(labels=['-1', '1', '0'], loc=0)
plt.show()
```

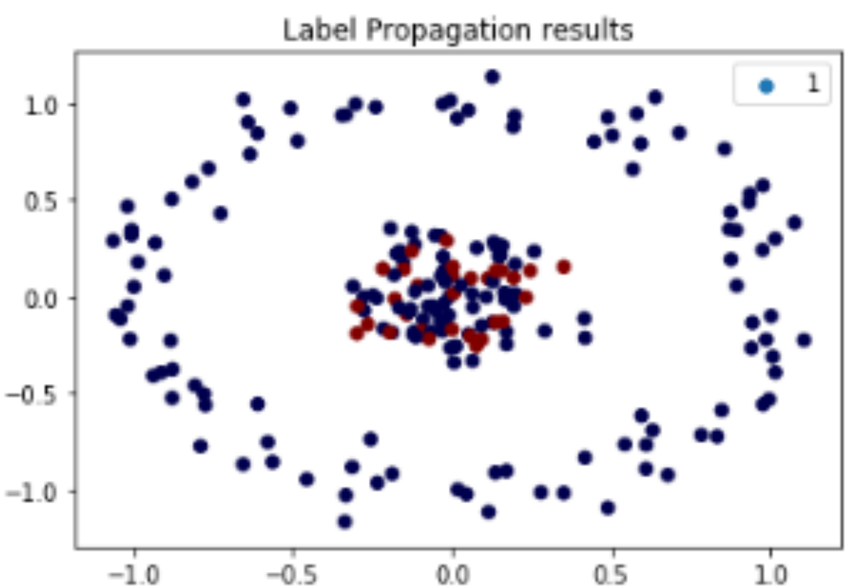


Copy


```
plt.scatter(Xc[lbls==-1,0], Xc[lbls==-1,1], c='k', s=10,facecolors='none', label='unlabel'  )
plt.scatter(Xc[lbls==1,0], Xc[lbls==1,1], c='r', s=10,facecolors='none' , label='1')
plt.scatter(Xc[lbls==0,0], Xc[lbls==0,1], c='g', s=10,facecolors='none' , label='0' )
plt.legend(labels=['-1', '1', '0'], loc=0)
plt.show()
```

Example

```
Prop_model.fit(Xc, lbls)
prop_labels = Prop_model.transduction_
plt.scatter(Xc[:,0], Xc[:,1], c=prop_labels, s=30,facecolors='none',  cmap='seismic')
plt.legend(labels=['1', '0'], loc=0)
plt.title('Label Propagation results')
plt.show()
```



Copy


```
Prop_model.fit(Xc, lbls)
prop_labels = Prop_model.transduction_
plt.scatter(Xc[:,0], Xc[:,1], c=prop_labels, s=30,facecolors='none',  cmap='seismic')
plt.legend(labels=['1', '0'], loc=0)
plt.title('Label Propagation results')
plt.show()
```

Thank you