Mahidol University
Faculty of Medicine Ramathibodi Hospital
Department of Clinical Epidemiology and Biostatistics

Semi-supervised learning

RADI608: Data Mining and Machine Learning
RADI602: Data Mining and Knowledge Discovery

Lect. Anuchate Pattanateepapon, D.Eng.

Section of Data Science for Healthcare

Department of Clinical Epidemiology and Biostatistics

Faculty of Medicine Ramathibodi Hospital, Mahidol University

Guideline

- What's a semi-supervised learning (SSL)?
- Examples of a semi-supervised learning
- Decision boundary of label propagation versus SVM on the Iris dataset by using Python
- How to apply semi-supervised learning idea in imbalanced dataset management

What's a semi-supervised learning (SSL)?

- a semi-supervised learning is falls into both a supervised learning and an un-supervised learning
- apply when a large amount of training dataset is unlabeled or unknown labeled

D17390	M33210	H18490	H40891	R77780	T49647	Class	
-1.1427	-1.11E+00	-0.52039	-3.15E-01	-1.76419	-2.75E+00	NA	unknown
-0.68703	-7.77E-01	-1.09346	-1.21E+00	-1.06206	-2.13E+00	NA	unknown
-1.09986	-1.31E-01	-2.74807	-1.01E+00	-2.26003	-1.50E+00	1	
-0.46549	-4.55E-01	-1.80478	-1.61E+00	-1.22345	-1.07E+00	NA	unknown
-0.89	-5.28E-01	-1.63602	-1.21E+00	-1.23269	-1.62E+00	1	
-1.01358	-4.95E-01	-0.5794	-3.58E-01	-1.5867	-1.86E+00	-1	
-0.75347	-1.57E+00	-1.35457	-1.33E+00	-1.19646	-1.74E+00	NA	unknown
-0.65955	-1.85E+00	-2.07593	-1.53E+00	-2.14439	-9.22E-01	-1	
-0.49417	-1.10E+00	-0.87457	-9.17E-01	-0.81782	-3.17E-01	NA	unknown
-0.66509	-2.46E-01	-1.55021	-1.36E+00	-1.17008	-1.26E+00	NA	unknown

SSL is suitable for datasets with:

- Small amounts of labeled data
- Large amounts of unlabeled data (labeling requires effort)
- High certainty in labels

SSL is suitable for datasets with:

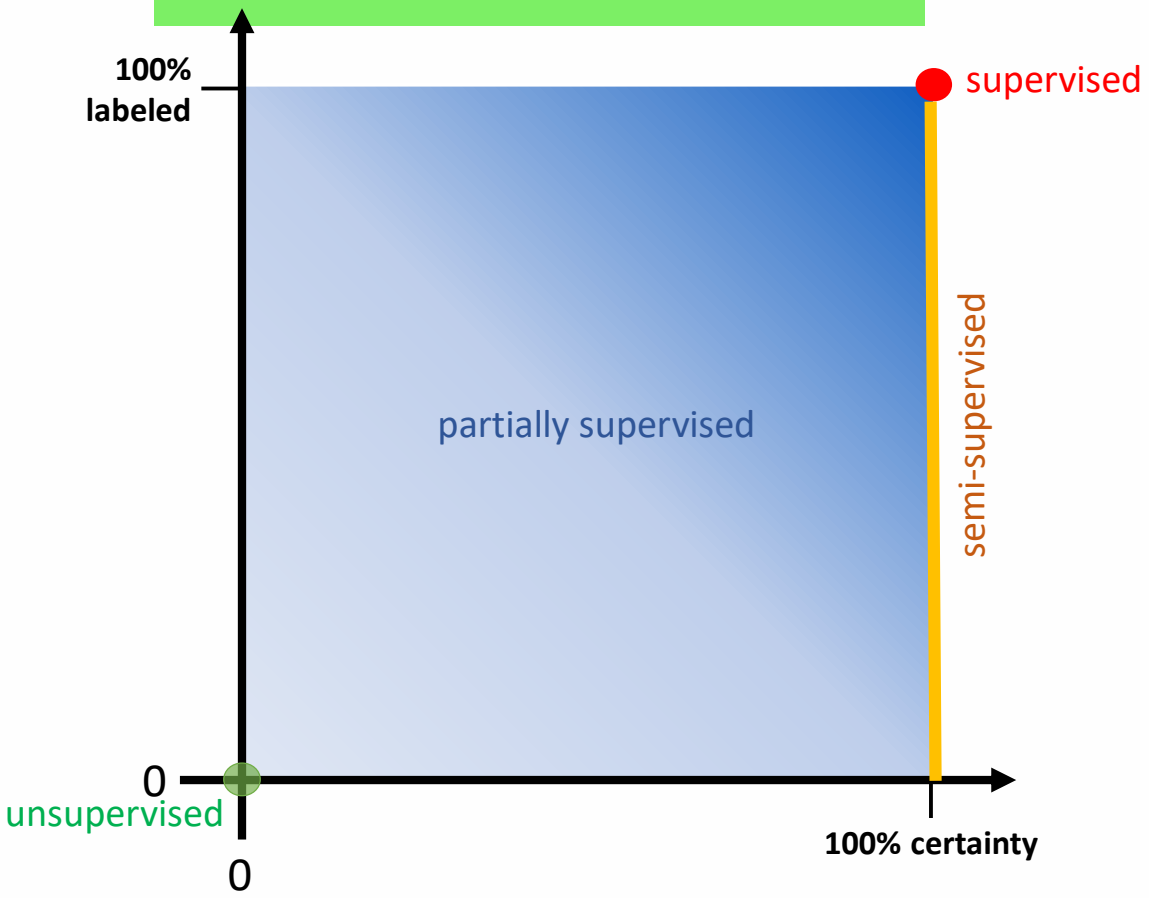
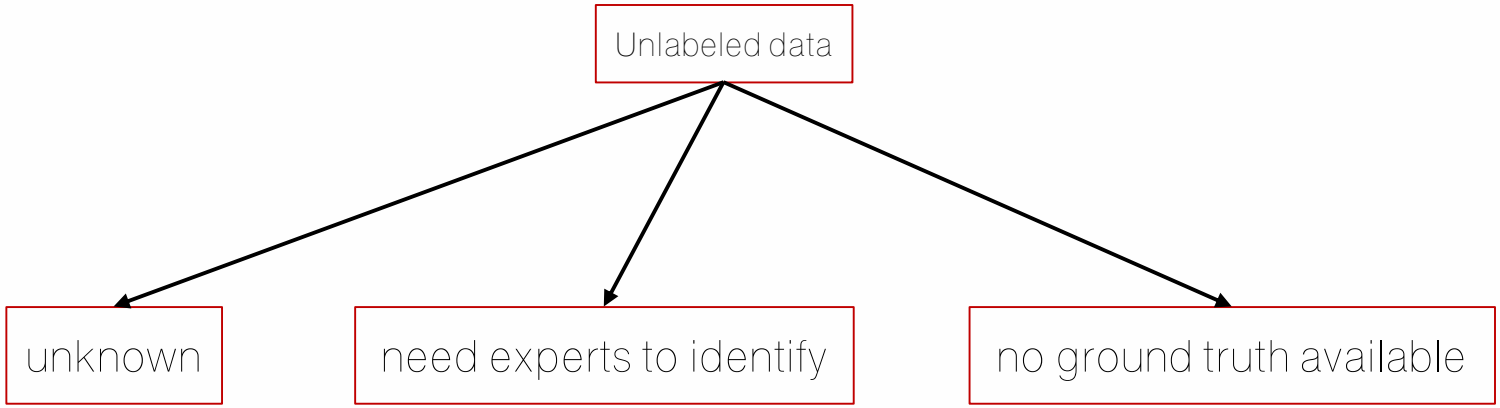


Image: (Biecek, 2012)

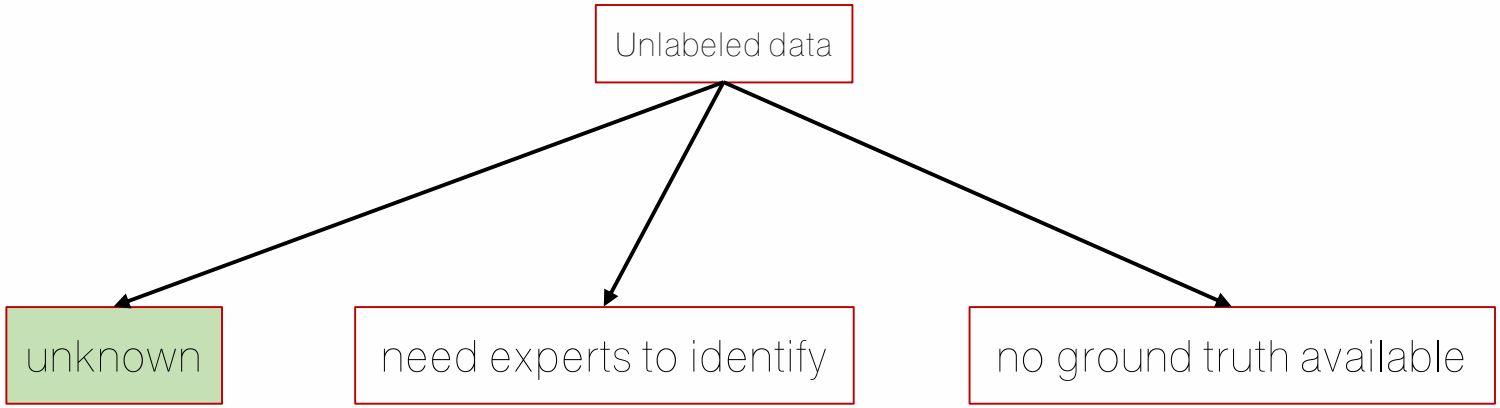
SSL limitation

- SSL is not always guaranteed to work!
- Performance may also degrade due to noisy instances.

Why the healthcare data are incomplete labeled?

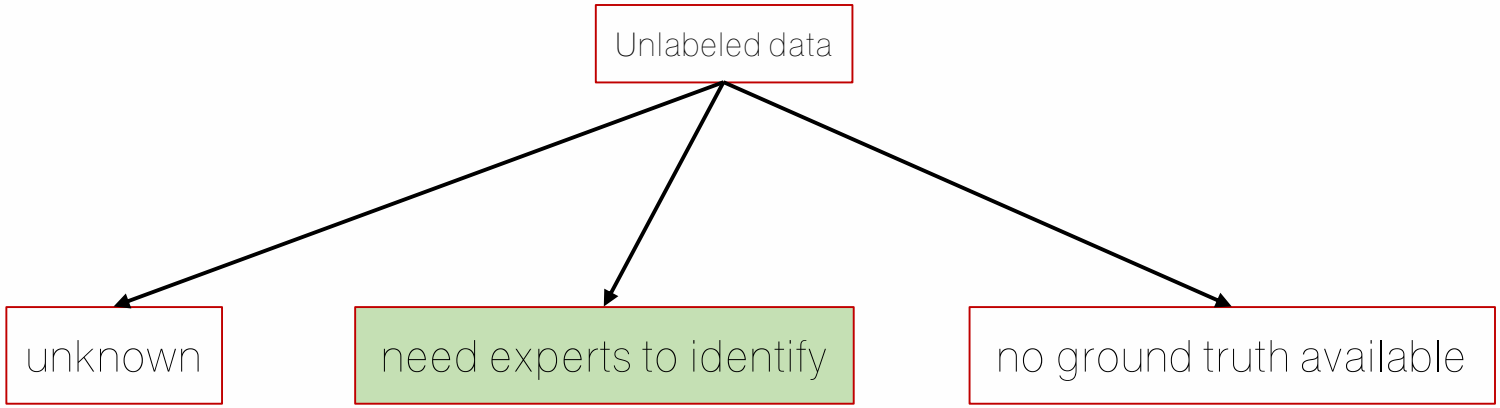


Why the healthcare data are incomplete labeled?



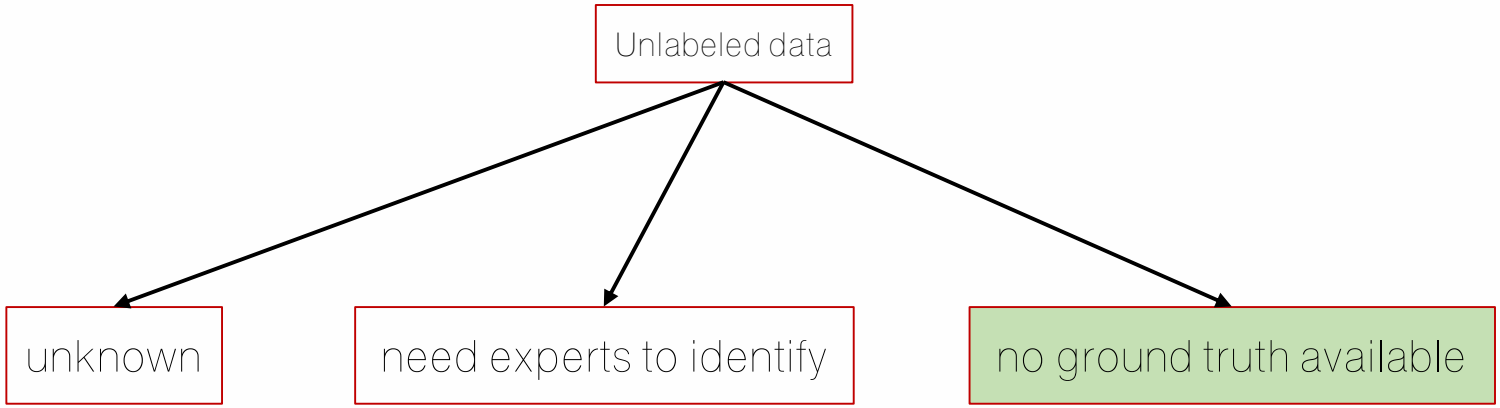
From previous cases, the investigating targets are not recorded when collecting the data, and we have to inspect them for a current study.

Why the healthcare data are incomplete labeled?



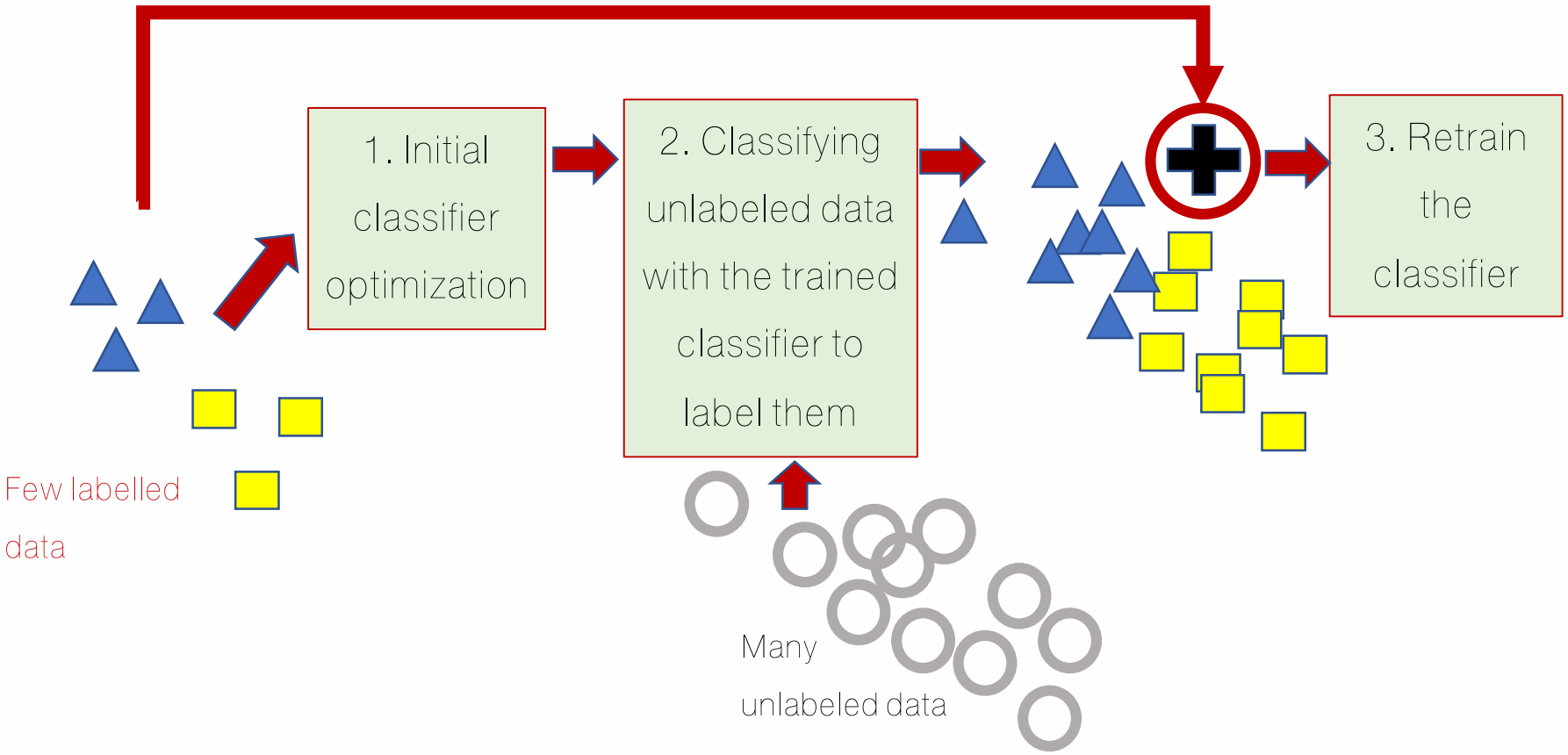
It requires domain experts to label or confirm a target, which would increase the cost and duration of the data collection.

Why the healthcare data are incomplete labeled?



"alive" cases varied from generally healthy to very ill and anywhere in between, then there is no ground truth available for the healthy cases

A basic semi-supervised learning process



Examples of a semi-supervised learning

- Semi-Supervised Text Categorization using Recursive K-means clustering (Harsha S Gowda, Mahamad Suhil, D S Guru and Lavanya Narayana Raju, 2017)
- Transductive support vector machines and applications in bioinformatics for promoter recognition (*N. Kasabov and Shaoning Pang*, 2003)
- Learning from Labeled and Unlabeled Data with Label Propagation (Z. Xiaojin, G. Zoubin, 2002)

Semi-Supervised Text Categorization using Recursive K-means clustering

Semi-Supervised Text Categorization using Recursive K-means clustering

Harsha S Gowda, Mahamad Suhil, D S Guru and Lavanya Narayana Raju

Department of Studies in Computer Science, University of Mysore, Mysore, India.
harshasgmysore@gmail.com, mahamad45@yahoo.co.in,
dsg@compsci.uni-mysore.ac.in and swaralavz@gmail.com

Abstract. In this paper, we present a semi-supervised learning algorithm for classification of text documents. A method of labeling unlabeled text documents is presented. The presented method is based on the principle of divide and conquer strategy. It uses recursive *K*-means algorithm for partitioning both labeled and unlabeled data collection. The *K*-means algorithm is applied recursively on each partition till a desired level partition is achieved such that each partition contains labeled documents of a single class. Once the desired clusters are obtained, the respective cluster centroids are considered as representatives of the clusters and the nearest neighbor rule is used for classifying an unknown text document. Series of experiments have been conducted to bring out the superiority of the proposed model over other recent state of the art models on 20Newsgroups dataset.

Semi-Supervised Text Categorization using Recursive K-means clustering

K-means clustering	*Recursive K-means clustering*
Train a model by using unlabeled data	*Train a model by using unlabeled and labeled data*
One-time clustering	*Recursive clustering*

Recursive K-means clustering

- Recursive K-means algorithm is partitioning both labeled and unlabeled data.
- Applied the K-means algorithm recursively on each partition till a desired level partition is achieved such that each partition contains labeled documents of a single class.
- Once the desired clusters are obtained, the respective cluster centroids are considered as representatives of the clusters and the nearest neighbor rule is used for classifying an unknown text document

Recursive K-means clustering

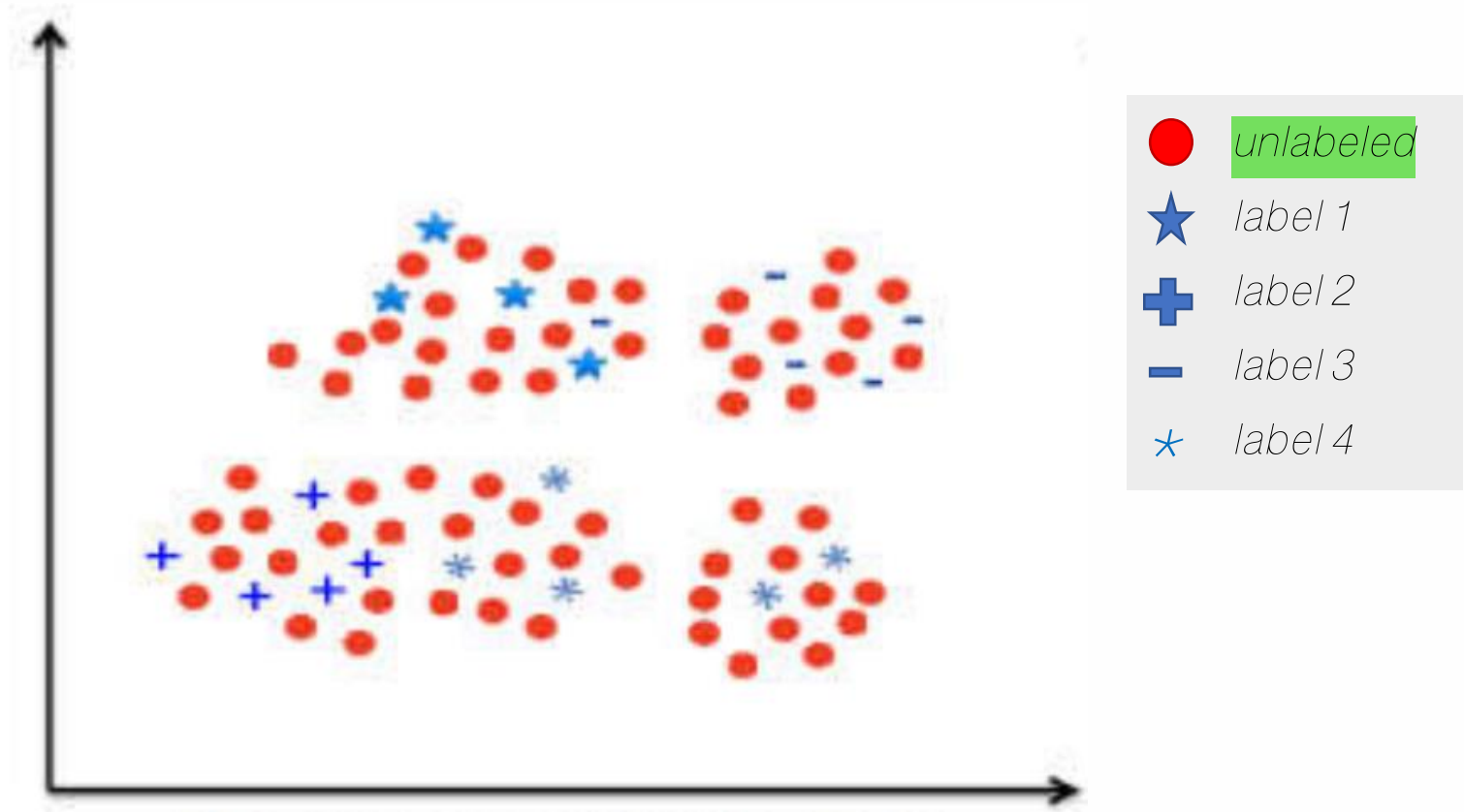
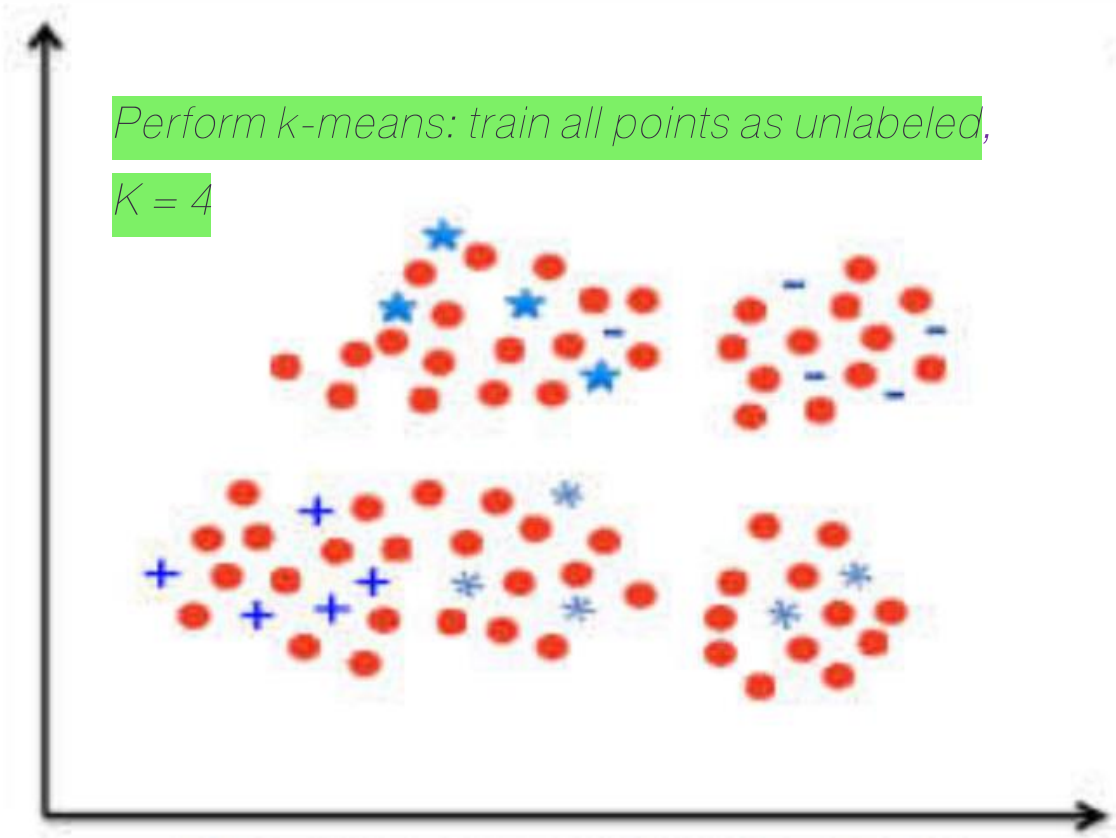


Illustration 1 : Feature space with samples

Recursive K-means clustering

Perform k-means: train all points as unlabeled,

K = 4



unlabeled
label 1
label 2
label 3
label 4

Illustration 1 : Feature space with samples

Recursive K-means clustering

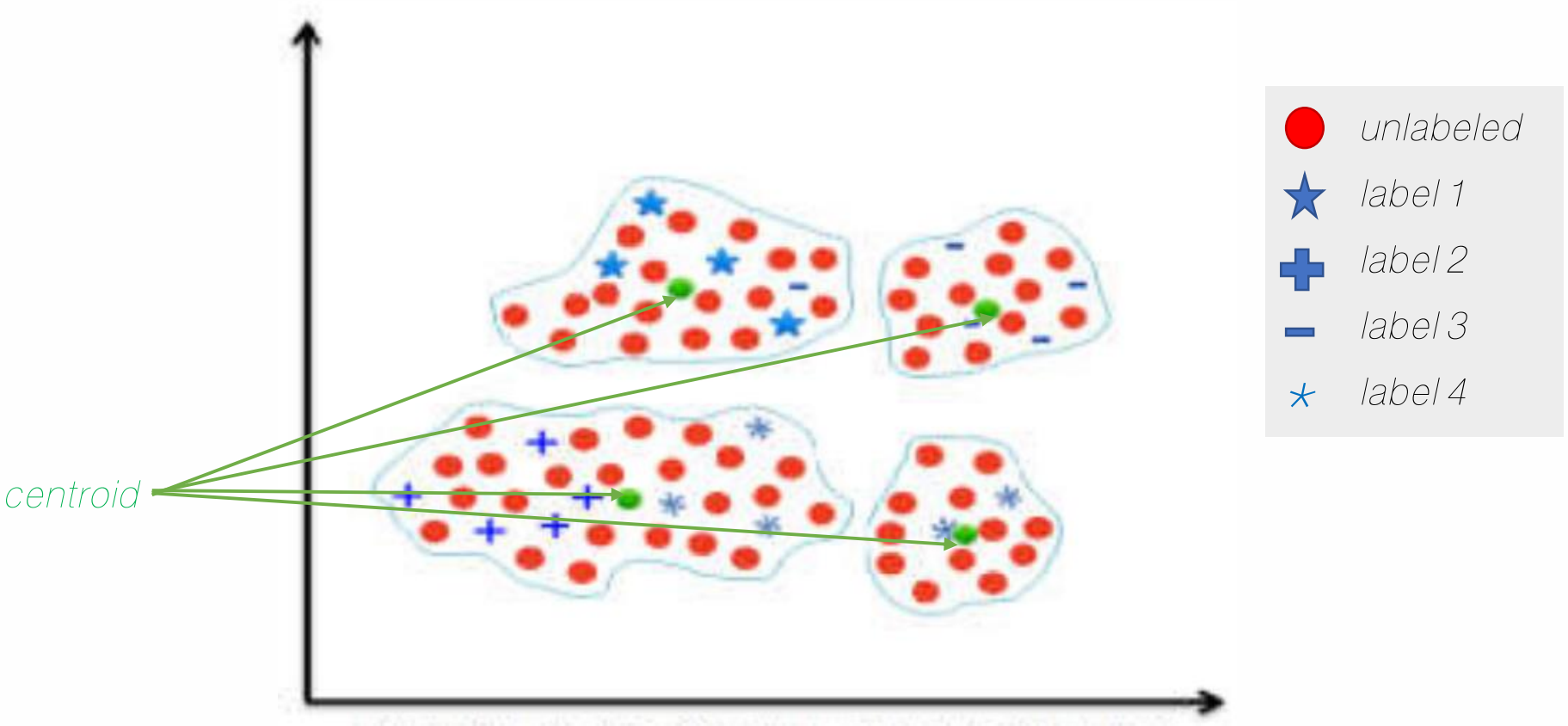


Illustration 2 : After k-means clustering round 1

Recursive K-means clustering

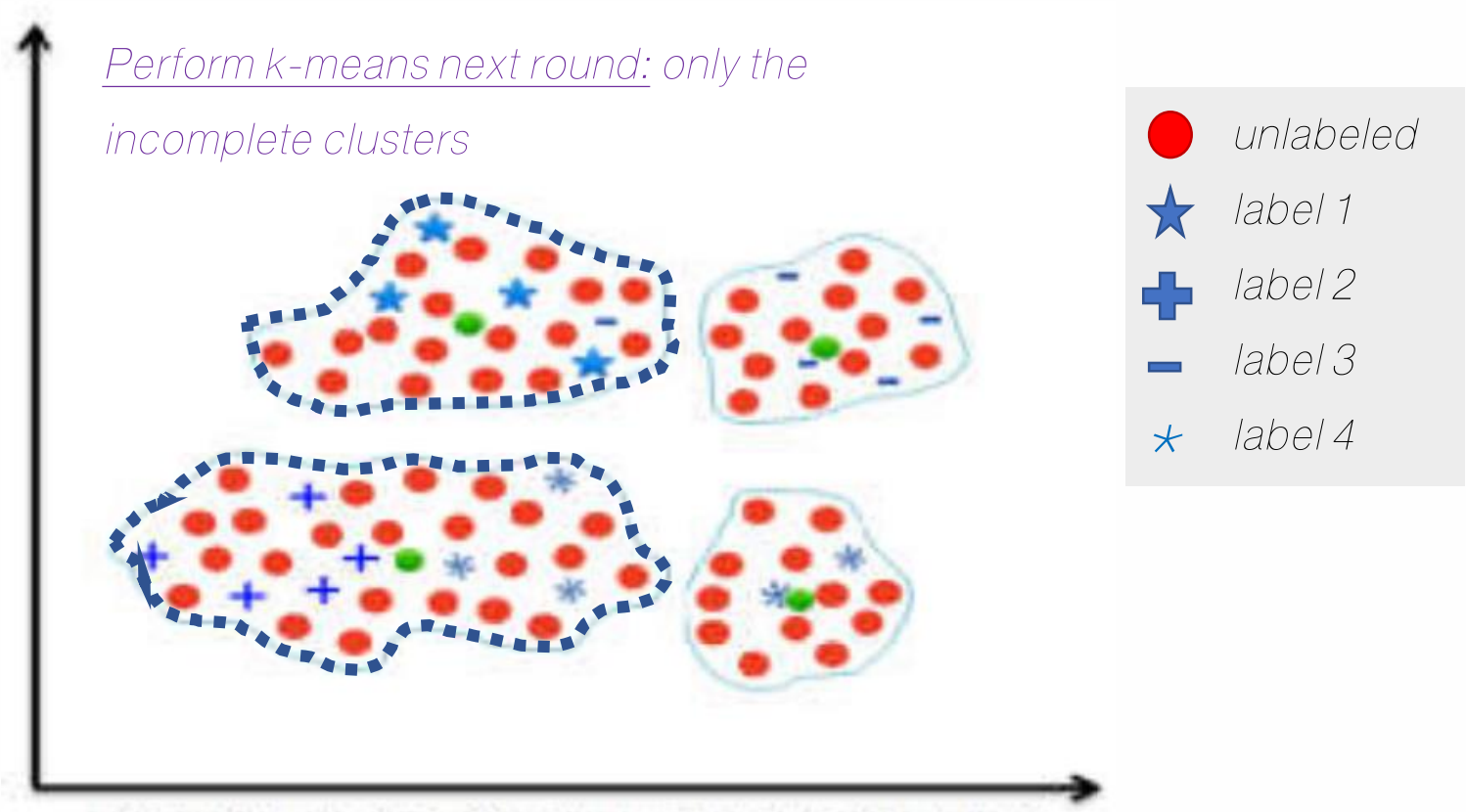


Illustration 2 : After k-means clustering round 1

Recursive K-means clustering

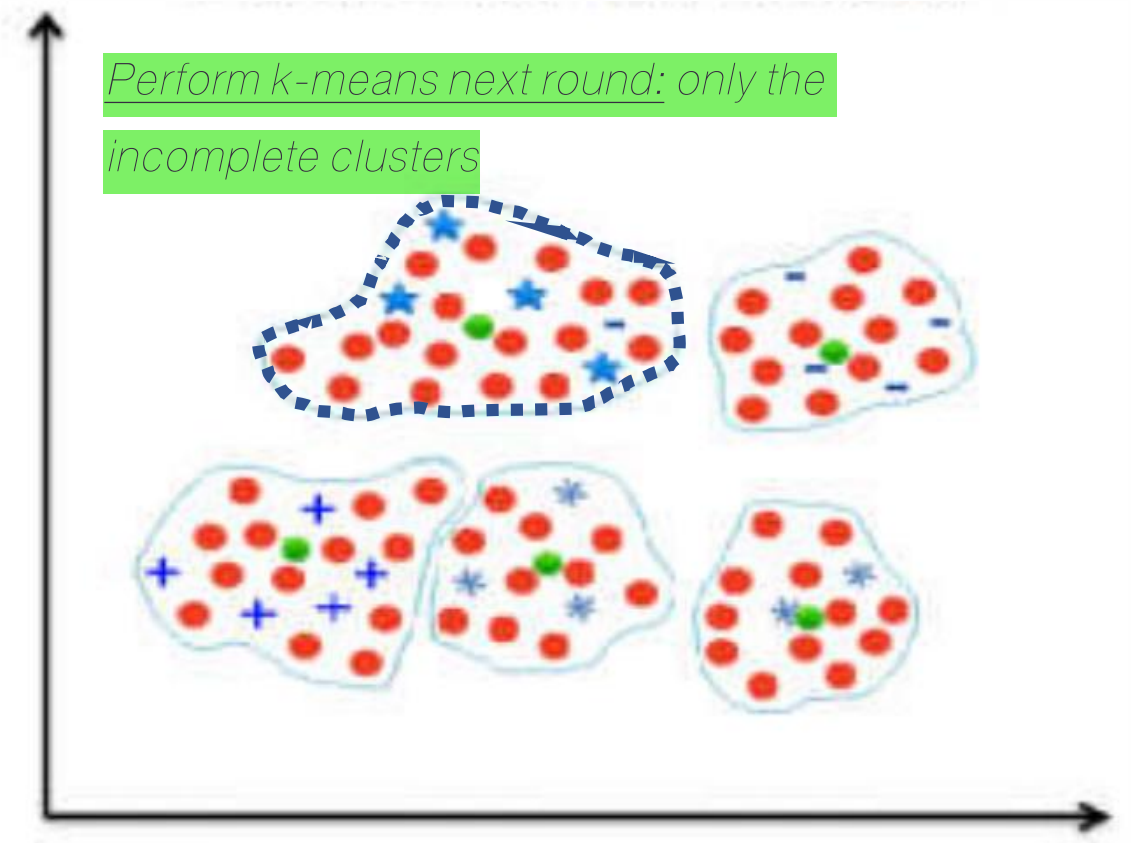


Illustration 3 : Recursive k-means clustering round 2

Recursive K-means clustering

Stop recursive K-means clustering: *centroid not change or archived level partition*

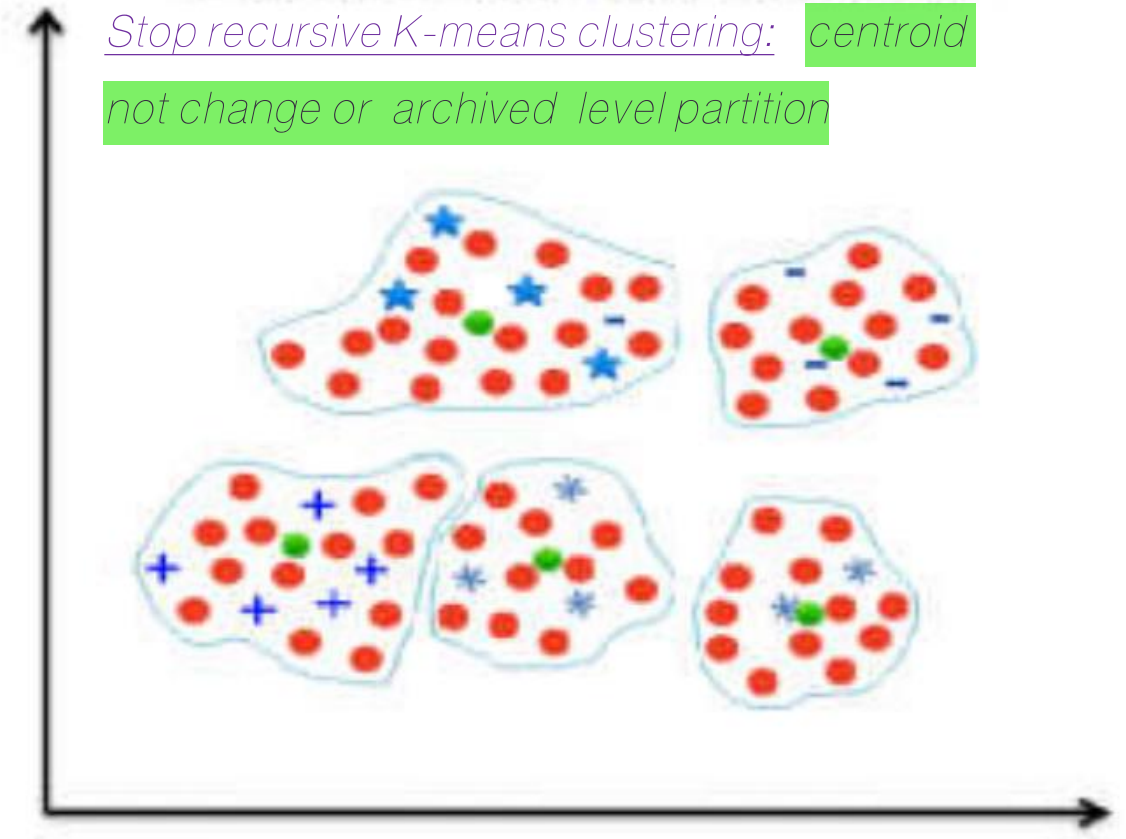


Illustration 4: Recursive k-means clustering round 3

Recursive K-means clustering

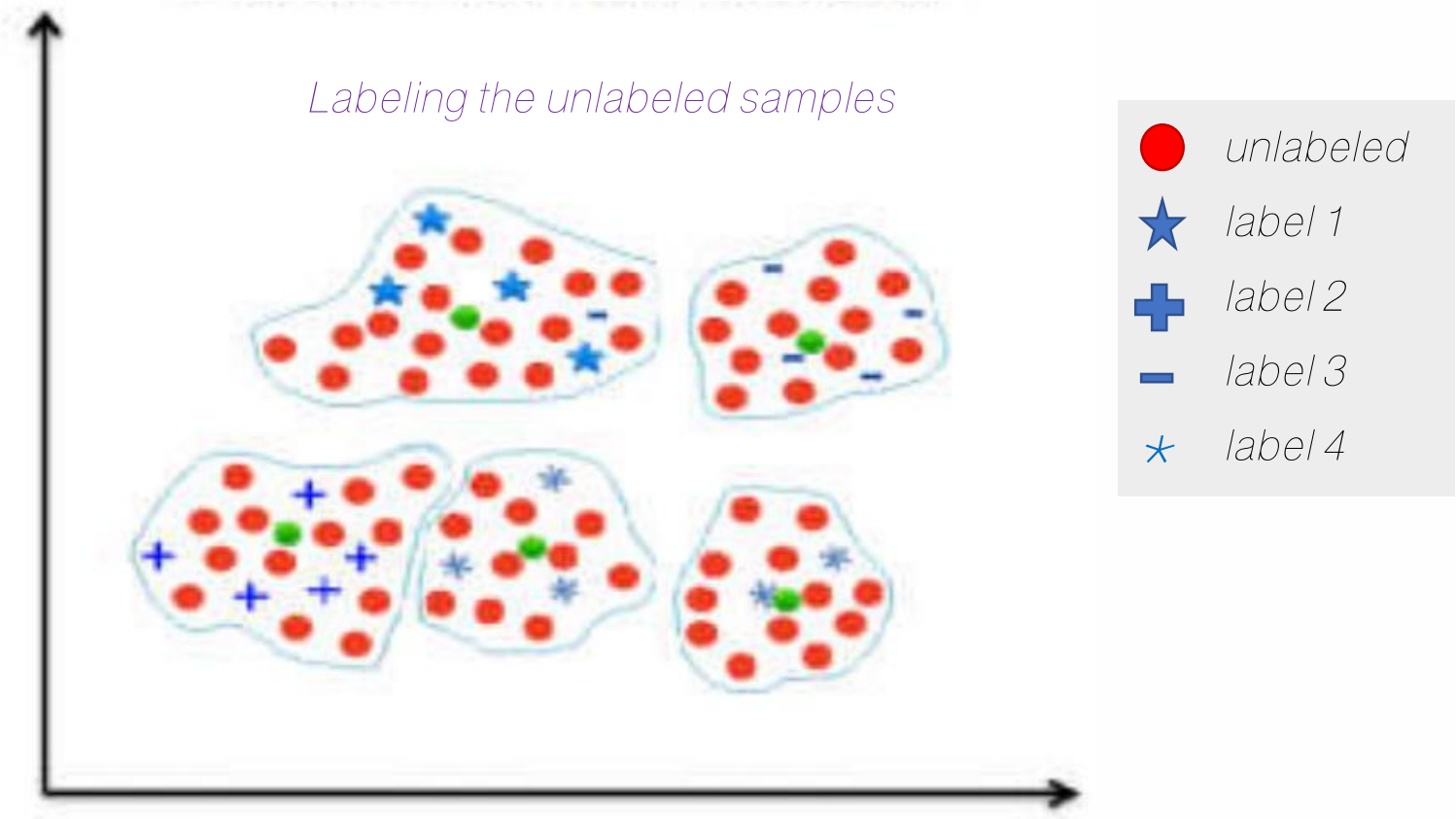


Illustration 4: Recursive k-means clustering round 3

Recursive K-means clustering

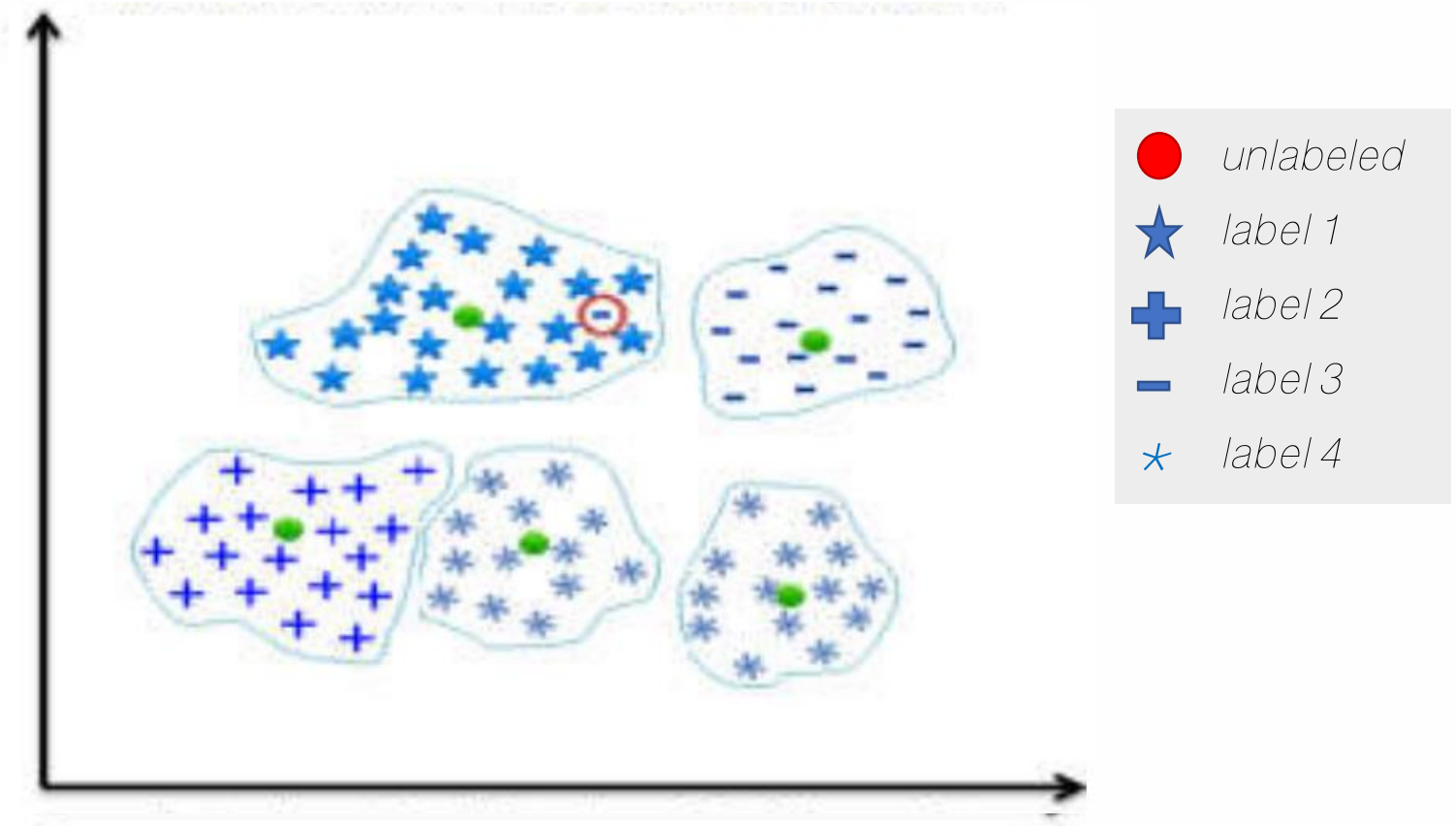


Illustration 5 : Labeling the unlabeled samples

Transductive support vector machines and applications in bioinformatics for promoter recognition

IEEE Int. Conf. Neural Networks & Signal Processing
Nanjing, China, December 14-17, 2003

TRANSDUCTIVE SUPPORT VECTOR MACHINES AND APPLICATIONS IN BIOINFOMATICS FOR PROMOTER RECOGNITION

Nikola Kasabov, Shaoning Pang

Knowledge Engineering & Discover Research Institute
Auckland University of Technology, Private Bag 92006, Auckland 1020, New Zealand

ABSTRACT

This paper introduces a novel Transductive Support Vector Machine (TSVM) model and compares it with the traditional inductive SVM on a key problem in Bioinformatics - promoter recognition. While inductive reasoning is concerned with the development of a model (a function) to approximate data from the whole problem space (induction), and consecutively using this model to predict output values for a new input vector (deduction), in the transductive inference systems a model is developed for every new input vector based on some closest to the new vector data from an existing database and this model is used to predict only the output for this vector. The TSVM outperforms by far the inductive SVM models applied on the same problems. Analysis is given on the advantages and disadvantages of the TSVM. Hybrid TSVM-evolving connections systems are discussed as directions for future research.

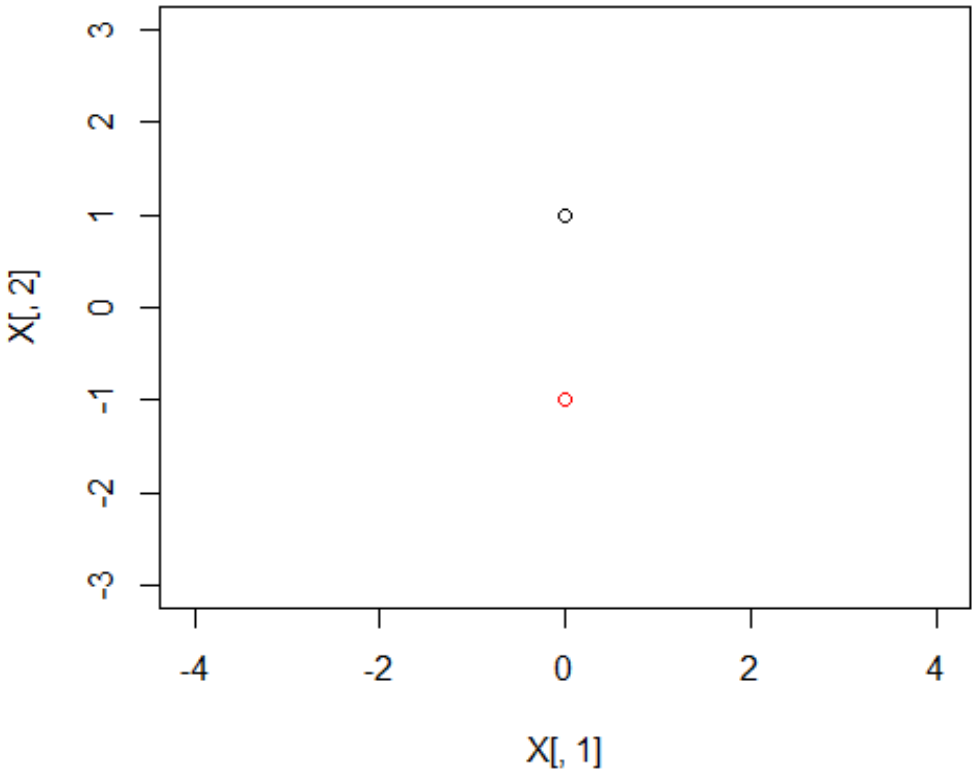
on the activation of one or several neighboring local models (rules). The inductive learning and inference approach is useful when a global model ("the big picture") of the problem is needed even in its very approximate form, when incremental, on-line learning is applied to adjust this model on new data and trace its evolution.

Generally speaking, inductive inference is concerned with the estimation of a function (a model) based on data from the whole problem space and using this model to predict output values for a new input vector, which can be any point in this space (deduction) - Fig.1. Most of the statistical, connectionist and fuzzy learning methods, such as SVM, MLP; RBF; ANFIS (see [10]; [16]) and ECOS [15] that include DENFIS [17], EFuNN [18, 19] and many more, have been developed and tested on inductive reasoning problems.

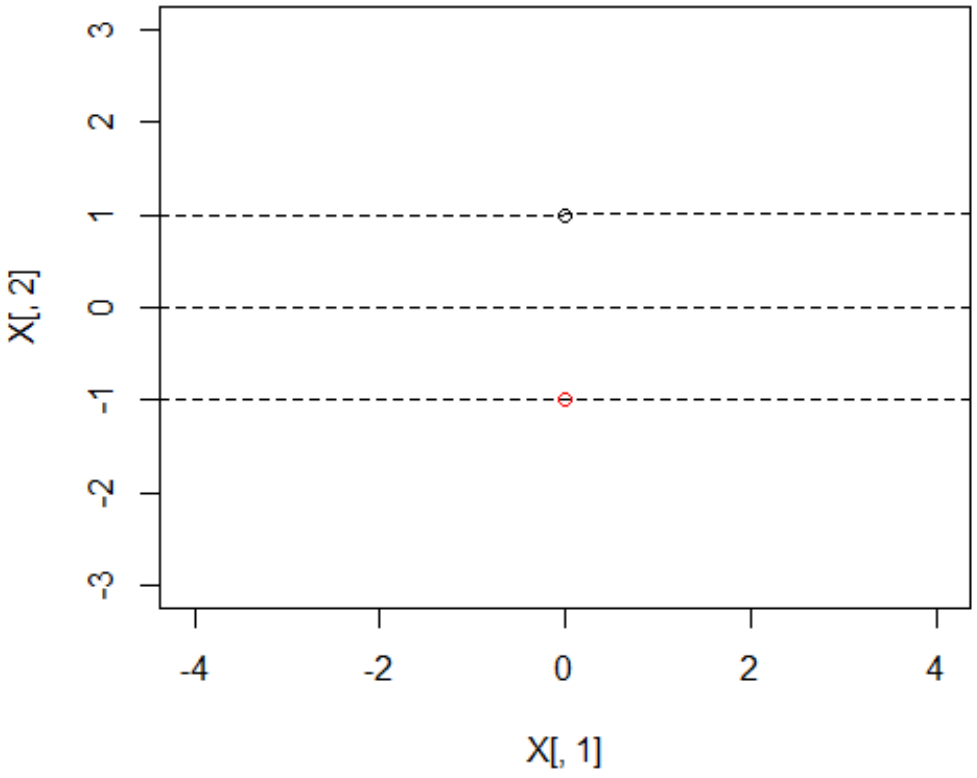
Transductive support vector machines and applications in bioinformatics for promoter recognition

Inductive SVM or basic SVM	*Transductive SVM*
The SVM is concerned with the development of a model (a function) to approximate data from the whole problem space (induction), and consecutively using this model to predict output values for a new input vector (deduction)	*in theTSVM, a model is developed for every new input vector based on some closest to the new vector data from an existing database and this model is used to predict only the output for this vector*

Transductive support vector machines



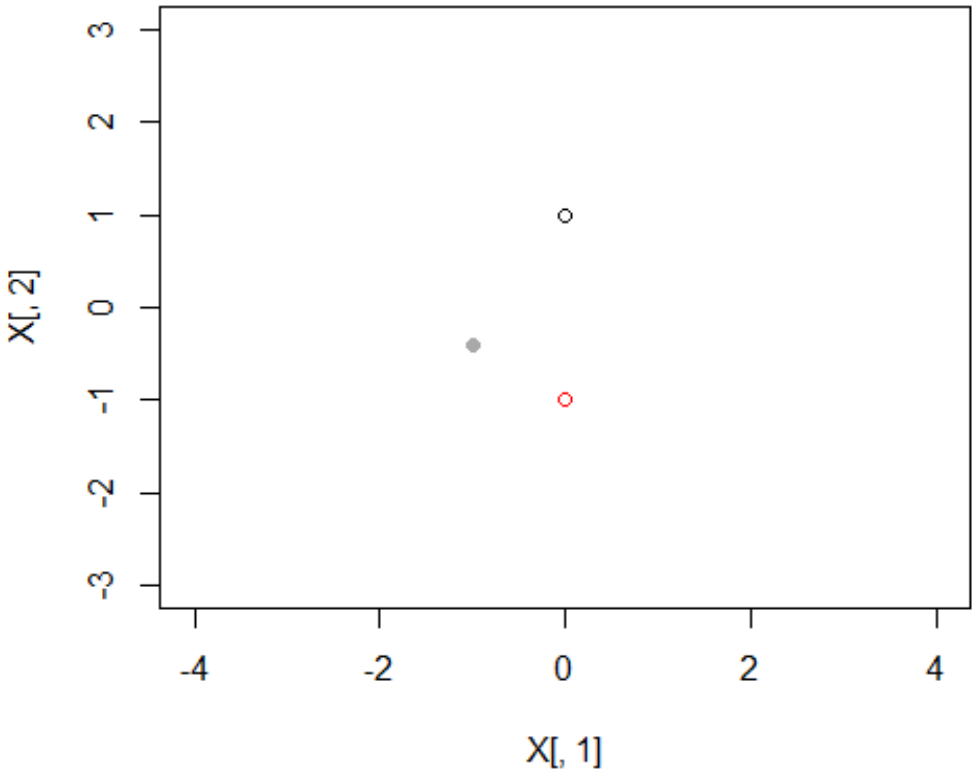
Transductive support vector machines



SVM

Transductive support vector machines

A balancing constraint enfored that causes the fraction of objects assigned to each label in the unlabeled data to be similar to the label fraction in the labeled data

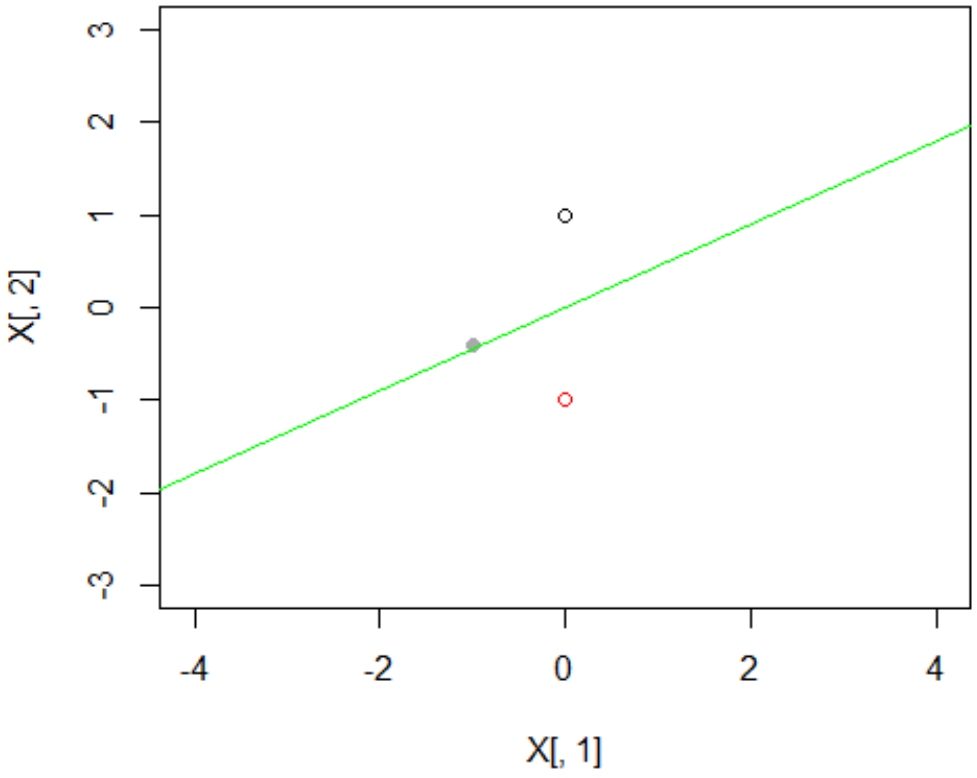


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

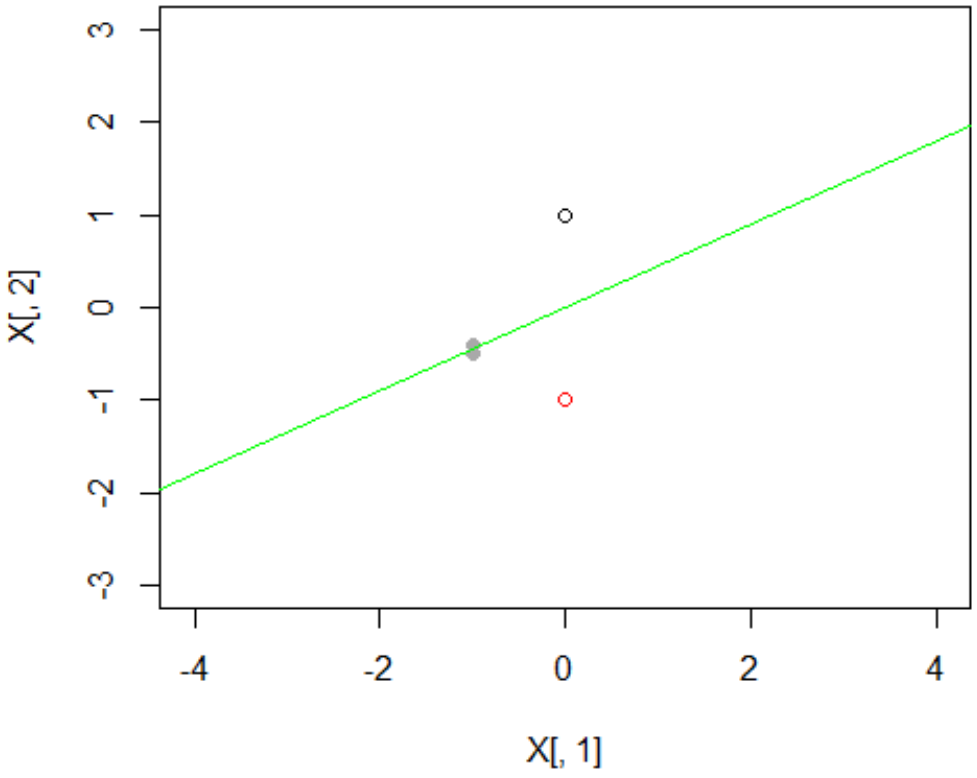


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

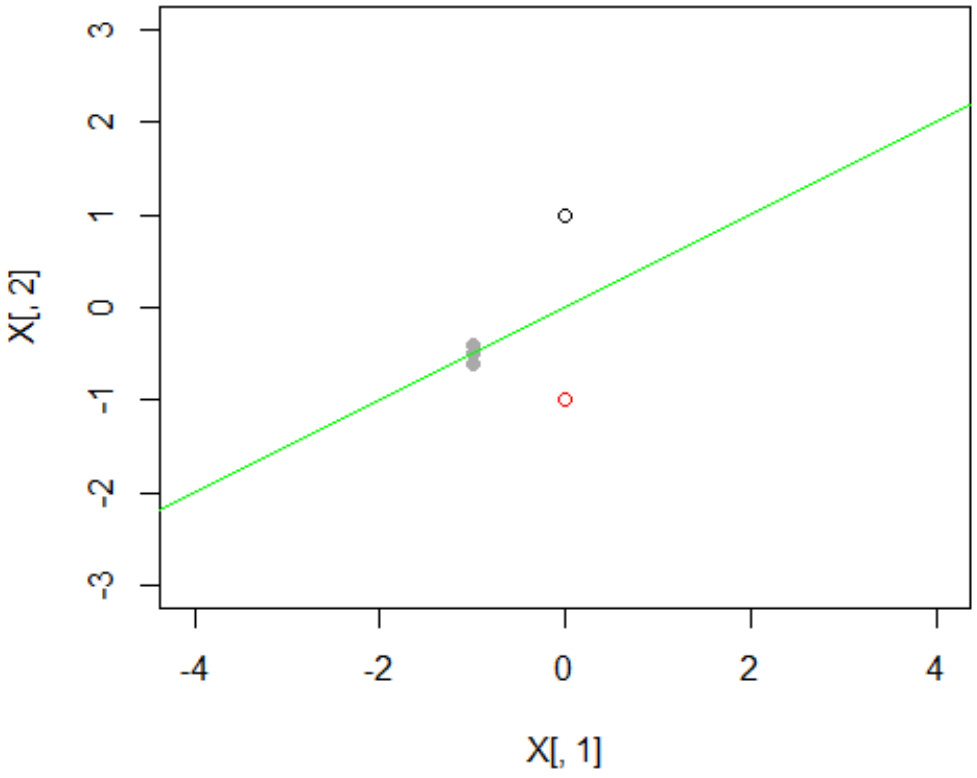


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

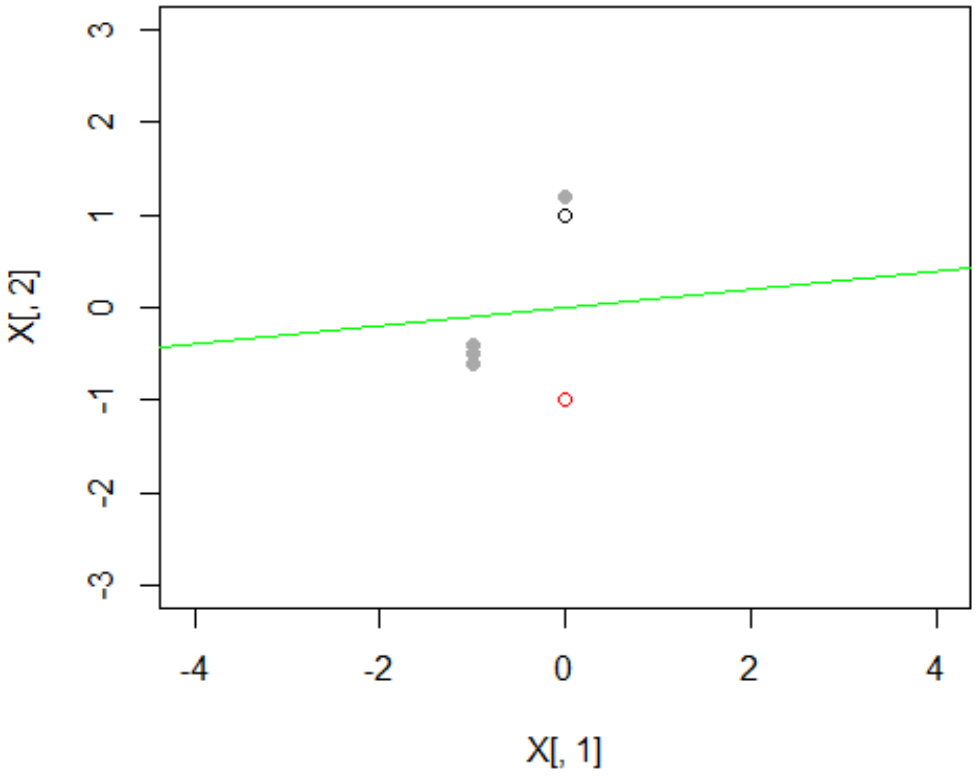


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

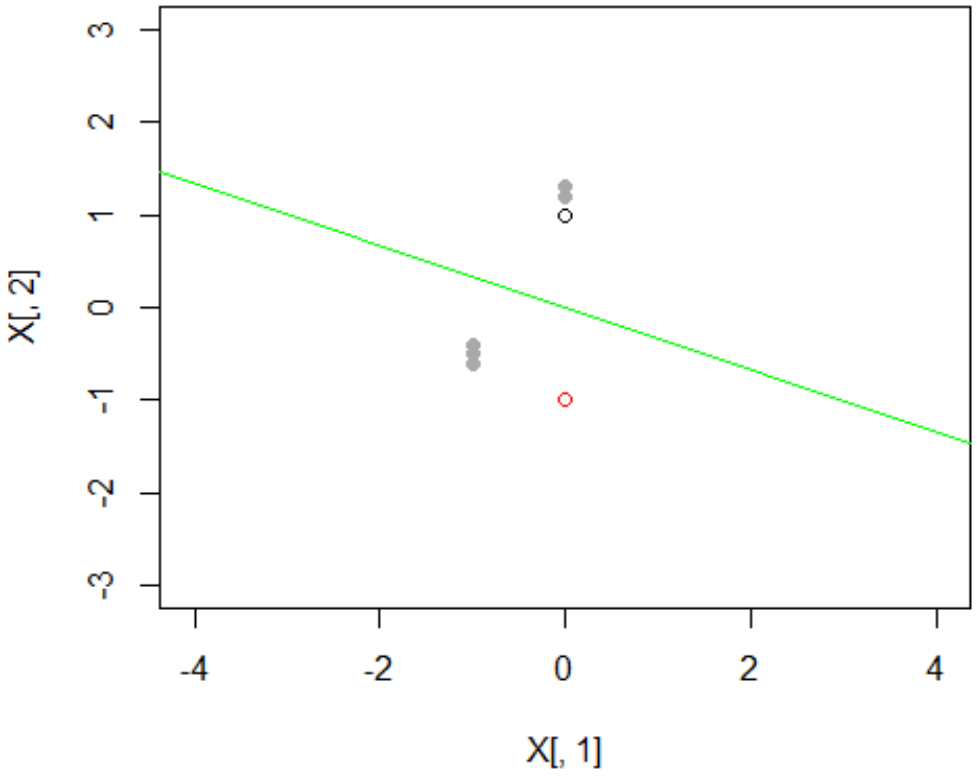


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

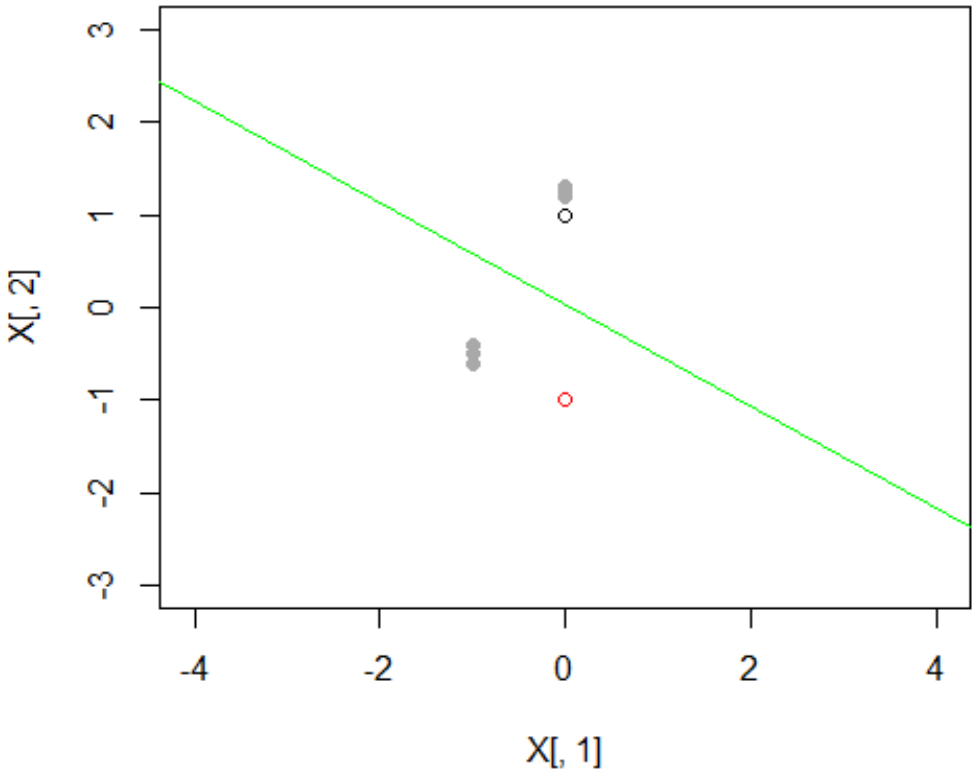


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

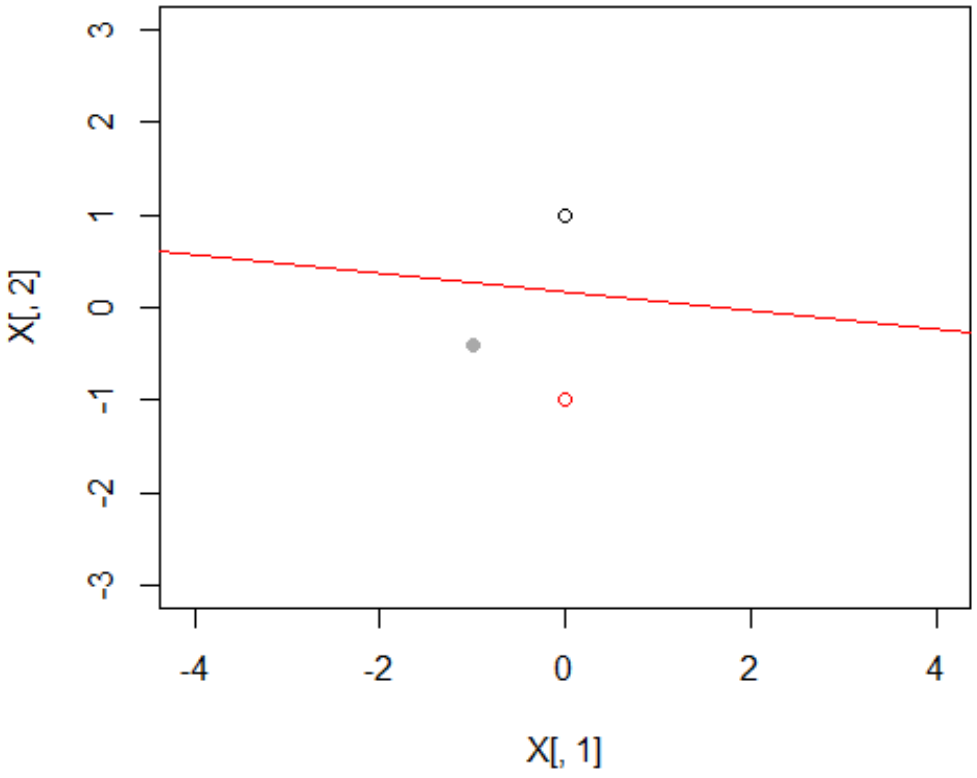


TSVMs

Balancing

constraint = TRUE

Transductive support vector machines

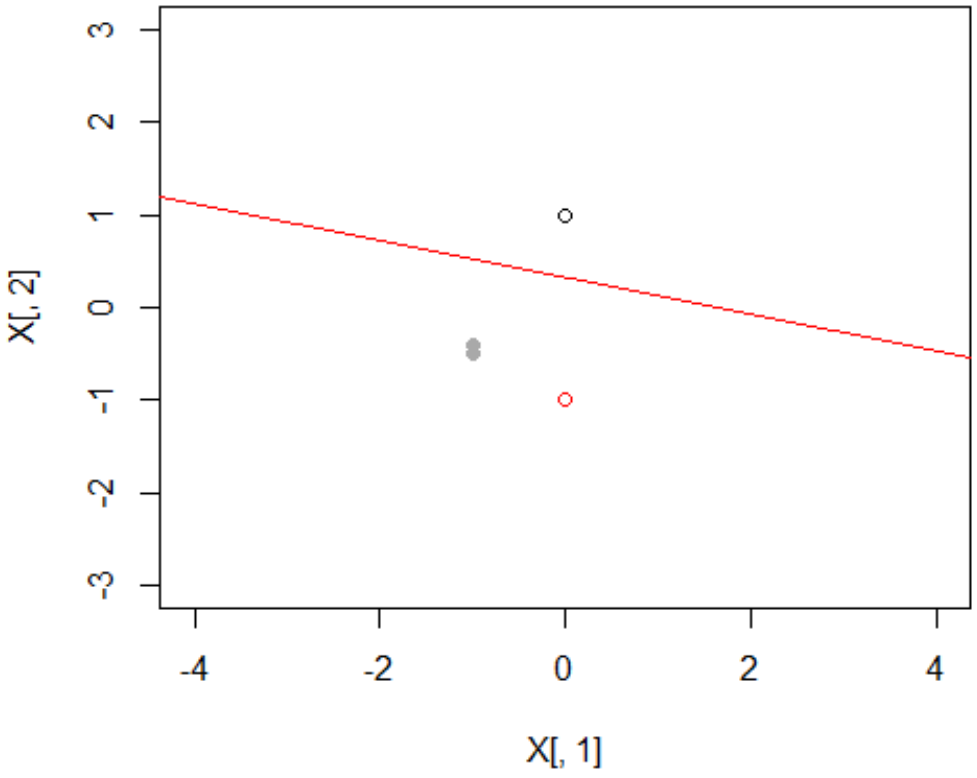


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

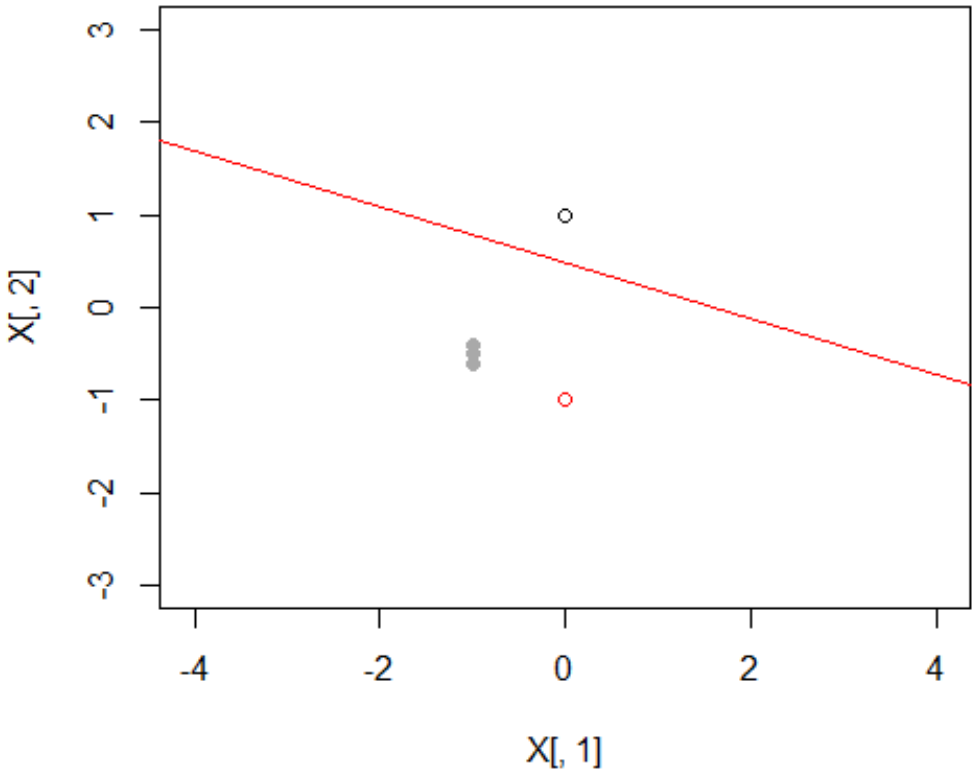


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

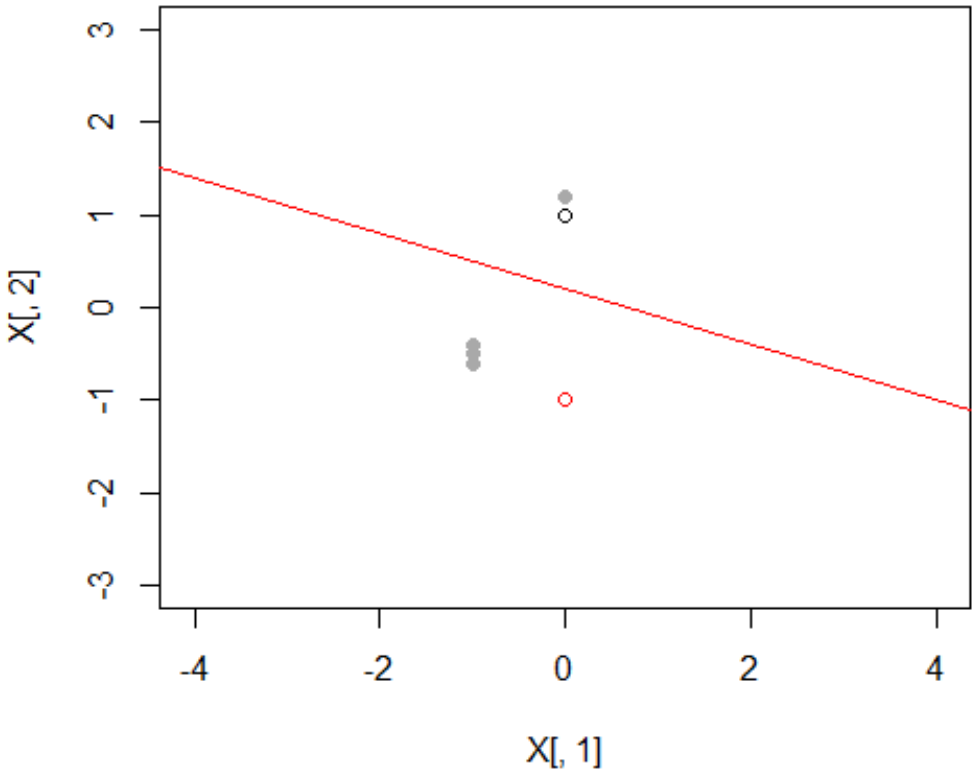


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

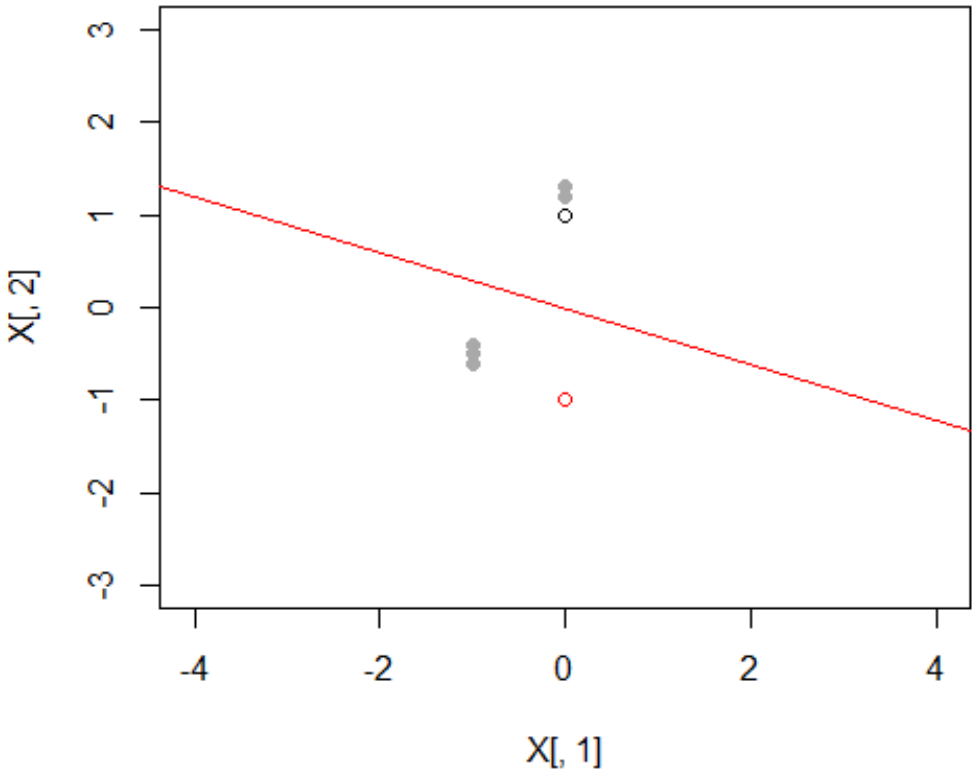


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

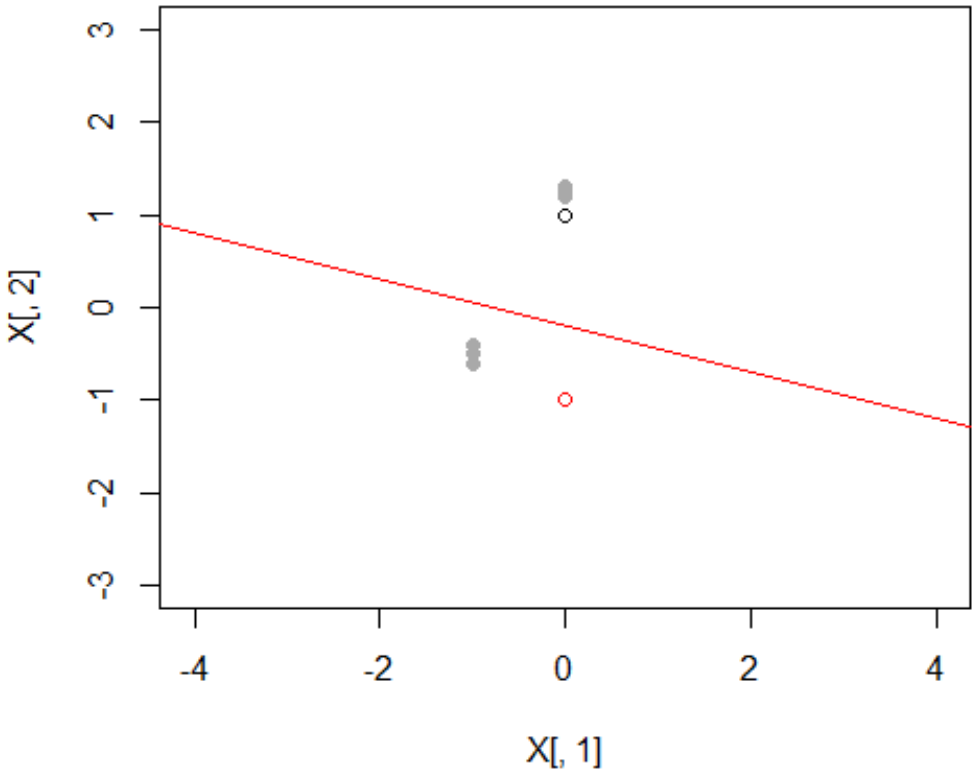


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

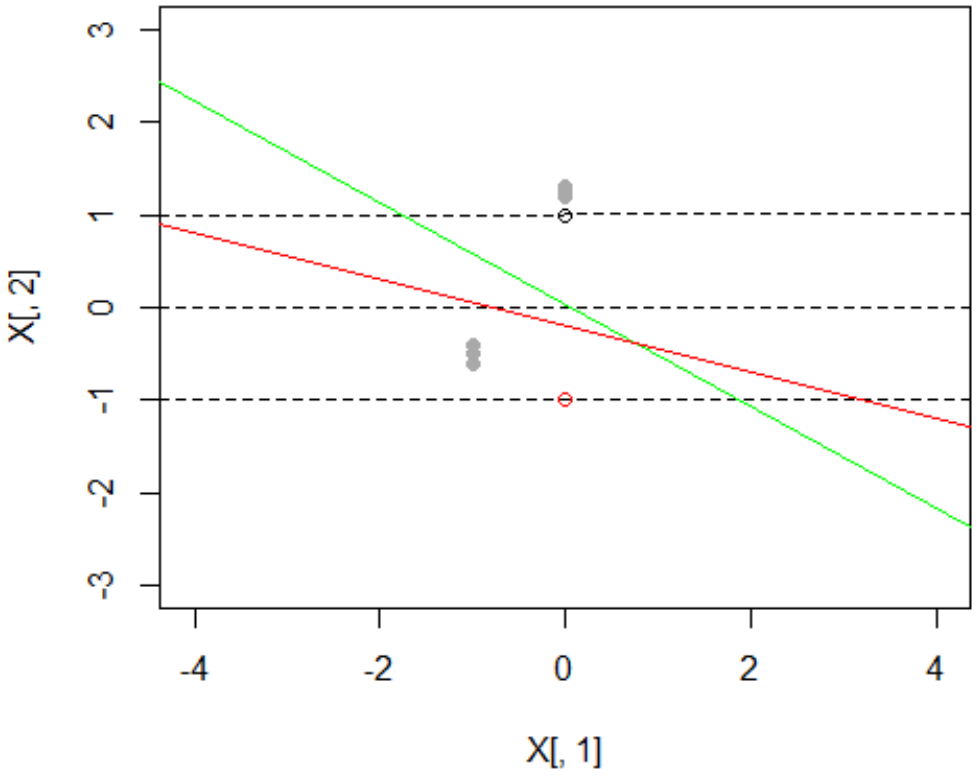


TSVMs

Balancing

constraint = FALSE

Transductive support vector machines

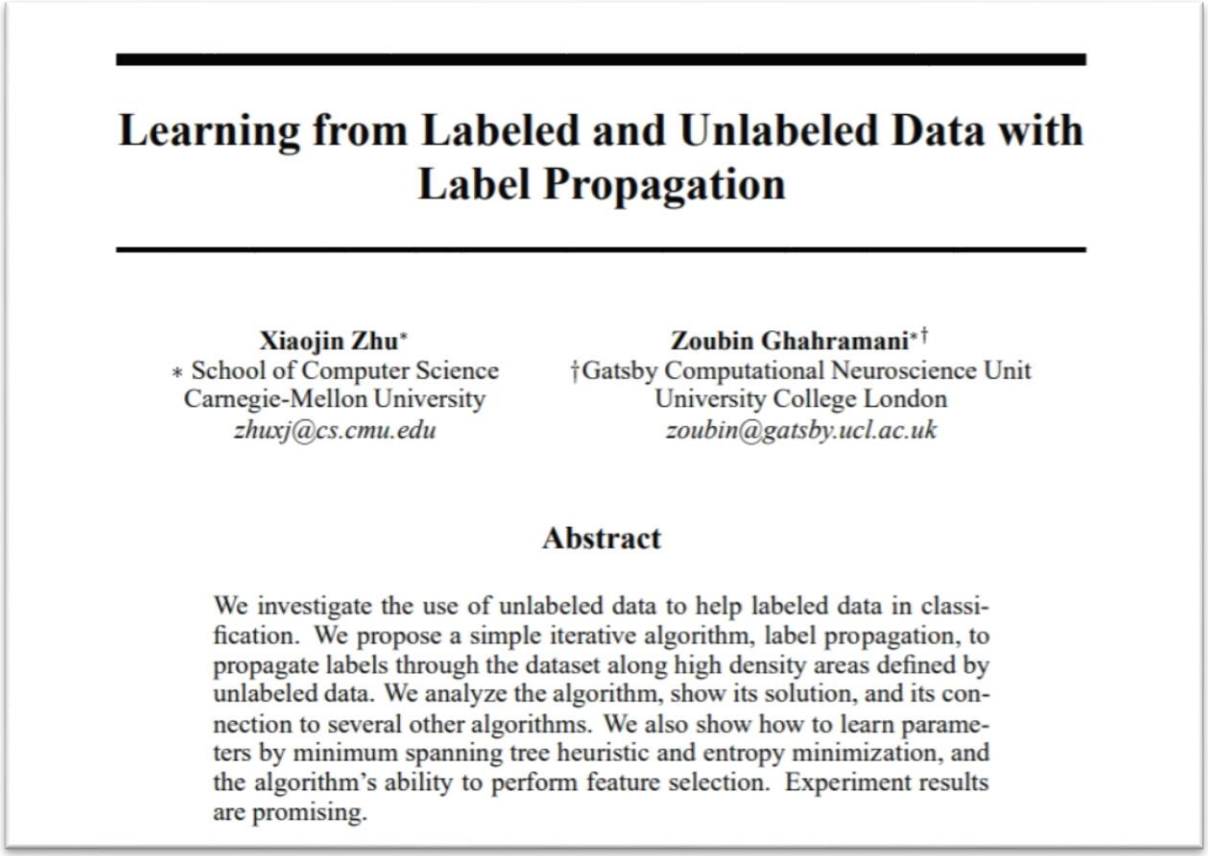


SVM

TSVMs (BC=T)

TSVMs (BC=F)

Learning from Labeled and Unlabeled Data with Label Propagation



Learning from Labeled and Unlabeled Data with Label Propagation

Xiaojin Zhu[*]
∗ School of Computer Science
Carnegie-Mellon University
zhuxj@cs.cmu.edu

Zoubin Ghahramani[*†]
†Gatsby Computational Neuroscience Unit
University College London
zoubin@gatsby.ucl.ac.uk

Abstract

We investigate the use of unlabeled data to help labeled data in classification. We propose a simple iterative algorithm, label propagation, to propagate labels through the dataset along high density areas defined by unlabeled data. We analyze the algorithm, show its solution, and its connection to several other algorithms. We also show how to learn parameters by minimum spanning tree heuristic and entropy minimization, and the algorithm's ability to perform feature selection. Experiment results are promising.

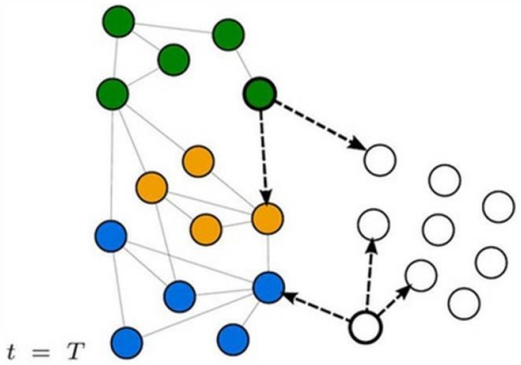
Label Propagation

Label propagation is a semi-supervised machine learning algorithm that assigns labels to previously unlabeled data points. At the start of the algorithm, a (generally small) subset of the data points have labels (or classifications).

These labels are propagated to the unlabeled points throughout the course of the algorithm.

Every node is initialized with a unique label, then the labels diffuse through the network. Consequently, densely connected groups reach a common label quickly. When many such dense (consensus) groups are created throughout the network, they continue to expand outwards until it is impossible to do so.

Label Propagation



The process has 5 steps:

1. Initialize the labels at all nodes in the network. For a given node x, $C_x(0) = x$.

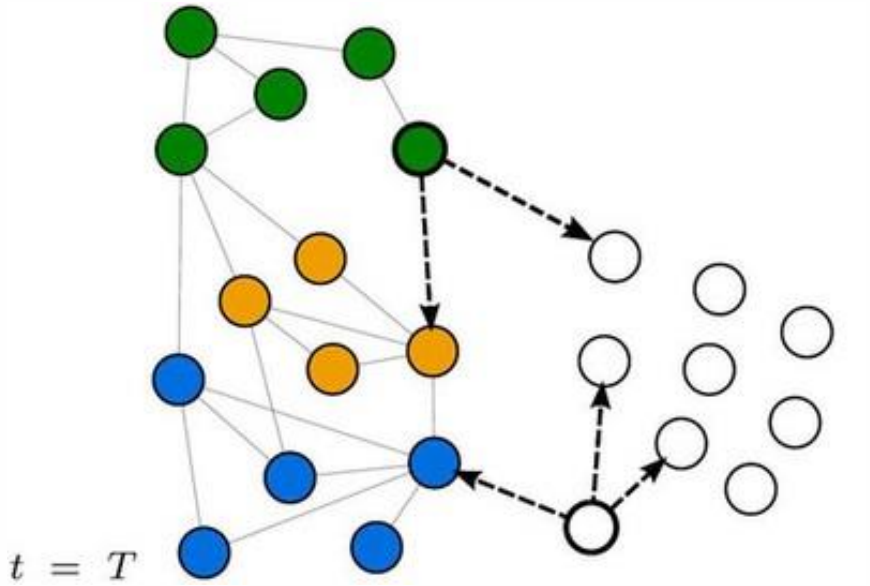
2. Set t = 1.

3. Arrange the nodes in the network in a random order and set it to X. (Initial the direction ---------->)

4. For each x $\in$ X chosen in that specific order, let $C_x(t) = f(C_{xi1}(t), ..., C_{xim}(t), C_{xi(m+1)}(t-1), ..., C_{xik}(t-1))$. f here returns the label occurring with the highest frequency among neighbors. Select a label at random if there are multiple highest frequency labels.

5. If every node has a label that the maximum number of their neighbours have, then stop the algorithm. Else, set t = t + 1 and go to (3).

Label Propagation

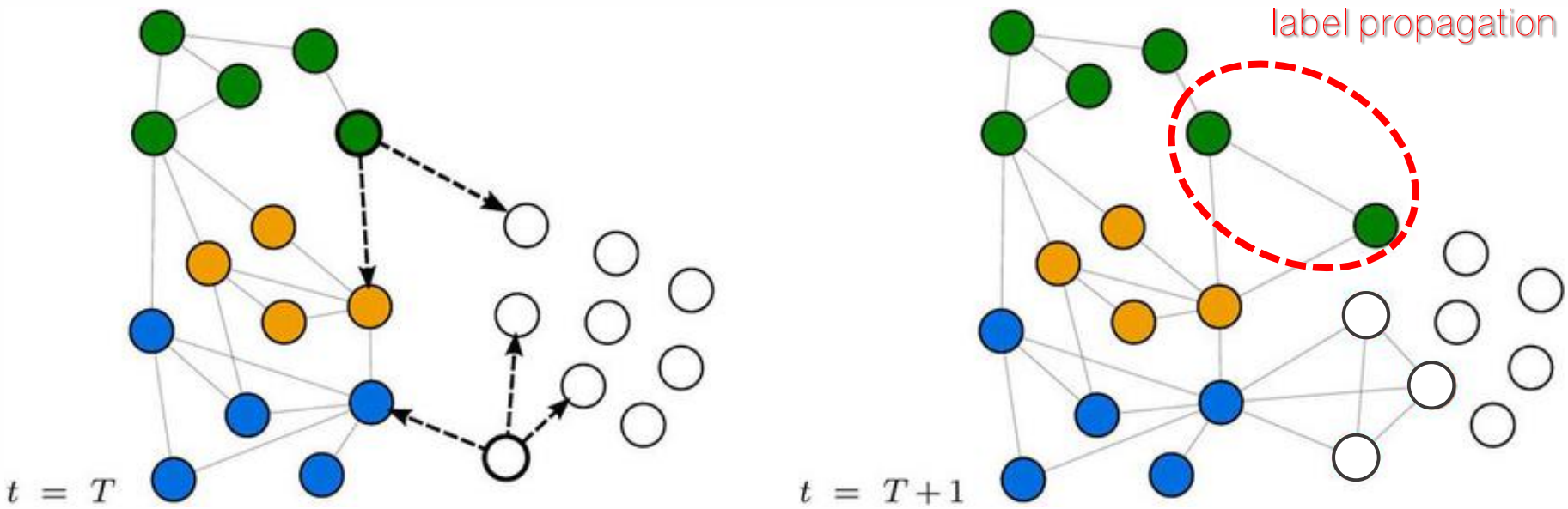


- Initialize the labels instance at all nodes (labeled data), Here are three class = {green, yellow, blue}
- Arrange the nodes in the network in a random order and set it to X (Initial the direction ---------->)

Label Propagation

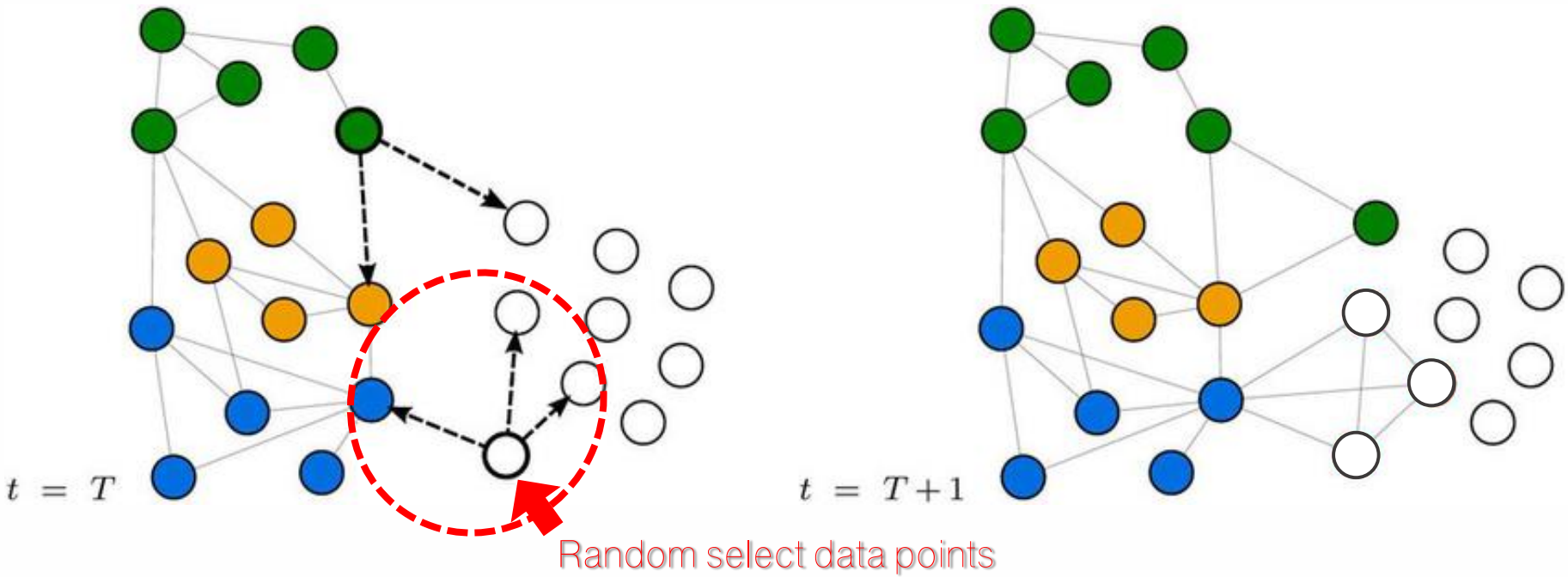
A labeled node (whose label is depicted in green) chooses to form an alliance with m = 2 partners, one having a different label (depicted in yellow) and one non-labeled, at time t = T.

Label Propagation



The initiator propagates its green label at time t = T + 1 only to the previously non-labeled node. The link with the yellow node is still formed, but the label propagation does not occur.

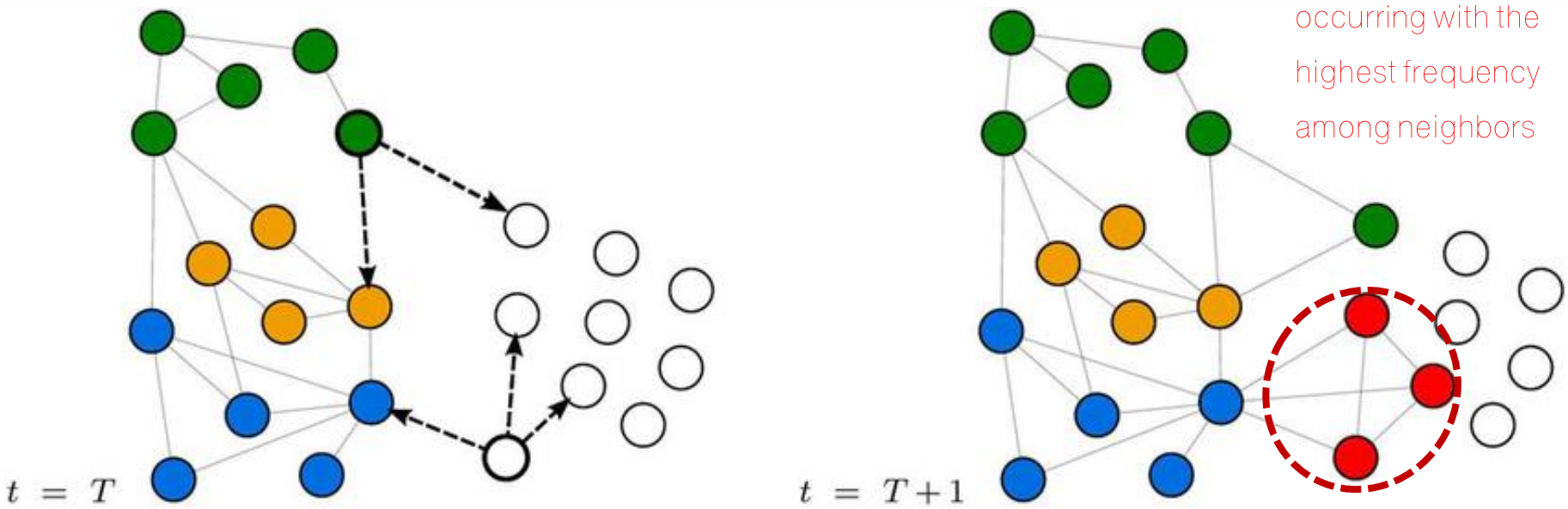
Label Propagation



A non-labeled node gets activated at time t = T and forms an alliance with m = 3 partners, two non-labeled nodes and one labeled (blue) node.

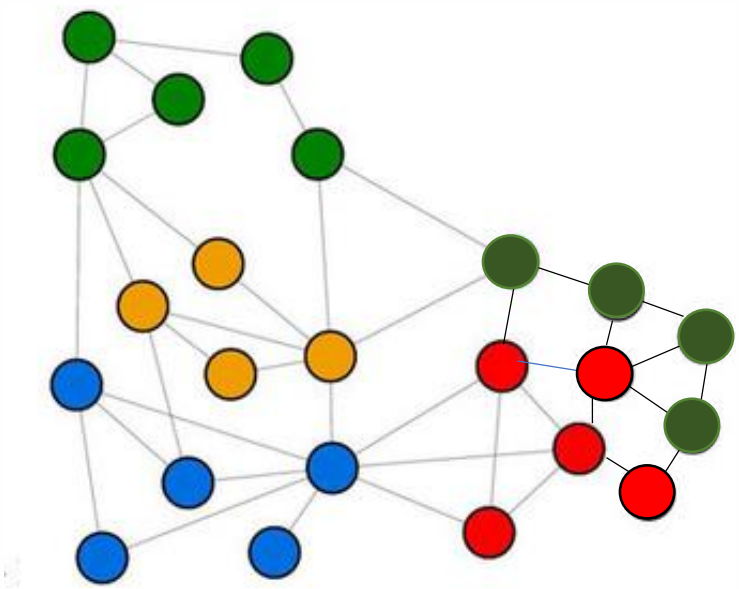
Label Propagation

returns the label occurring with the highest frequency among neighbors



The non-labeled initiator takes a new arbitrary label (depicted in red) at time t = T + 1 and propagates it only to its previously non-labeled partners. The red label is not propagated to the blue node, even though the links are regularly formed.

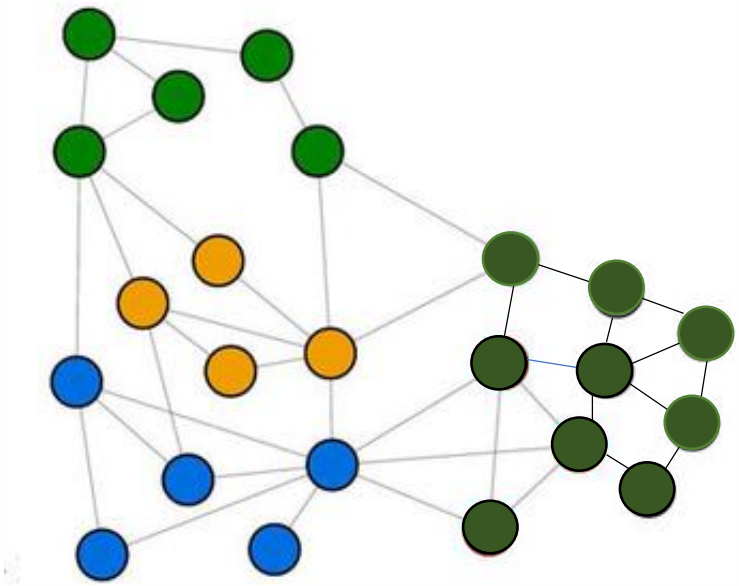
Label Propagation



$$t = T + j,$$

$$j = 1, 2, 3, \ldots, J$$

Label Propagation



$t = T + J,$

$j = 1, 2, 3, \ldots, J$

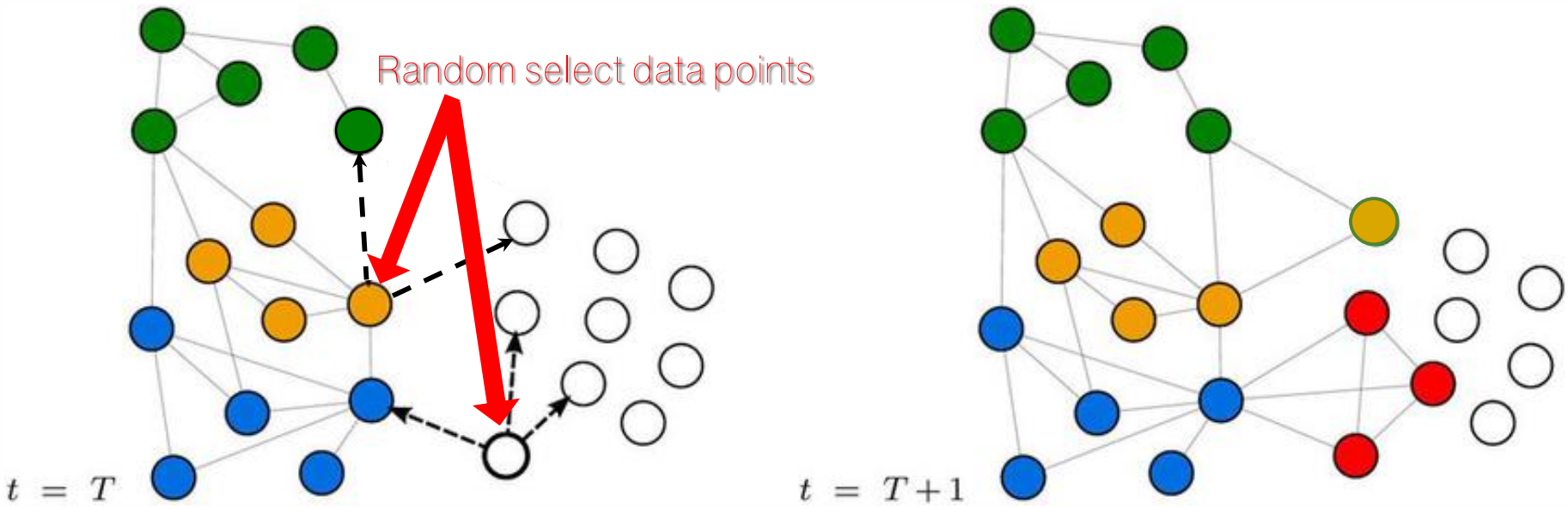
If every node has a label that the maximum number of their neighbors have, then stop the algorithm

Label Propagation

Alternative solution #1

(a different random point)

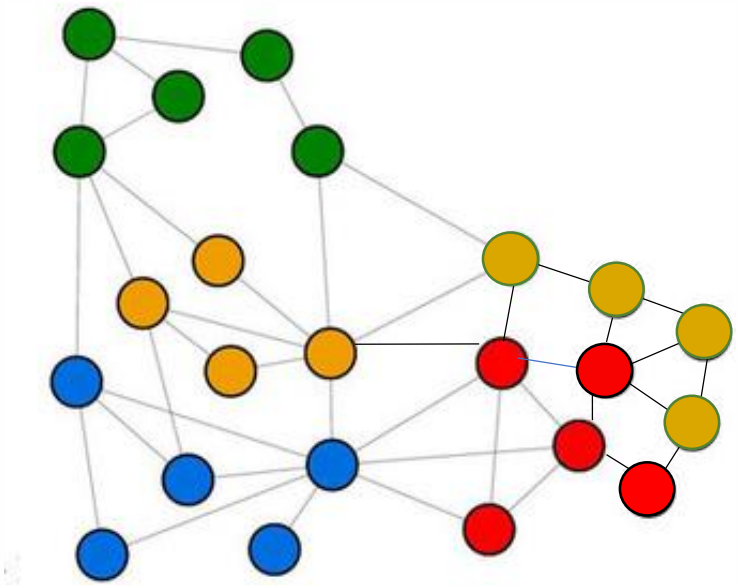
Label Propagation



Alternative solution #1

(a different random point)

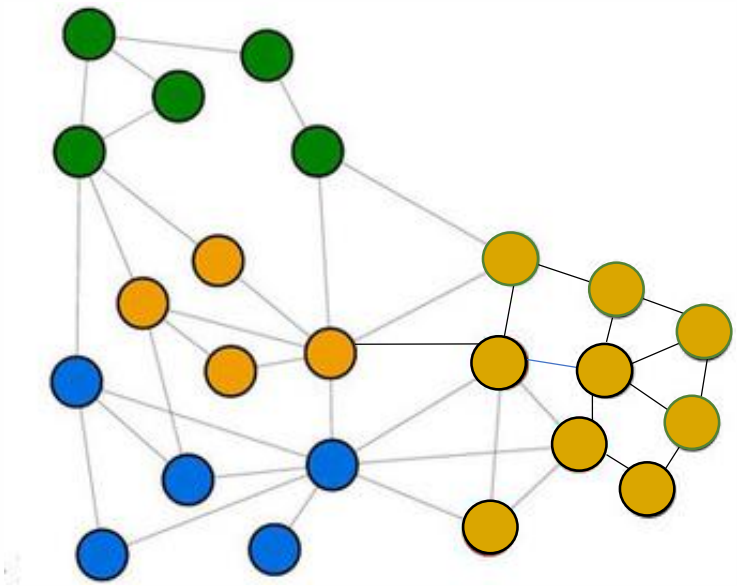
Label Propagation



Alternative solution #1

(a different random point)

Label Propagation



Alternative solution #1

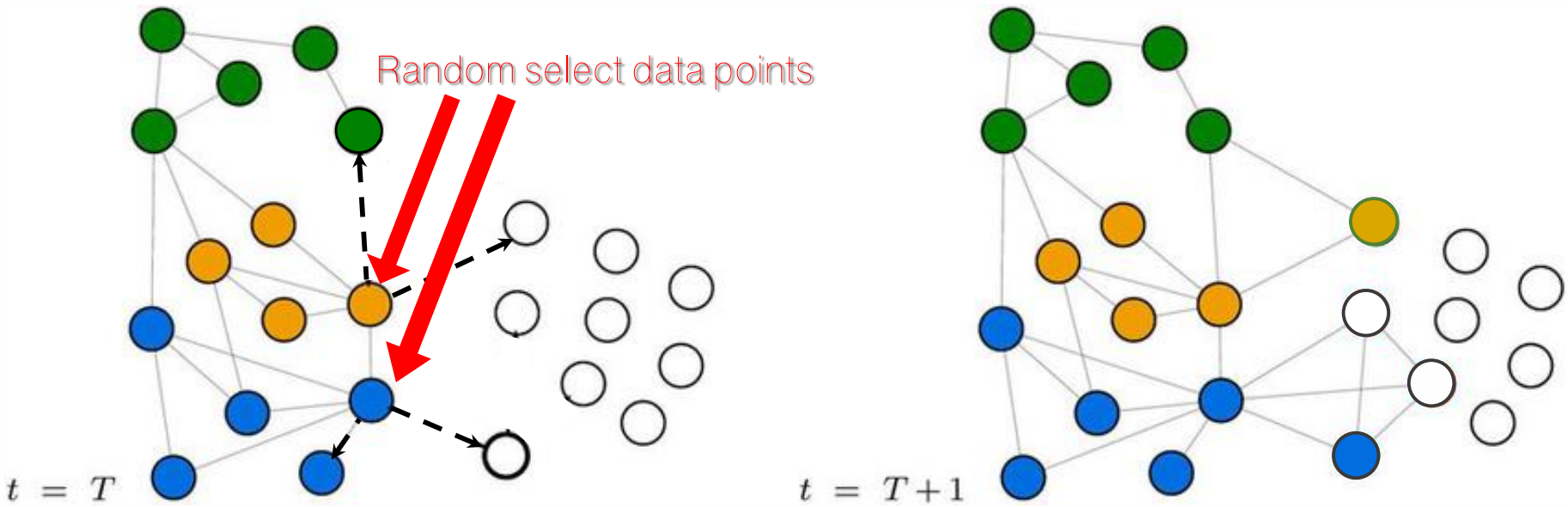
(a different random point)

Label Propagation

Alternative solution #2

(a different random point)

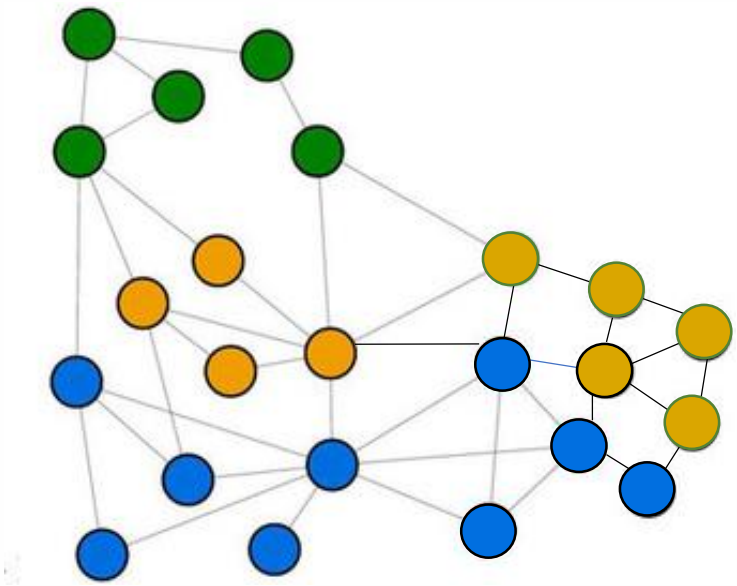
Label Propagation



Alternative solution #2

(a different random point)

Label Propagation



Alternative solution #2

(a different random point)

Decision boundary of label propagation versus SVM on the Iris dataset by using Python

https://scikit-learn.org/stable/auto_examples/semi_supervised/plot_label_propagation_versus_svm_iris.html#sphx-glr-auto-examples-semi-supervised-plot-label-propagation-versus-svm-iris-py

label propagation versus SVM

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn import svm
from sklearn.semi_supervised import LabelSpreading

rng = np.random.RandomState(0)
iris = datasets.load_iris()

X = iris.data[:, :2]
y = iris.target
```

label propagation versus SVM

```
h = .02
```

step size in the mesh

```
y_30 = np.copy(y)
y_30[rng.rand(len(y)) < 0.3] = -1
```

Copy y for 30% spreading example and random assign label = -1 by 30% (Original y has 3 classes = 0, 1, 2 then -1 means unlabeled)

```
y_50 = np.copy(y)
y_50[rng.rand(len(y)) < 0.5] = -1
```

Copy y for 50% spreading example and random assign label = -1 by 50%

```
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
ls30 = (LabelSpreading().fit(X, y_30), y_30)
ls50 = (LabelSpreading().fit(X, y_50), y_50)
ls100 = (LabelSpreading().fit(X, y), y)
rbf_svc = (svm.SVC(kernel='rbf', gamma=.5).fit(X, y), y)
```

Train three models 30% , 50%, and 100% spreading examples, and one model of SVM (ls30, ls50, ls100, rbf_svc are create for plotting)

Train SVM

label propagation versus SVM

```
# create a mesh to plot in. Return coordinate matrices from coordinate vectors
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
# title for the plots
titles = ['Label Spreading 30% data',
          'Label Spreading 50% data',
          'Label Spreading 100% data',
          'SVC with rbf kernel']
```

-1 : for unlabeled
0: for class 0
1: for class 1
2: for class 2

```
color_map = {-1: (1, 1, 1), 0: (0, 0, .9), 1: (1, 0, 0), 2: (.8, .6, 0)}
```

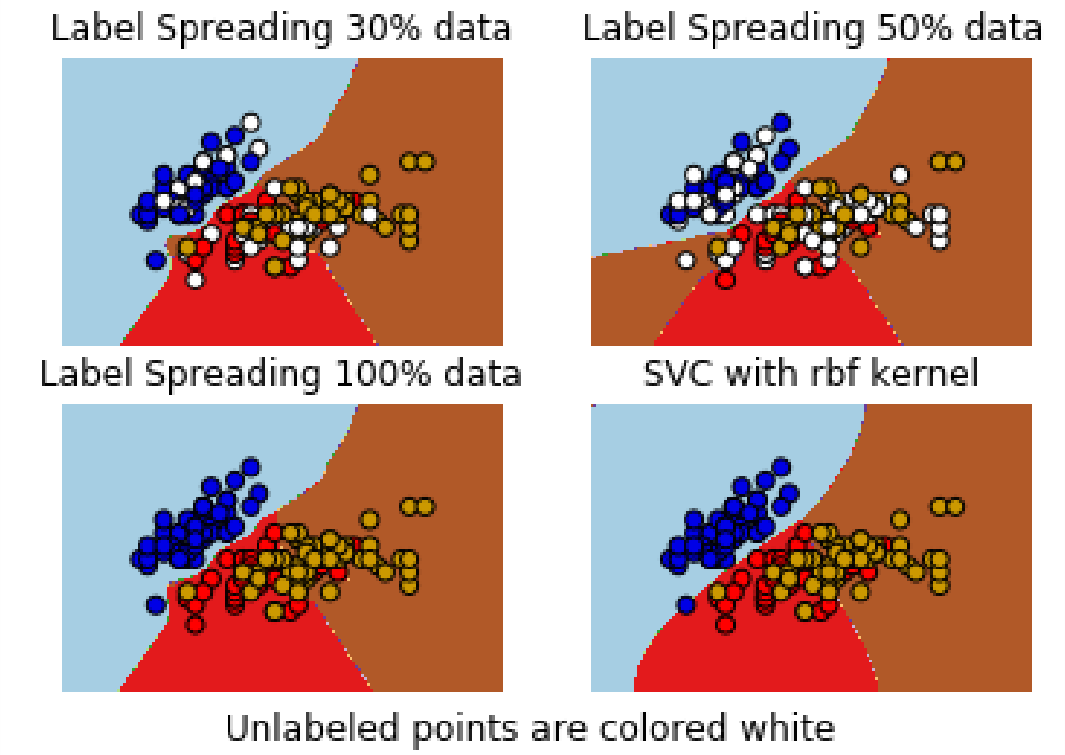
label propagation versus SVM

Plot points and decision boundary

```
for i, (clf, y_train) in enumerate((ls30, ls50, ls100, rbf_svc)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired)
    plt.axis('off')

    # Plot also the training points
    colors = [color_map[y] for y in y_train]
    plt.scatter(X[:, 0], X[:, 1], c=colors, edgecolors='black')

    plt.title(titles[i])
```

label propagation versus SVM

```
plt.suptitle("Unlabeled points are colored white", y=0.1)
plt.show()
```



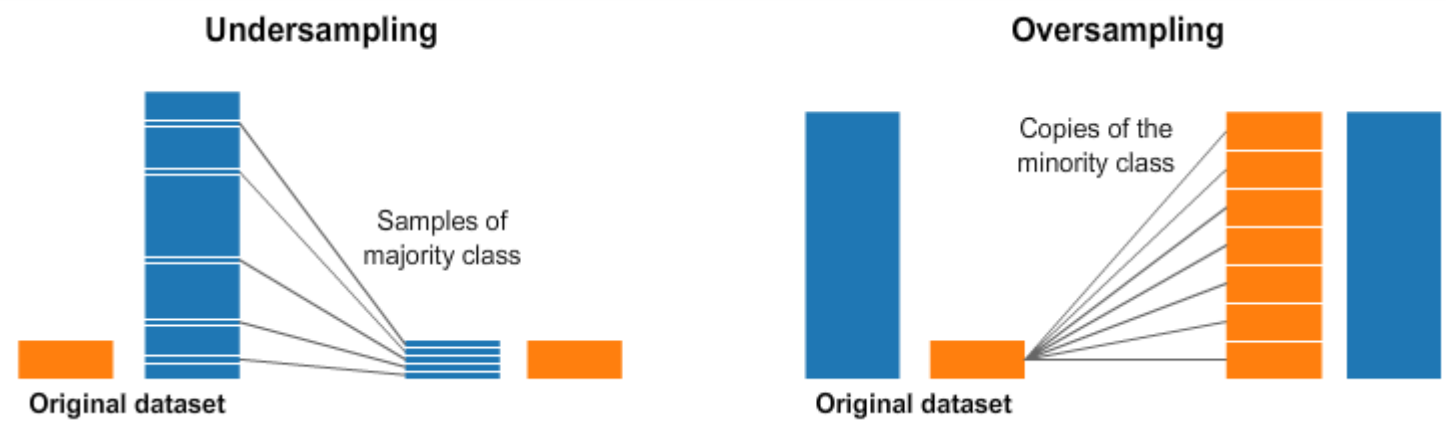
How to apply semi-supervised learning idea in imbalanced dataset management

What can you do? If your dataset is imbalanced

- Collect more data (difficult in many domains)
- Delete data from the majority class
- Create synthetic data
- Adapt your learning algorithm (cost sensitive classification)

Random over/under sampling

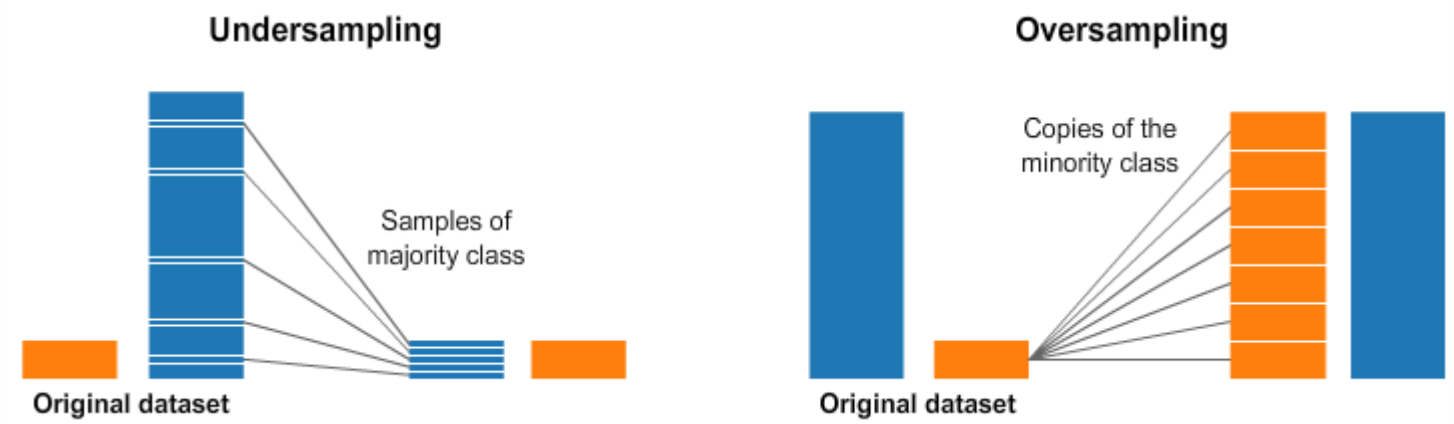
- Random undersampling: randomly delete data points from the majority class.
- Random oversampling: randomly duplicate data points from the minority class.



https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html

Problems with these approaches:

- Loss of information (in the case of under sampling)
- Overfitting and fixed boundaries (over sampling)



https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html

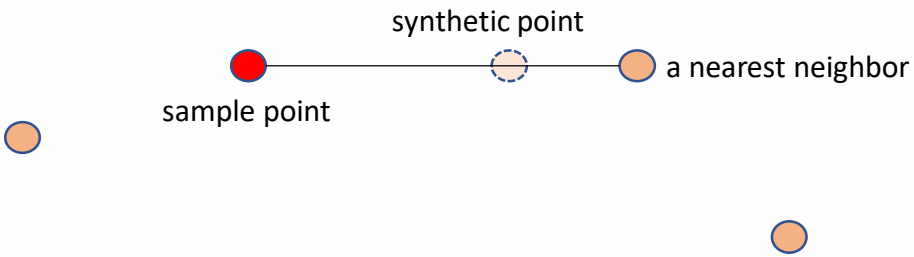
SMOTE - Oversampling Technique

- Synthetic Minority Over-sampling Technique (Chawla).
- Creates new data points from the minority class.
- Operates in the feature space.

SMOTE – Main Steps

1. Take the difference between a sample point and one of its nearest neighbors.
2. Multiply the difference by a random number between 0 and 1 and add it to the feature vector.

This causes the selection of a random point along the line segment between two specific features.



Danger of information injection and overfitting

Do not create synthetic points on the entire dataset before splitting into train/test sets.

- Perform the preprocessing just on the training data!!
- For k-fold cross validation, you have to do it for each fold (just on the training set).

Danger of information injection and overfitting

SMOTE for Imbalanced Classification with Python

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

Imbalanced-Learn Library

In these examples, we will use the implementations provided by the imbalanced-learn Python library, which can be installed via anaconda prompt as follow:

```
conda install -c conda-forge imbalanced-learn
```

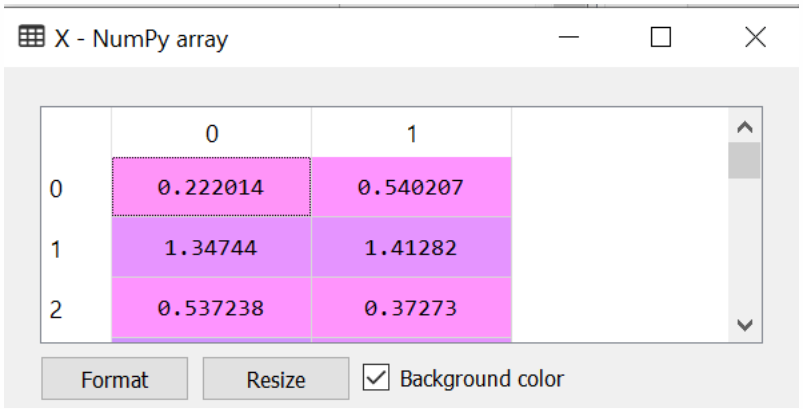
SMOTE in python

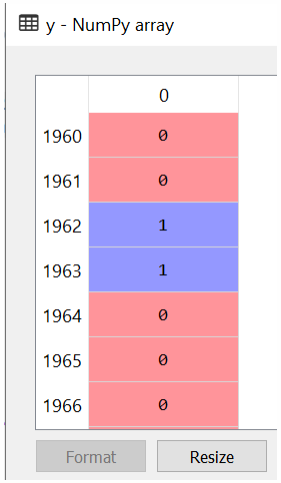
```
# check version number
import imblearn
print(imblearn.__version__)
```

```
0.7.0
```

SMOTE in python

```
# define dataset
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=10000, n_features=2,
n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
```





SMOTE in python

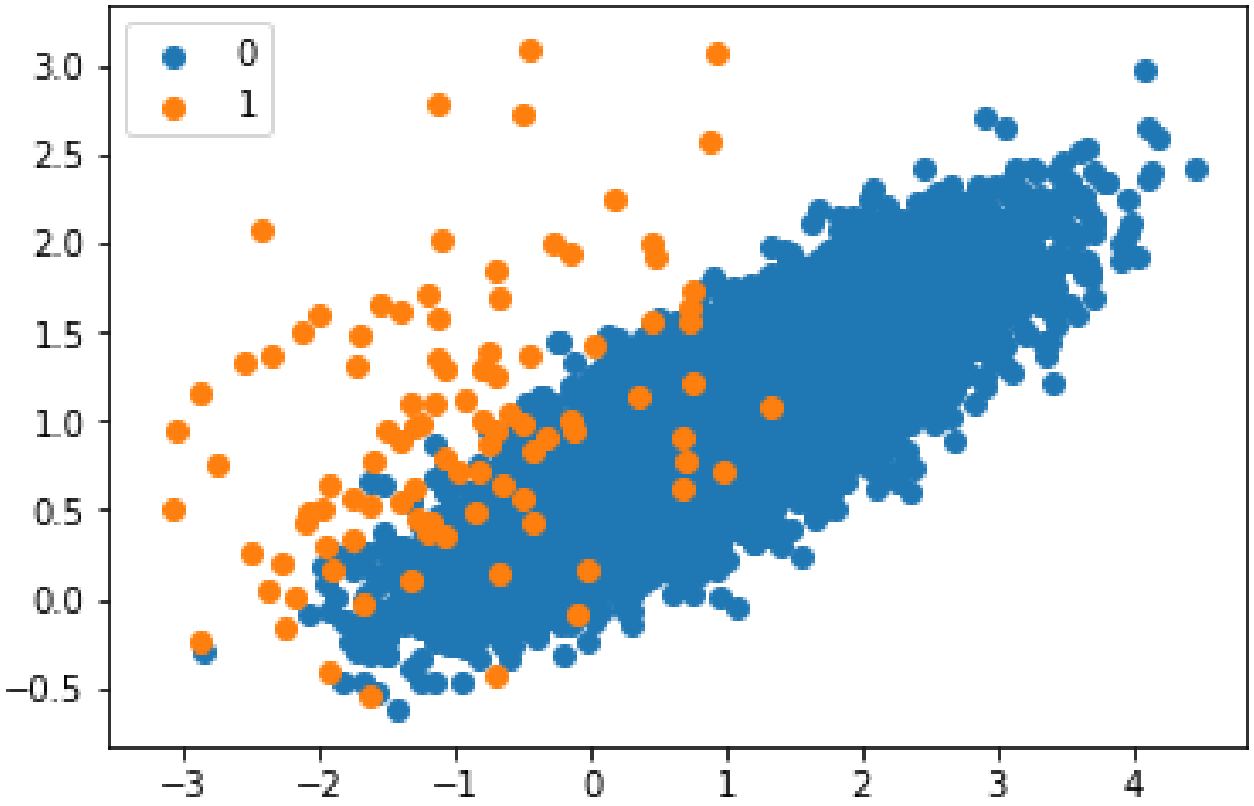
```
# summarize class distribution
from collections import Counter
counter = Counter(y)
print(counter)
```

```
Counter({0: 9900, 1: 100})
```

SMOTE in python

```
# scatter plot of examples by class label
from matplotlib import pyplot
from numpy import where

for label, _ in counter.items():
	row_ix = where(y == label)[0]
	pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

SMOTE in python



SMOTE in python

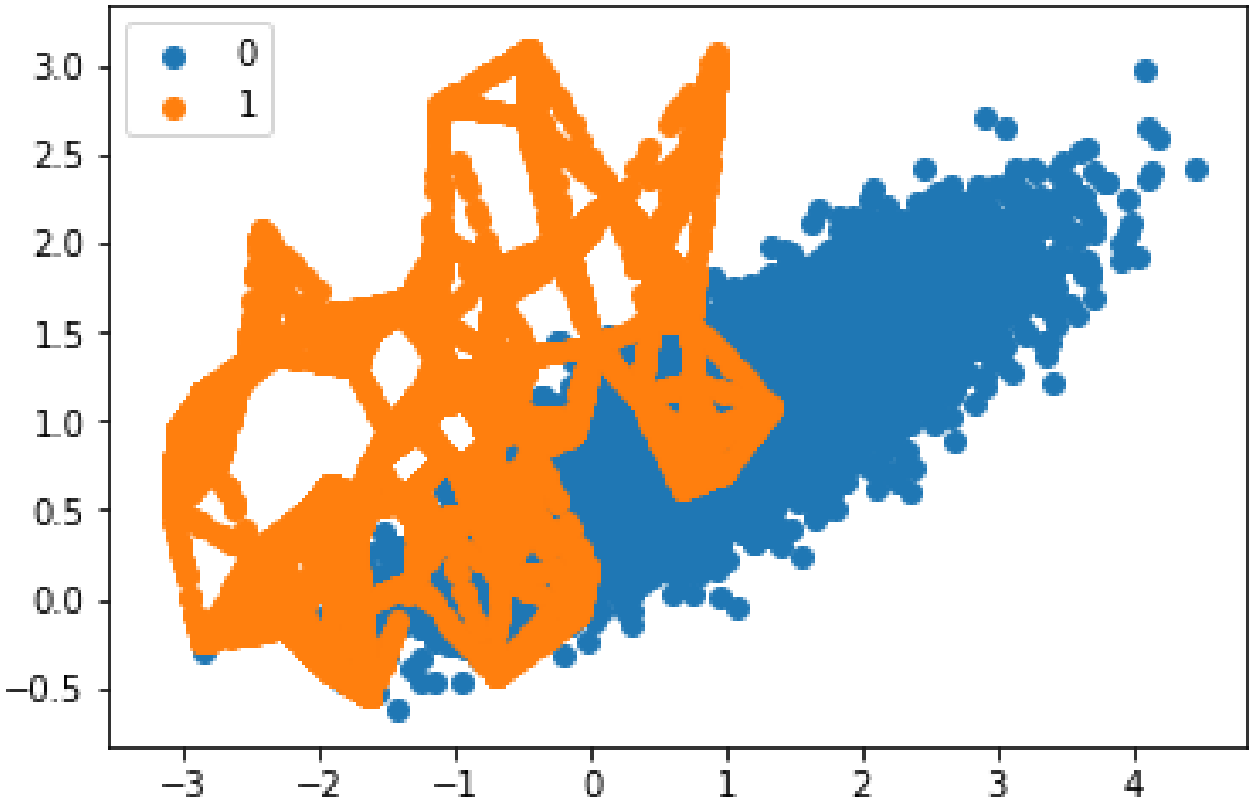
```
# transform the dataset
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)

# summarize the new class distribution
counter = Counter(y)
print(counter)
```

```
Counter({0: 9900, 1: 9900})
```

SMOTE in python

```
# scatter plot of examples by class label
for label, _ in counter.items():
	row_ix = where(y == label)[0]
	pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

SMOTE in python



With SMOTE

Transformed

SMOTE with random undersampling of the majority class

SMOTE with random undersampling of the majority class

- The original paper on SMOTE suggested combining SMOTE with random undersampling of the majority class.

- We can update our example
  - oversample the minority class to have 10 percent the number of examples of the majority class (e.g. about 1,000)
  - random undersampling to reduce the number of examples in the majority class to have 50 percent more than the minority class (e.g. about 2,000).

SMOTE with random undersampling of the majority class

```
# Oversample with SMOTE and random undersample for imbalanced
dataset
from imblearn.under_sampling import RandomUnderSampler
over = SMOTE(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=0.5)
```

if set = 0.5, means the number of over sampling is a half of under sampling

if set = 1, means the number of over sampling = under sampling

SMOTE with random undersampling of the majority class

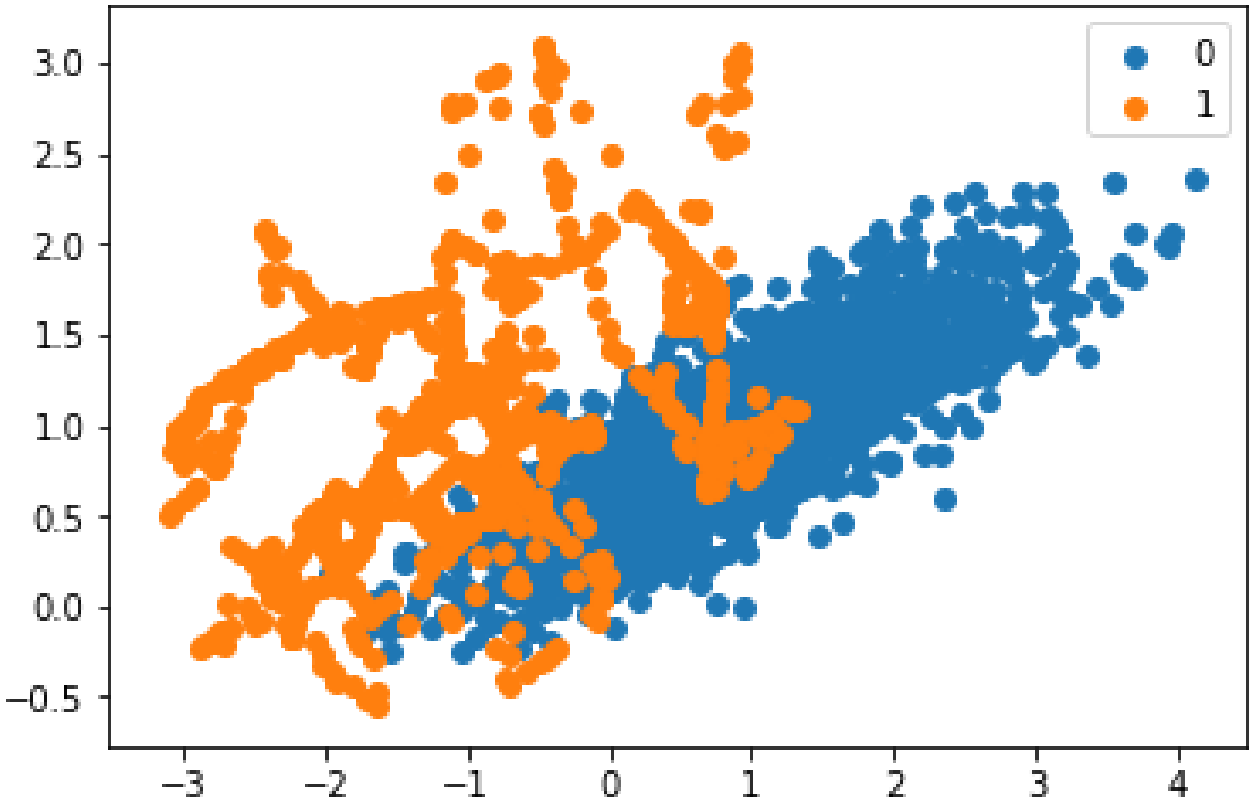
```
from imblearn.pipeline import Pipeline
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
X, y = make_classification(n_samples=10000, n_features=2,
n_redundant=0,
        n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
X, y = pipeline.fit_resample(X, y)
counter = Counter(y)
print(counter)
```

```
Counter({0: 1980, 1: 990})
```

SMOTE in python

```
# scatter plot of examples by class label
for label, _ in counter.items():
	row_ix = where(y == label)[0]
	pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

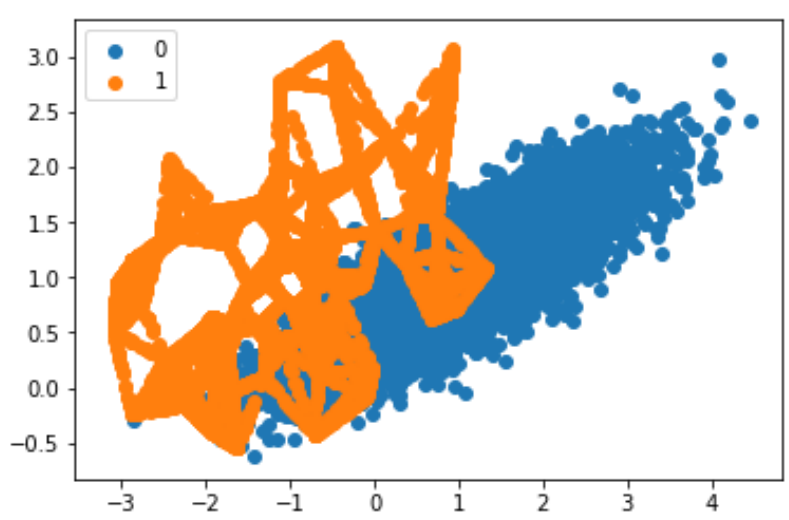
SMOTE in python



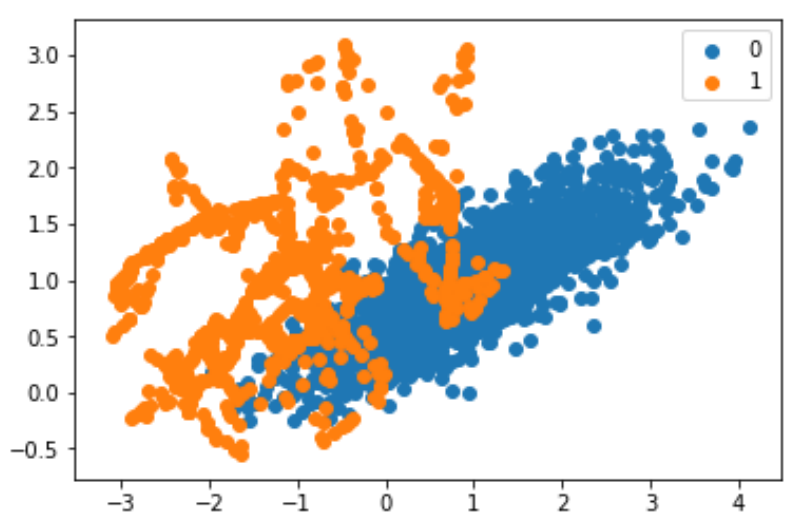
With SMOTE in undersampling

SMOTE in python

SMOTE



SMOTE with undersampling



SMOTE for Classification

SMOTE for Classification

```
# decision tree evaluated on imbalanced dataset
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
```

SMOTE for Classification

```
# define dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
# define model
model = DecisionTreeClassifier()
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
```

SMOTE for Classification

```
# print Mean Score of ROC AUC
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.764
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Use a SMOTE transformed version of the dataset.

SMOTE for Classification

```
# define dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
# define pipeline
steps = [('over', SMOTE()), ('model', DecisionTreeClassifier())]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))
```

Mean ROC AUC: 0.817

Use a SMOTE transformed version of the dataset combined with undersampling of the majority class

SMOTE for Classification

```
from imblearn.under_sampling import RandomUnderSampler
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
model = DecisionTreeClassifier()
over = SMOTE(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('over', over), ('under', under), ('model', model)]
pipeline = Pipeline(steps=steps)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))
```


Mean ROC AUC: 0.843

Use a SMOTE transformed version of the dataset combined with undersampling of the majority class

Test different values of the k-nearest neighbors selected in the SMOTE procedure when each new synthetic example is created

SMOTE for Classification

```
# define dataset
X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
```

SMOTE for Classification

```
k_values = [1, 2, 3, 4, 5, 6, 7]
for k in k_values:
	# define pipeline
	model = DecisionTreeClassifier()
	over = SMOTE(sampling_strategy=0.1, k_neighbors=k)
	under = RandomUnderSampler(sampling_strategy=0.5)
	steps = [('over', over), ('under', under), ('model', model)]
	pipeline = Pipeline(steps=steps)
	# evaluate pipeline
	cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
	scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
	score = mean(scores)
	print('> k=%d, Mean ROC AUC: %.3f' % (k, score))
```

SMOTE for Classification

```
> k=1, Mean ROC AUC: 0.818
> k=2, Mean ROC AUC: 0.832
> k=3, Mean ROC AUC: 0.849
> k=4, Mean ROC AUC: 0.841
> k=5, Mean ROC AUC: 0.848
> k=6, Mean ROC AUC: 0.845
> k=7, Mean ROC AUC: 0.855
```

In-Class Assignment:

1) From the example of how to use a SMOTE transformed version of the dataset combined with undersampling of the majority class and test different values of the k-nearest neighbors selected in the SMOTE procedure when each new synthetic example is created.
2) Please use SVM Classifier and run with 15-Fold cross validation.

Mahidol University
Faculty of Medicine Ramathibodi Hospital
Department of Clinical Epidemiology and Biostatistics

Note: please submit your code and training data.

Assignment: due date November 27, 2022

a) (6 points)

1) From colon.csv, please reduce data dimension by using 2 features (column's name or gene#: T62947 and H64807).
2) Generate labeled and unlabeled data by random selecting 42 samples without replacement and removing their class to perform as unlabeled data. The non-selected 20 samples will be used as labeled data (42:20).
3) Perform SVM by using 20 labelled samples.
4) Perform Label Propagation at label spreading 30%, 50%, and 70% by using Python and data from 1). (random replace the original 62 samples)
5) Compare and discuss the model performance from 3) and 4) by plotting two-dimension graph with decision boundary.

Mahidol University
Faculty of Medicine Ramathibodi Hospital
Department of Clinical Epidemiology and Biostatistics

Note: please submit your code and training data.

Assignment: due date November 27, 2022

b) (6 points)

1) Please select one dataset from a UCI (https://archive.ics.uci.edu/ml/index.php) and reduce data dimension by using only 2 features (select your own two features by random choosing or using any feature selection methods).
2) Generate labeled and unlabeled data by random selecting 68% of samples without replacement and removing their class to perform as unlabeled data. The non-selected 32% of samples will be used as labeled data
3) Perform SVM by using all labelled data from 2). (32% samples)
4) Perform Label Propagation at label spreading 30%, 50%, and 70% by using Python and data from 1). (random replace the original samples)
5) Compare and discuss the model performance from 3) and 4) by plotting two-dimension graph with decision boundary.

Note: please submit your code and training data.

Assignment: due date November 27, 2022

c) (8 points)

1) From colon.csv, please reduce data dimension by using 2 features (column's name or gene#: T62947 and H64807).
2) Use a SMOTE transformed version of the dataset + Decision Tree Classifier
3) Use a SMOTE transformed version of the dataset combined with undersampling of the majority class + Decision Tree Classifier
4) Use a SMOTE transformed version of the dataset combined with undersampling of the majority class + grid search k value for SMOTE oversampling for imbalanced classification + Decision Tree Classifier
5) Compare and discuss the models performance [the models from item 2), 3), and 4)]

References

1. *Harsha S Gowda, Mahamad Suhil, D S Guru and Lavanya Narayana Raju, Semi-Supervised Text Categorization using Recursive K-means clustering, International Conference on Recent Trends in Image Processing and Pattern Recognition, 2017.*
2. *N. Kasabov and Shaoning Pang, Transductive support vector machines and applications in bioinformatics for promoter recognition, International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*
3. *Collobert, R. et al., 2006. Large scale transductive SVMs. Journal of Machine Learning Research, 7, pp.1687-1712.*