Pokémon Recognition

吳易倫

Outline

- Problem
- Dataset
- Package Introduction
- Tasks
    - 1-1 Data Preprocessing
    - 1-2 KNN
    - 1-3 SVM
    - 1-4 PCA

Problem

Pokemon Master

Problem

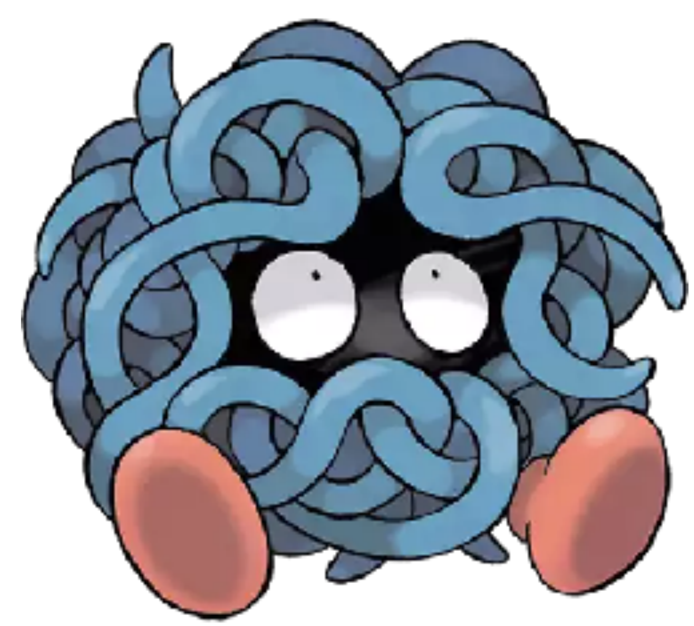
Charizard

Pikachu

?

Gengar

Tangela



Dataset

- 4 classes
- 20 images per class

Package Introduction

- scikit-learn
  - A free software machine learning library for the python
  - Simple and efficient tools for data mining and data analysis
- numpy
  - A powerful N-dimensional array object
  - Sophisticated functions
- Pillow
  - PIL is the Python Imaging Library
- matplotlib
  - A Python 2D plotting library

Tasks

Get source code

- Clone the source code from GitHub

```
$ git clone https://github.com/w86763777/HCC-ML-LAB
```

- Move current directory to LAB1

```
$ cd HCC-ML-LAB/LAB1
```

Setup Environment

- **Ubuntu** or other **Linux** like system
- Use virtual environment
  - Name your virtual environment
  - Specify the version of python
  - Virtual environment is clean at the beginning
  - The package version only depend on current project

Setup Environment

- Create virtual environment
  - Install pip

```
$ sudo apt-get install python3-pip
```

  - Install virtualenv using pip3

```
$ sudo pip3 install virtualenv
```

  - Create a virtual environment

At the root of your project (i.e. HCC-ML-LAB/LAB1)

```
$ virtualenv -p python3 venv
...
done.
```

    - -p specify python interpreter
    - “venv” is your environment name

Setup Environment

- Activate virtual environment

```
$ source venv/bin/activate
(venv) $
```

- Install packages at a time

```
$ pip install -r requirements.txt
...
Successfully installed ...
```

- Leave virtual environment

```
(venv) $ deactivate
$
```

Setup Environment

- In requirements.txt

```
flake8==3.7.7
kiwisolver==1.0.1
matplotlib==3.0.3
...
```

- What is requirements.txt ?
  - You don’t have to manually type pip install several times to get all of your packages installed
  - You don’t have to worry about getting the right version installed

Note. How to create requirements.txt ?

```
$ pip freeze > requirements.txt
```

1-1 Data Preprocessing

- Dataset
  1. https://drive.google.com/open?id=1vkkLO49h6Gk_V4bVAAUJixURWjfxPm4e
  2. Use download script to facilitate the progress. Just run

     ```
     (venv) $ python download.py
     ```

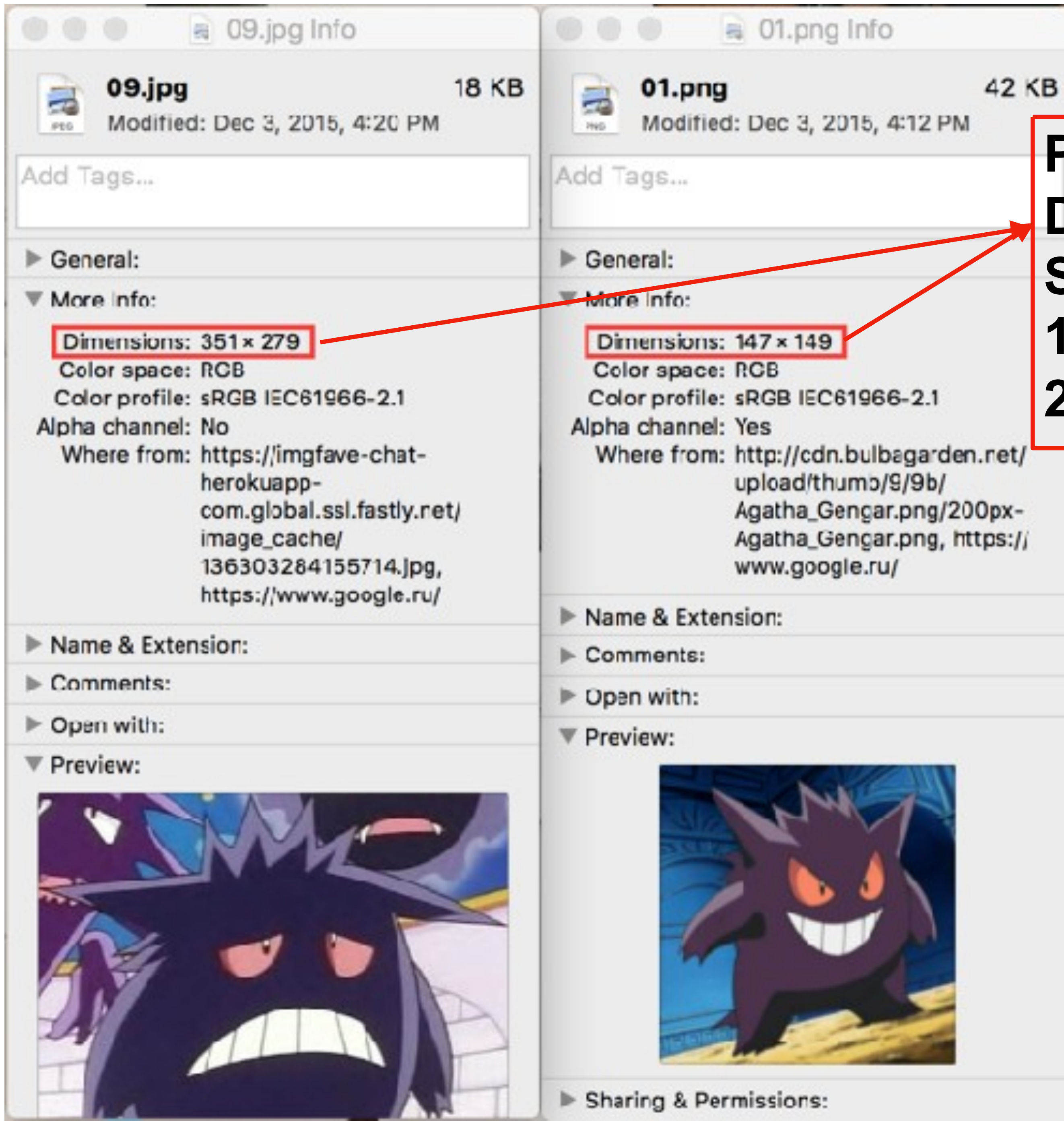
- Unzip pokemon.tar.gz

```
(venv) $ tar −xvf pokemon.tar.gz
```

- You should see

```
└── pokemon
    ├── Charizard
    ├── Gengar
    ├── Pikachu
    └── Tangela
```

1-1 Data Preprocessing



Problem:
Different image size
Solution:
1.Resize all the images into 200x200
2.convert into grayscale

1-1 Data Preprocessing

LAB1.py

```
for i, path in enumerate(sorted(paths)):
    img = Image.open(path)
    # TODO: Checkpoint 1, Preprocessing
    # 2. Convert RGB image to grayscale
    # 1. Resize image into 200x200
    ####
    '''
    img = img.convert(...)
    img = ImageOps.fit(...)
    '''

    new_path = os.path.join(
        POKEMON_PROCESSED_PATH, pokemon_name, '%d.jpg' % i)
    img.save(new_path)
```

1-1 Data Preprocessing

```
img = img.convert(mode=None, matrix=None, dither=None, palette=0,
                  colors=256)


img = PIL.ImageOps.fit(img, size, method=0, bleed=0.0,
                       centering=(0.5, 0.5))
```

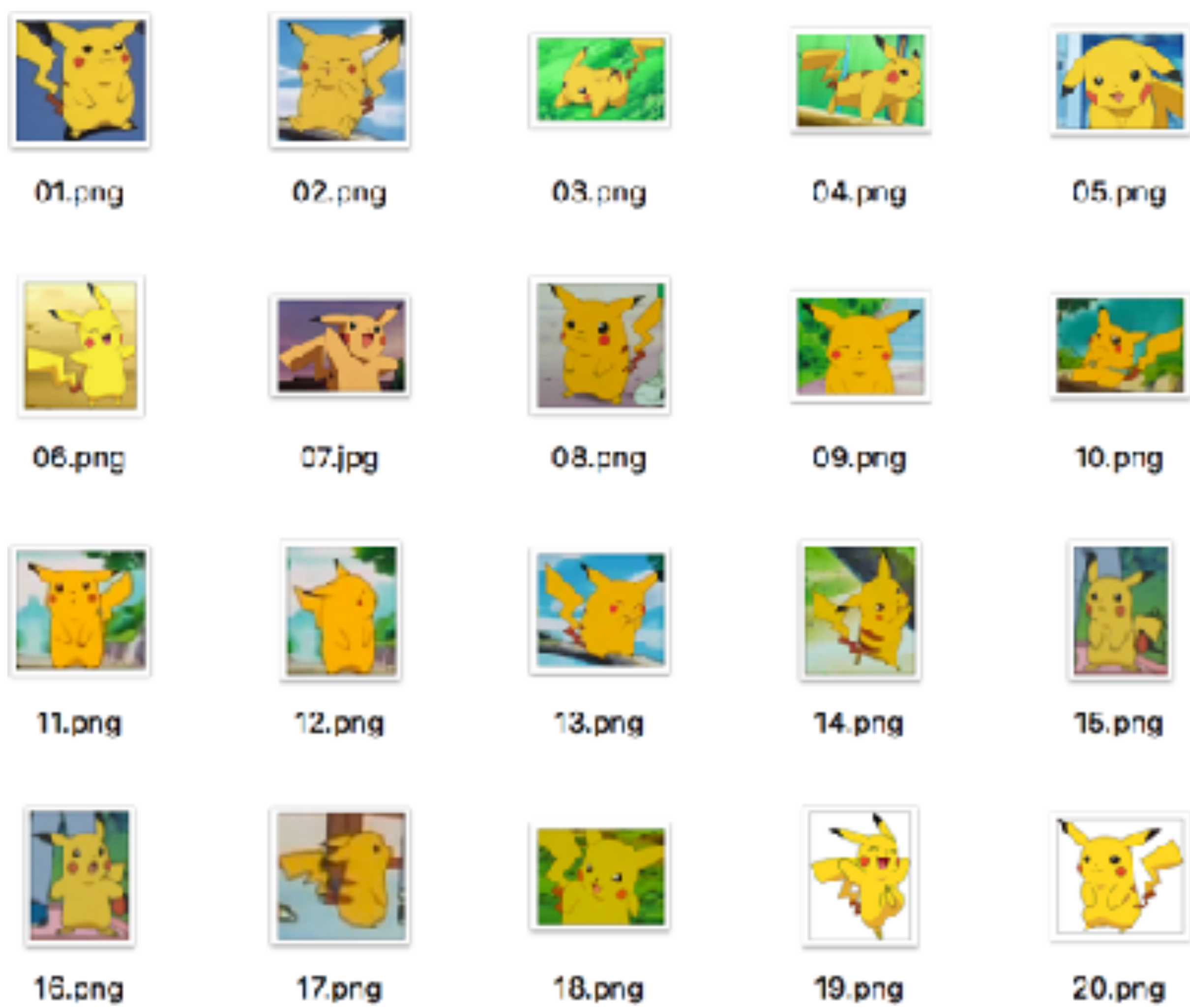
PIL.ImageOps.fit
https://pillow.readthedocs.io/en/stable/reference/ImageOps.html#PIL.ImageOps.fit

Image.convert:
https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.convert

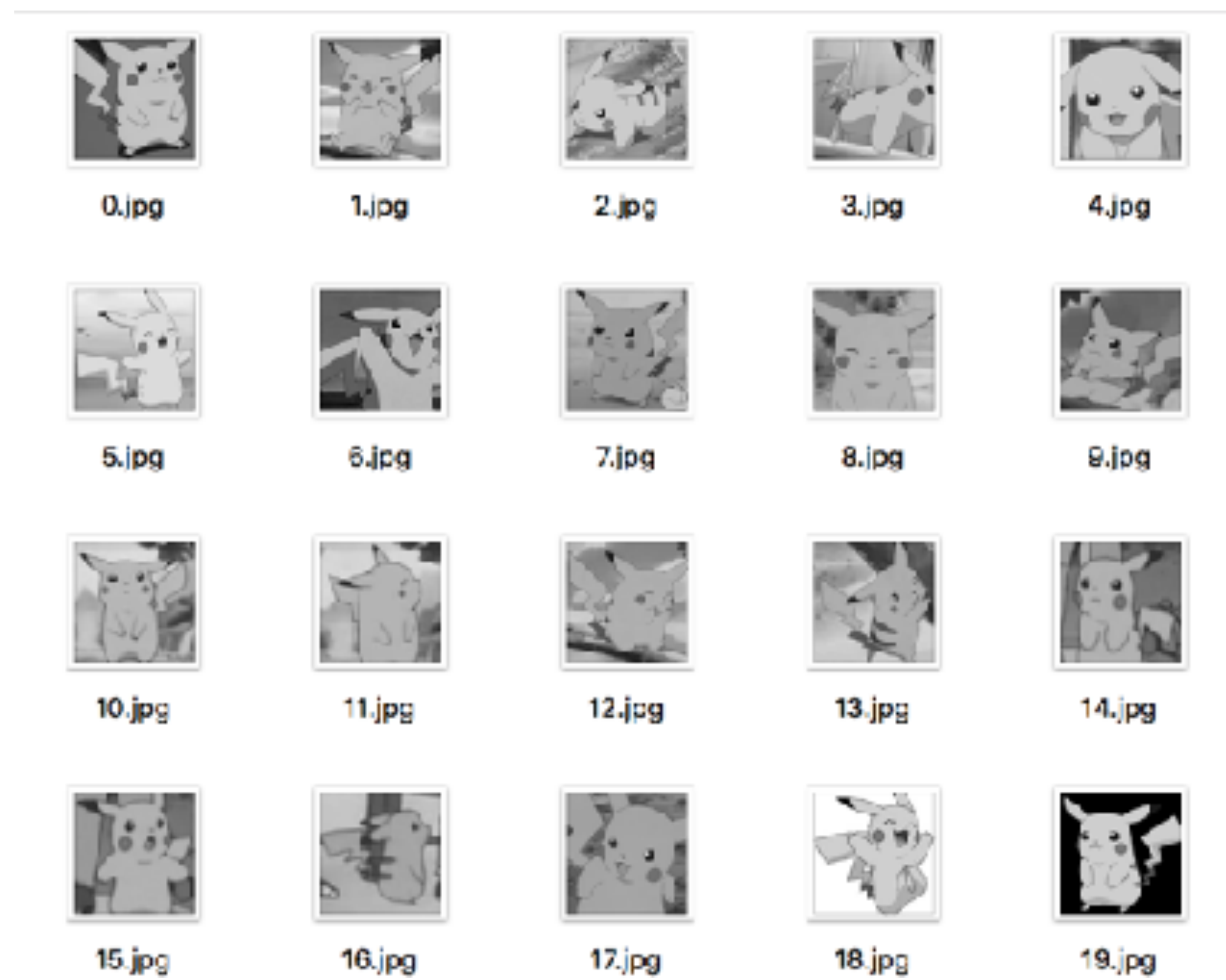
1-1 Data Preprocessing

Checkpoint 1

./pokemon/Pikachu



./pokemon_processed/Pikachu



1-1 Data Preprocessing

- Pull files from server
  ex. copy ~/HCC-ML-LAB/LAB1/pokemon/Pikachu/01.png

```
$ scp nctuece@140.113.146.xxx:~/HCC-ML-LAB/LAB1/pokemon/Pikachu/01.png ./
```

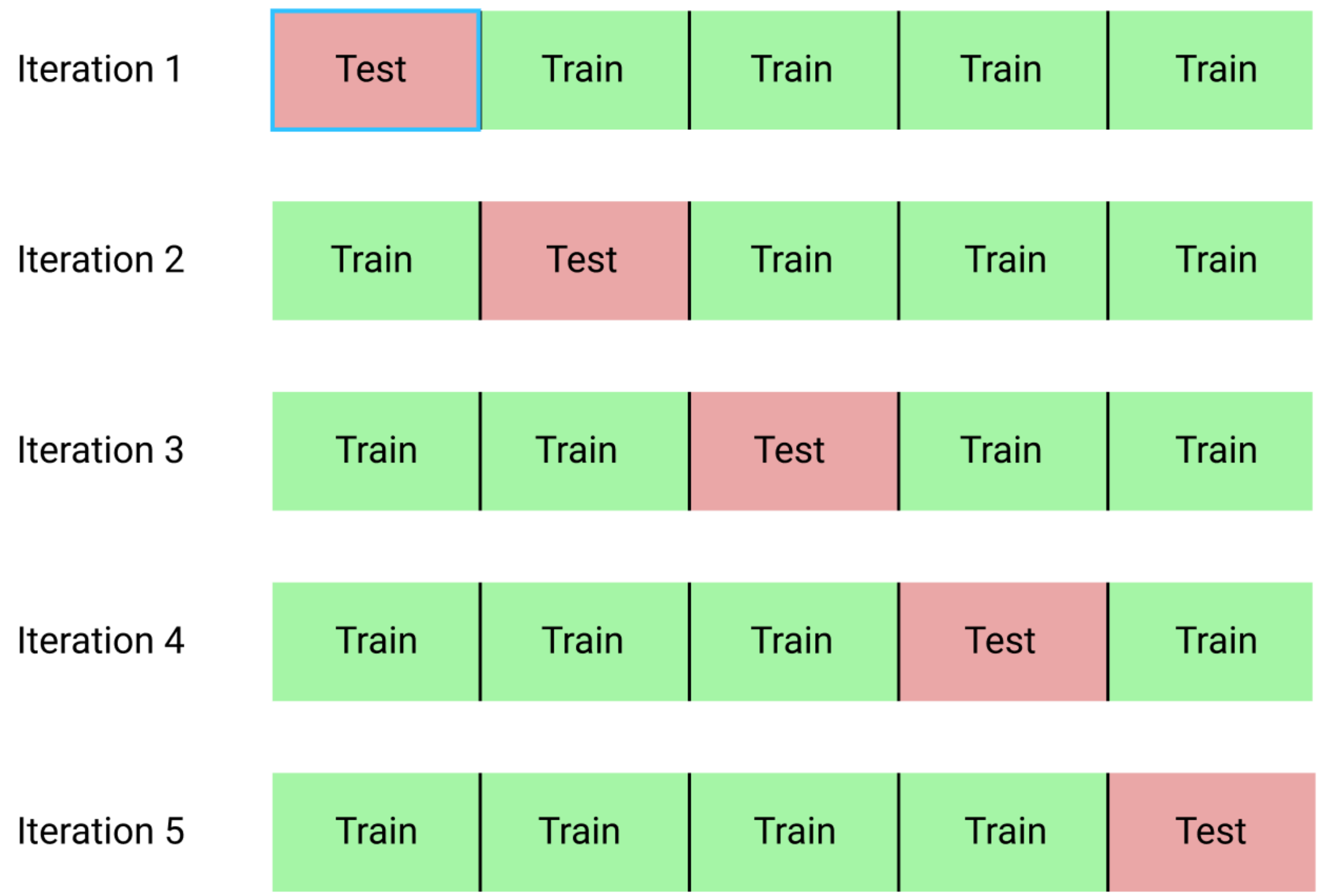
- Push files to server
  ex. Copy LAB1.py to server

```
$ scp ./LAB1.py nctuece@140.113.146.xxx:~/HCC-ML-LAB/LAB1
```

Cross-Validation

- K-fold cross validation:
  資料分割成K組子樣本，每次實驗中，一組單獨的子樣本被保留作為測試，其他K-1個樣本用來訓練。重複K次取平均。

```
from sklearn.model_selection import KFold
```



Evaluation

	真女	真男
猜女	True Potisive(TP)	False Positives(FP)
猜男	False Negatives (FN)	True Negatives(TN)

Precision

Recall

$$Precision = \frac{T_p}{T_p + F_p}$$

$$Recall = \frac{T_p}{T_p + T_n}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Evaluation

```
from sklearn.metrics import classification_report
```

	precision	recall	f1-score	support
Charizard	1.00	0.67	0.80	6
Gengar	1.00	1.00	1.00	2
Pikachu	0.62	1.00	0.77	5
Tangela	1.00	0.80	0.89	5
micro avg	0.83	0.83	0.83	18
macro avg	0.91	0.87	0.86	18
weighted avg	0.90	0.83	0.84	18

```
from sklearn.metrics import confusion_matrix
```

	Charizard	Gengar	Pikachu	Tangela
Charizard	4	0	2	0
Gengar	0	2	0	0
Pikachu	0	0	5	0
Tangela	0	0	1	4

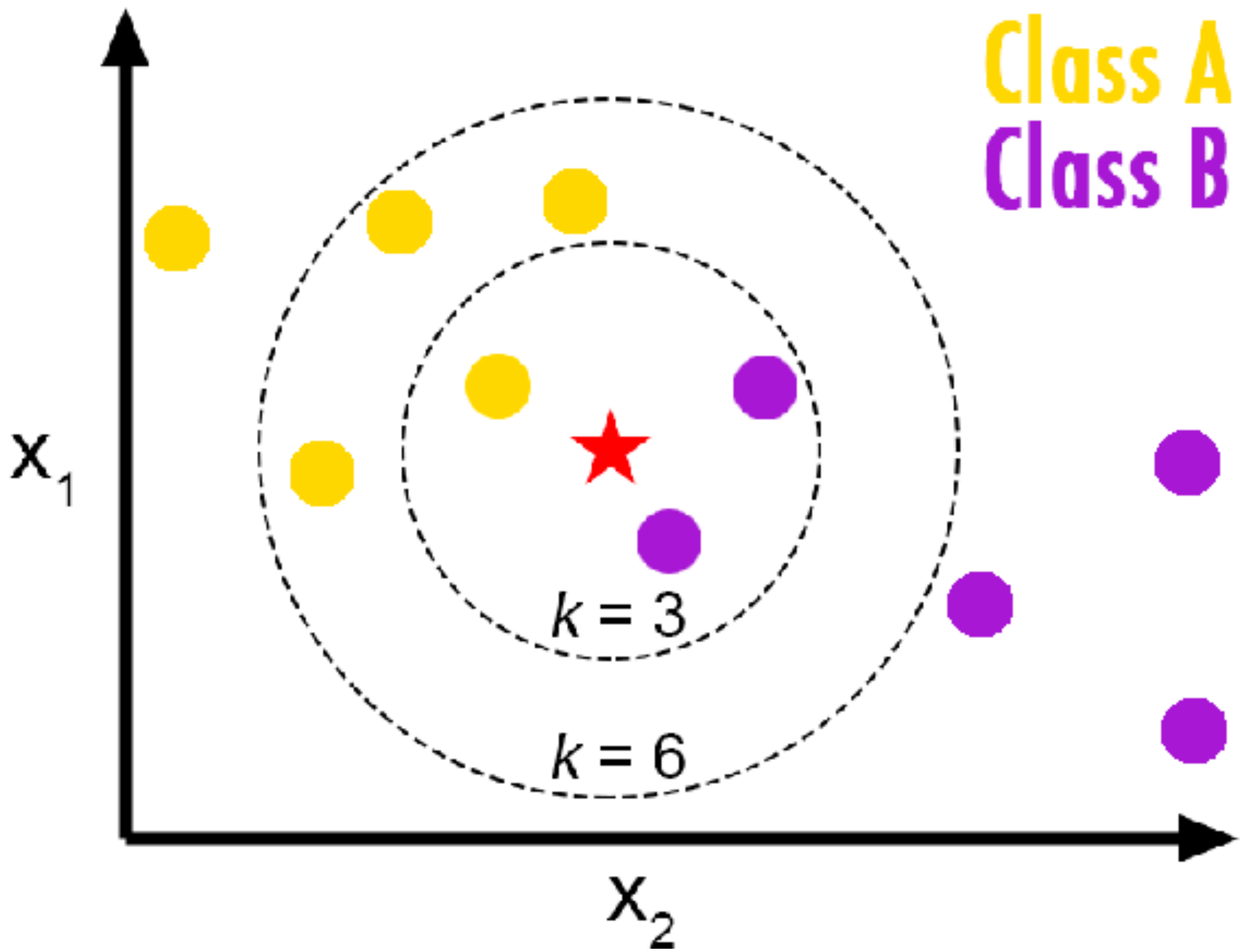
Evaluation

```
from sklearn.metrics import precision_score, recall_score
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
```

	precision	recall	f1-score	support
Charizard	1.00	0.40	0.57	5
Gengar	0.71	1.00	0.83	5
Pikachu	0.67	1.00	0.80	4
Tangela	0.80	0.67	0.73	6
micro avg	0.75	0.75	0.75	20
macro avg	0.80	0.77	0.73	20
weighted avg	0.80	0.75	0.73	20

- Micro
  所有類別一起累加統計TP, TN, FP, FN
- Macro
  各類別的結果平均值
- Weighted
  以樣本數量為權重，加權平均版本的Macro

1-2 KNN



1-2 KNN

LAB1.py

```
# TODO: Checkpoint 2, Train an KNN classification model
# 1. Select appropriate paramter for GridSearchCV
print("Fitting KNN to the training set")
param_grid = {
    'n_neighbors': [1, ???]
}
clf = GridSearchCV(KNeighborsClassifier(), param_grid, cv=3, iid=False)
clf = clf.fit(X_train, y_train)
```

KNeighborsClassifier:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

GridSearchCV:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Checkpoint 2 Output:

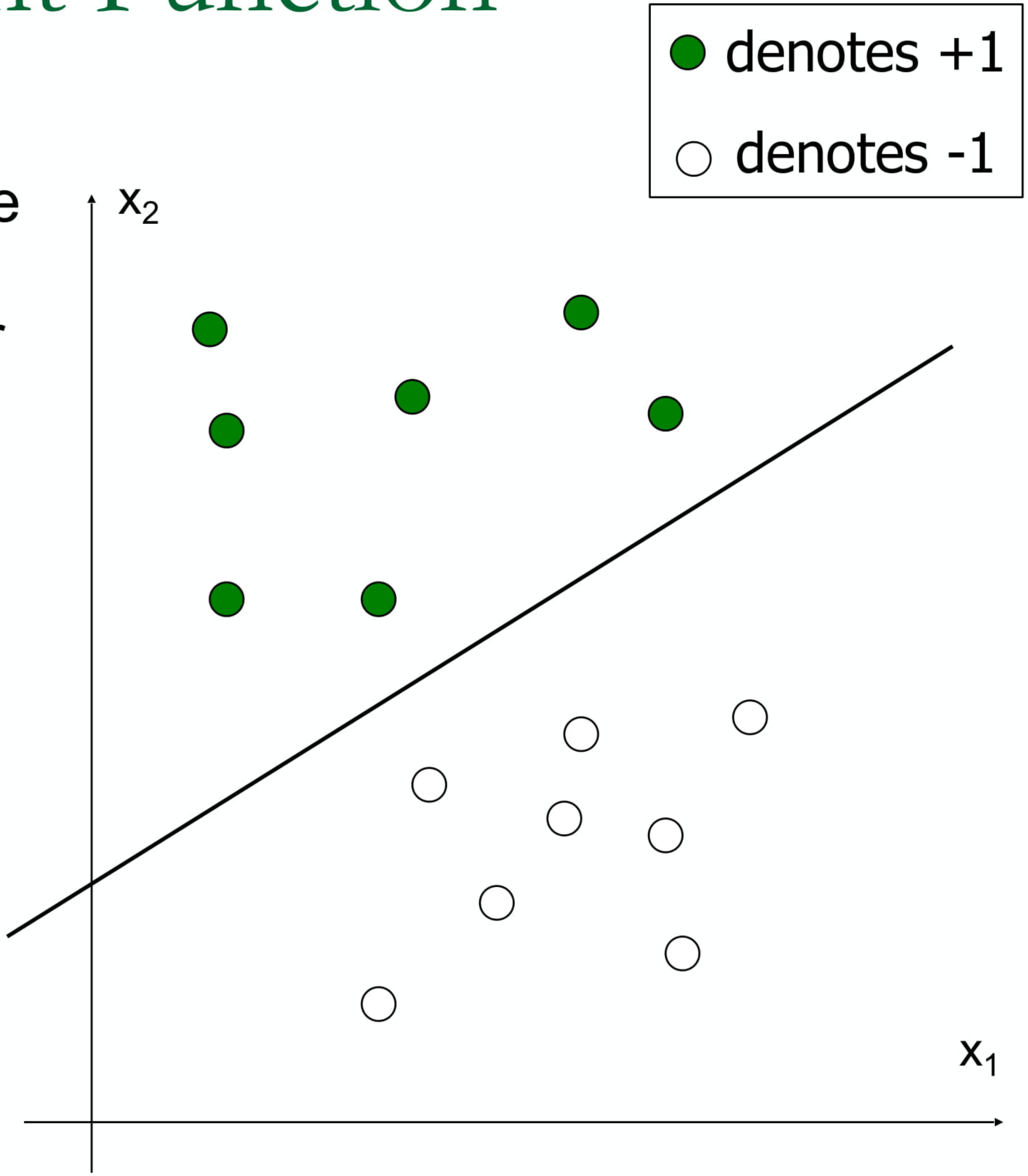
```
KFold average
precision: 0.8959
recall   : 0.8625
```

1-3 SVM

Linear Discriminant Function

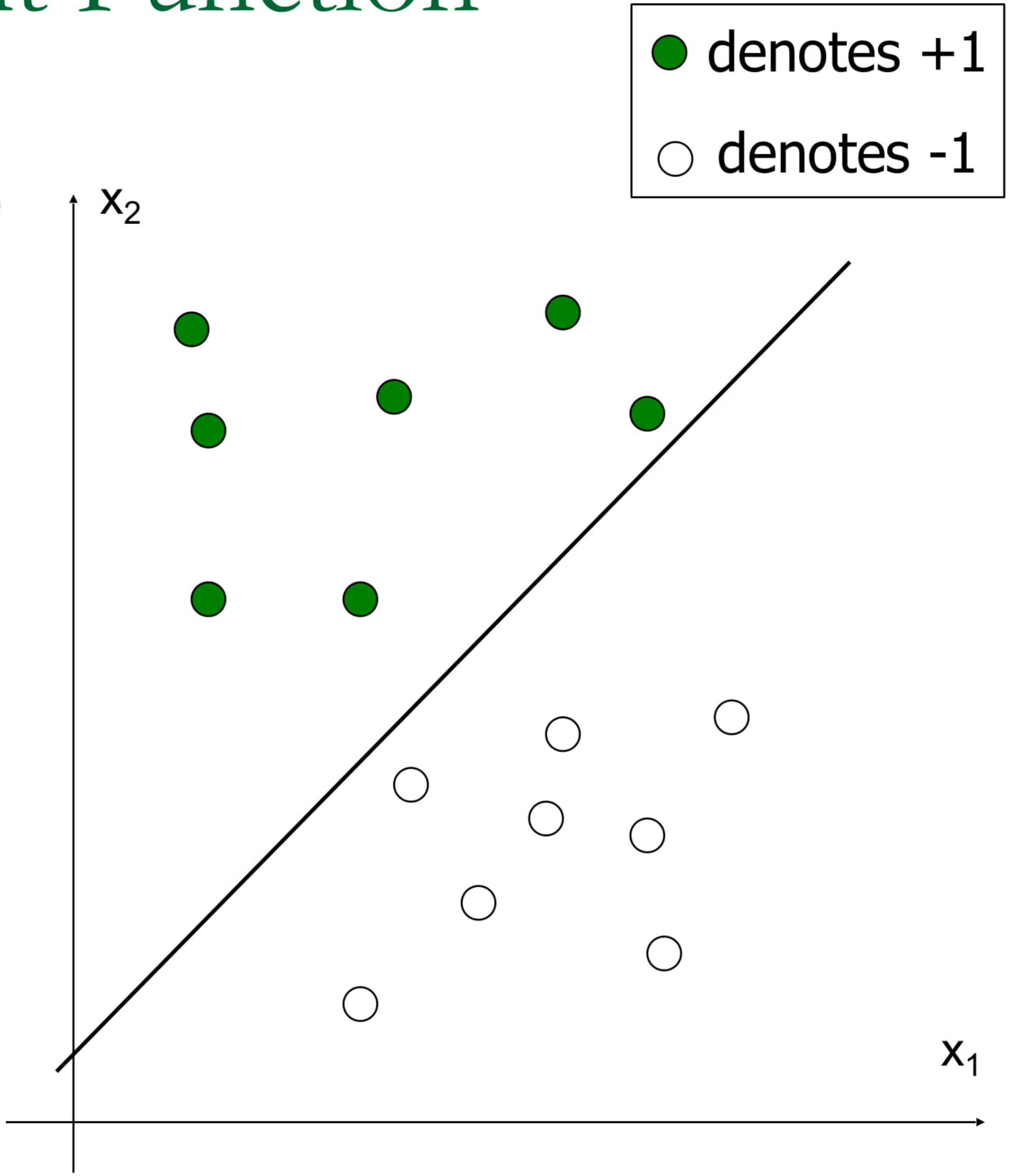
- How would you classify these points using a linear discriminant function in order to minimize the error rate?

- Infinite number of answers!

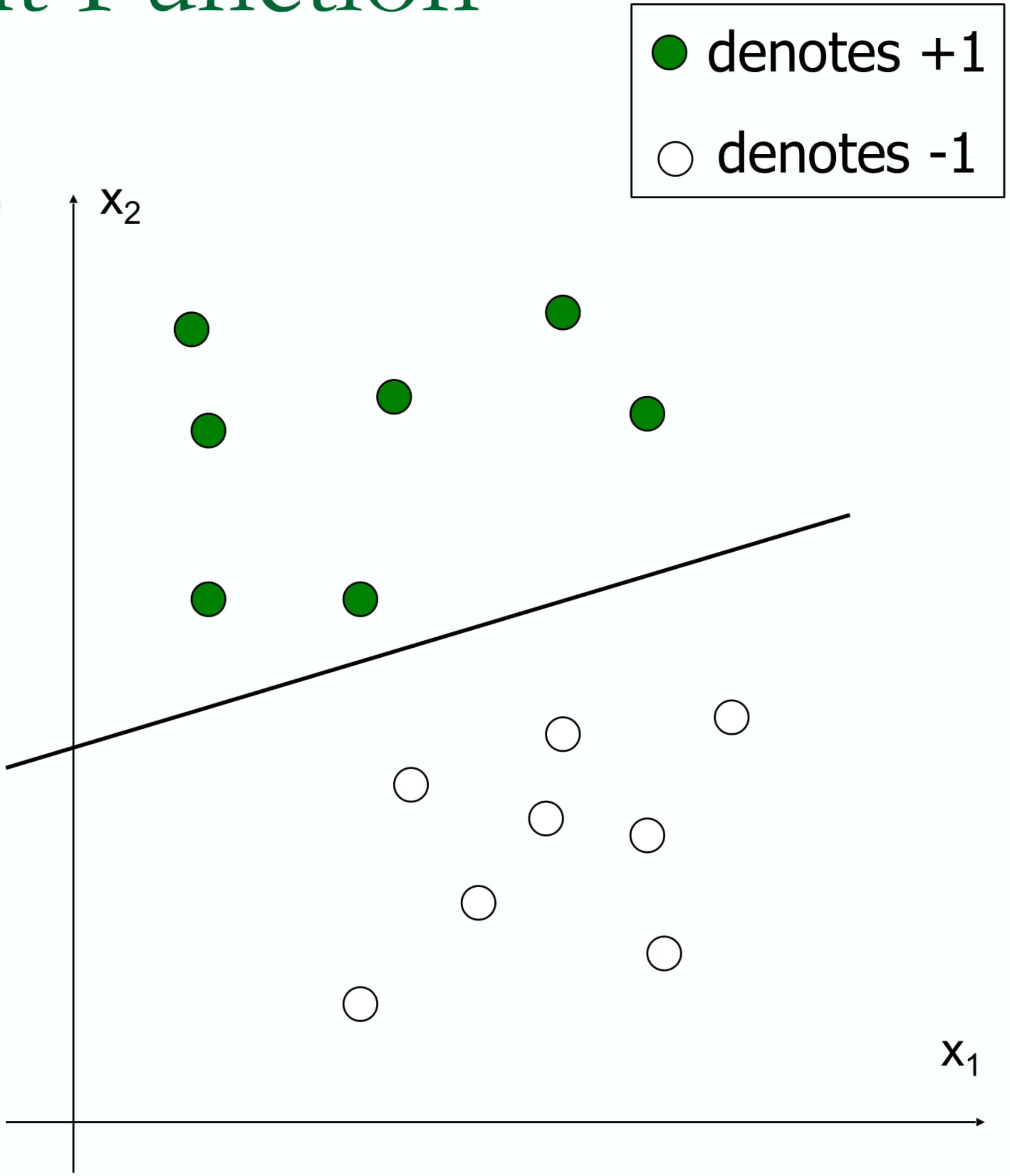


Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!



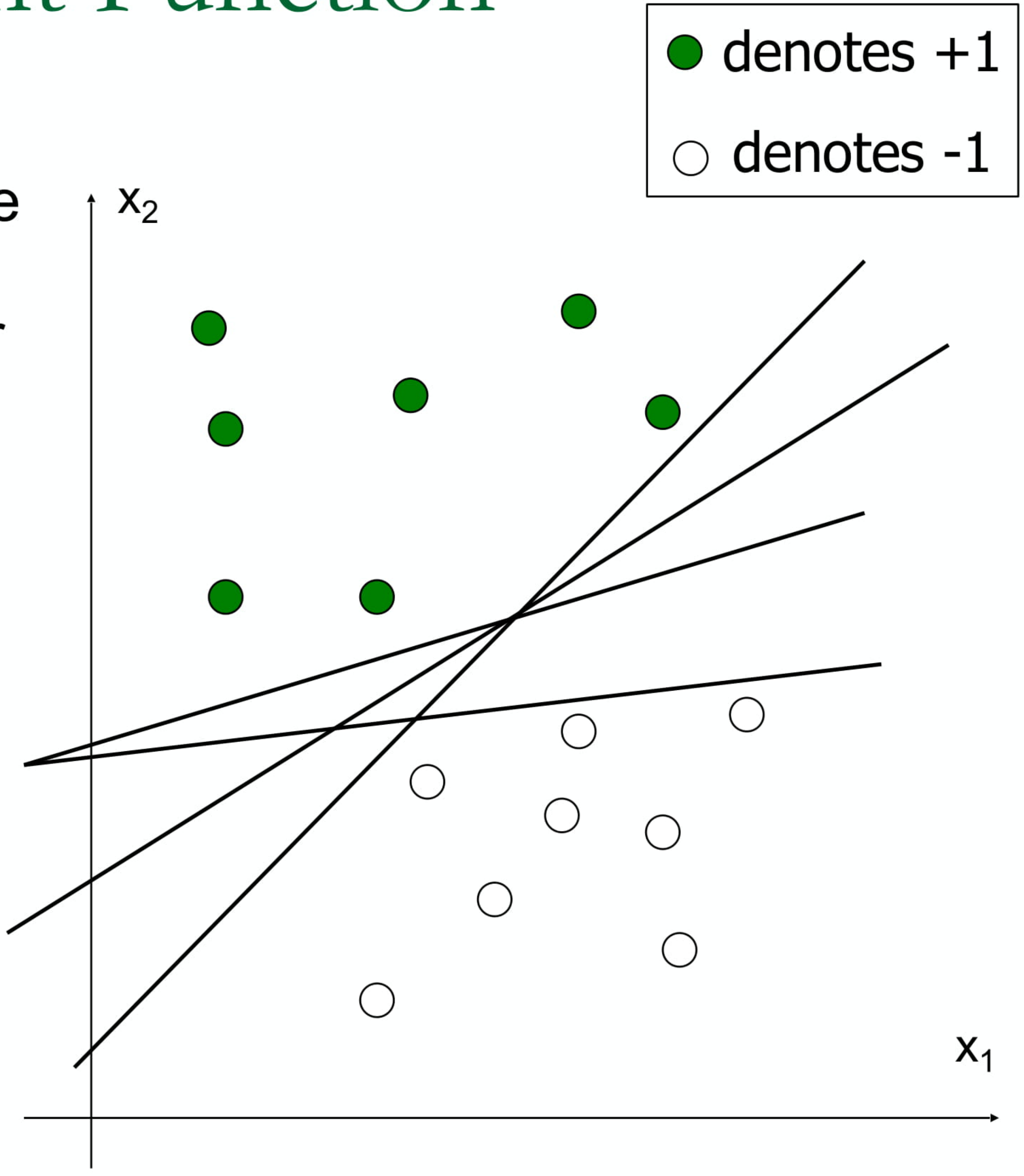
Linear Discriminant Function



- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!

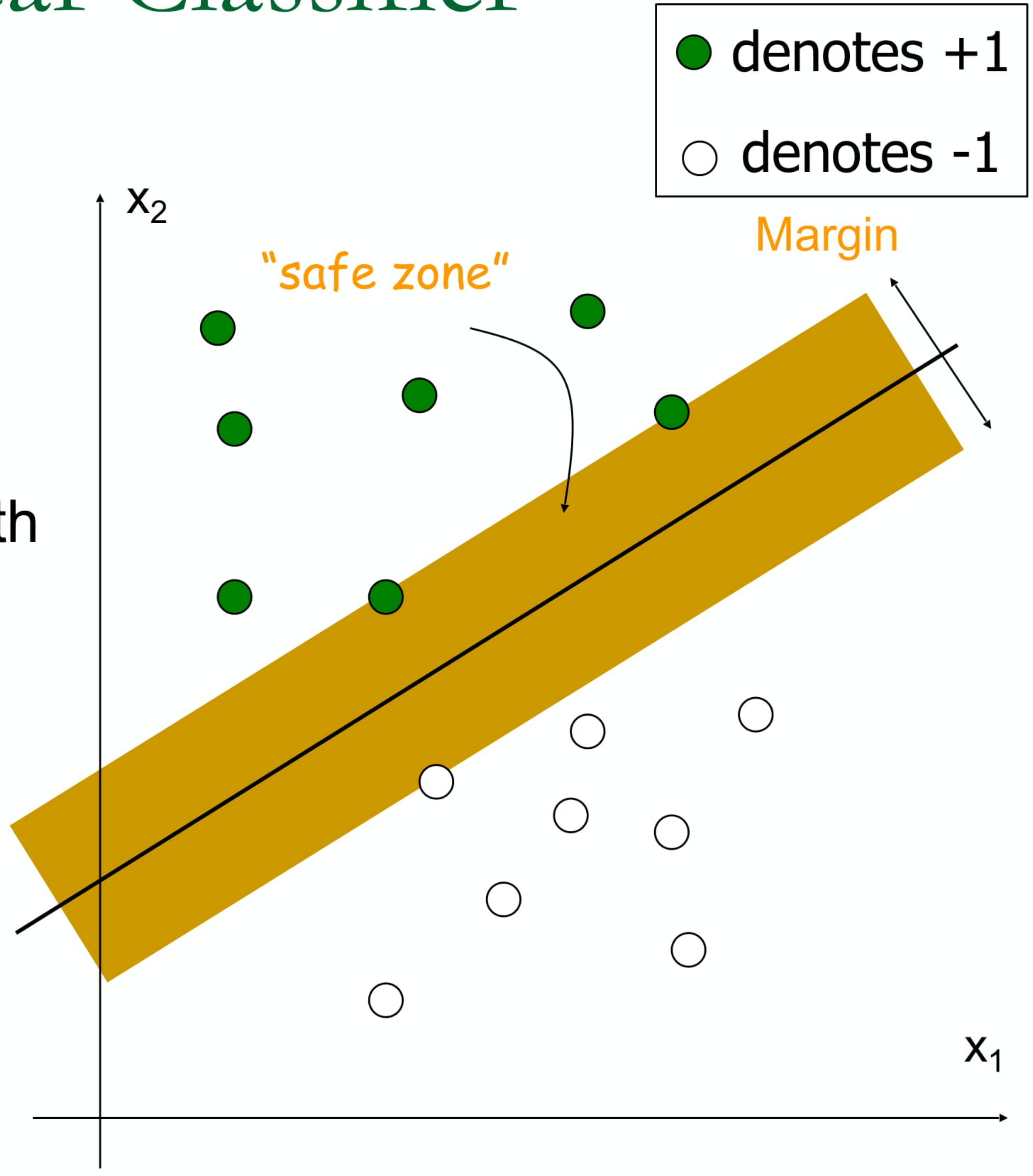
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!
- Which one is the best?



Large Margin Linear Classifier

- The linear discriminant function (classifier) with the maximum margin is the best
- Margin is defined as the width that the boundary could be increased by before hitting a data point
- Why it is the best?
  - Robust to outliners and thus strong generalization ability



Linear Discriminant Function

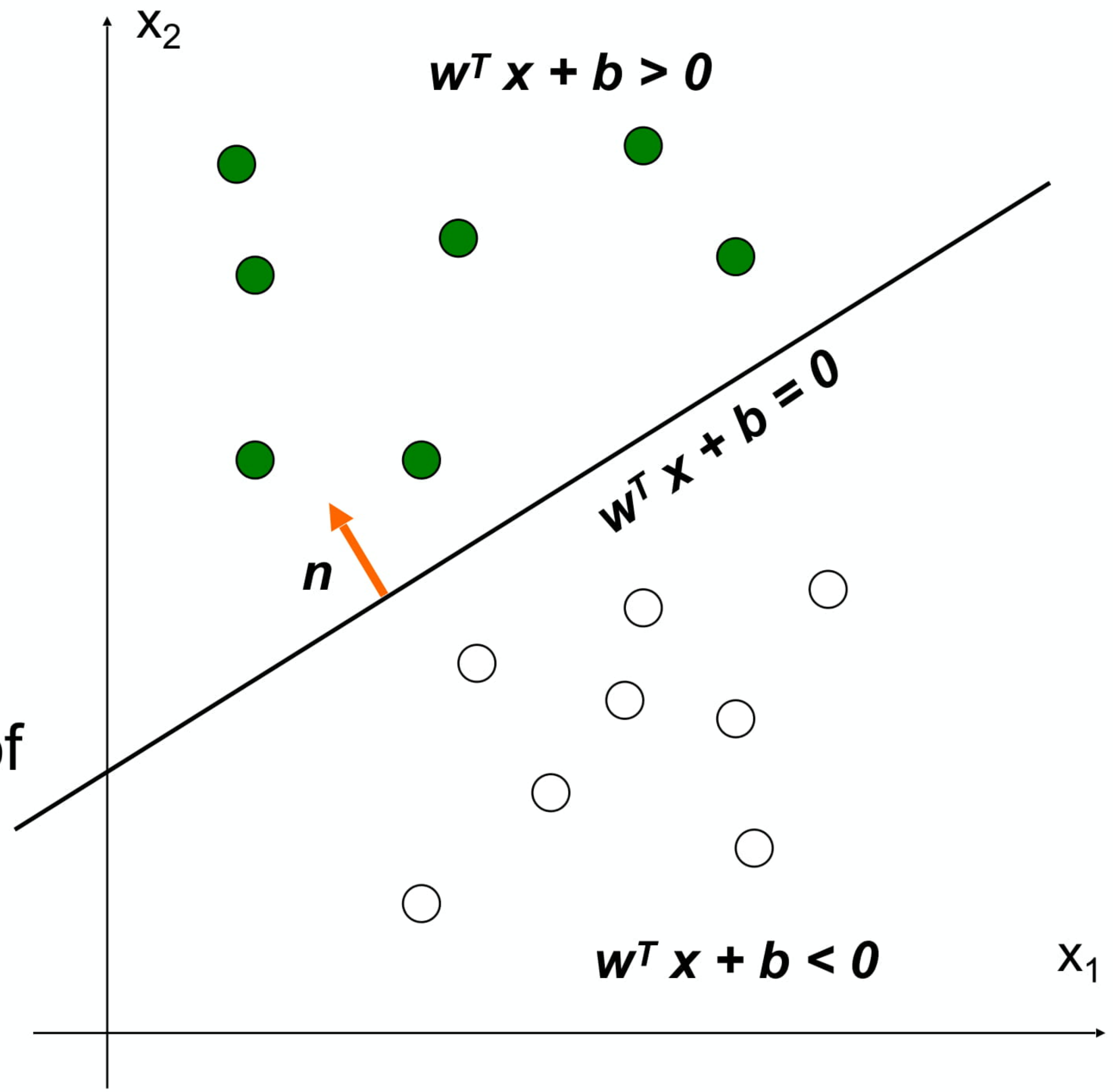
- g(x) is a linear function:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

- A hyper-plane in the feature space

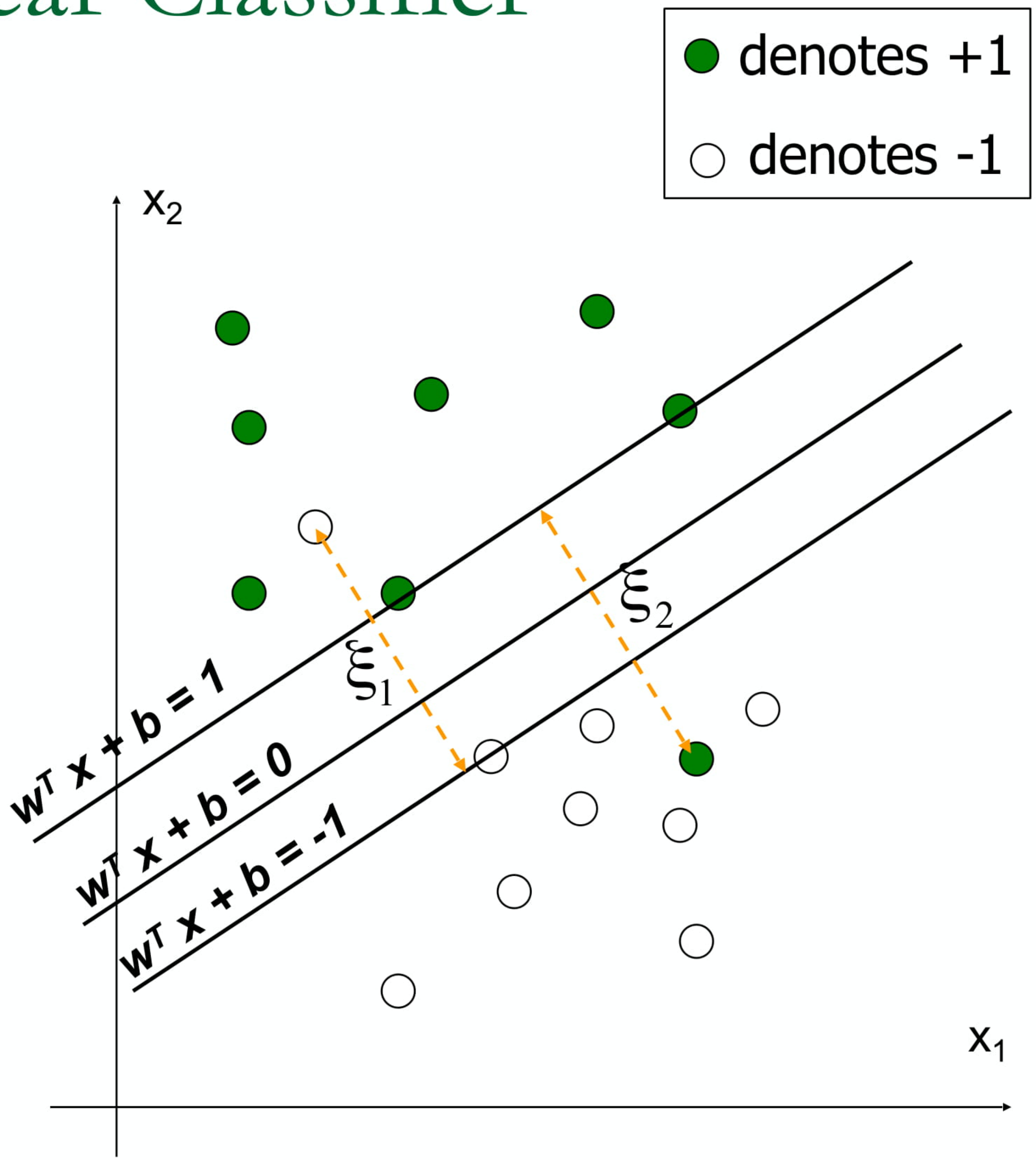
- (Unit-length) normal vector of the hyper-plane:

$$\mathbf{n} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$



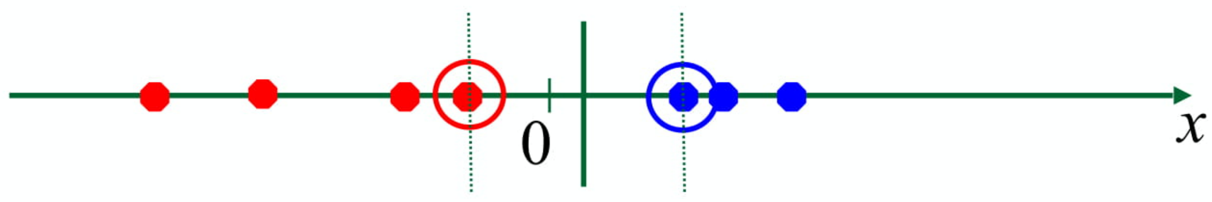
Large Margin Linear Classifier

- What if data is not linear separable? (noisy data, outliers, etc.)
- Slack variables ξ_i can be added to allow mis-classification of difficult or noisy data points



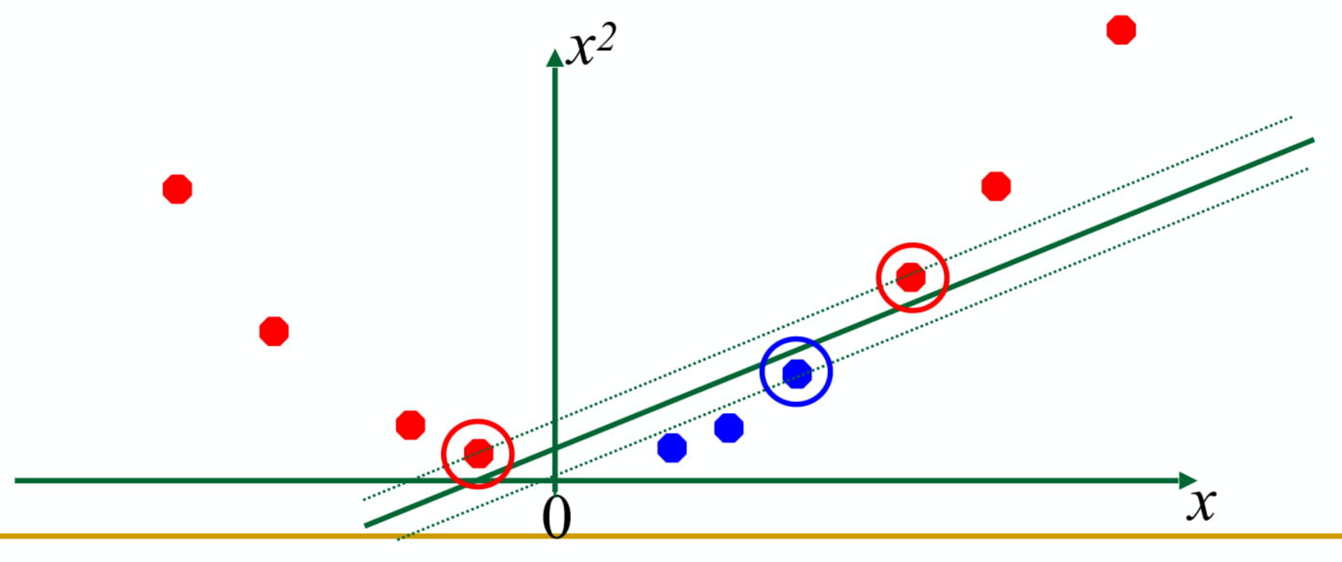
Non-linear SVMs

- Datasets that are linearly separable with noise work out great:



- But what are we going to do if the dataset is just too hard?

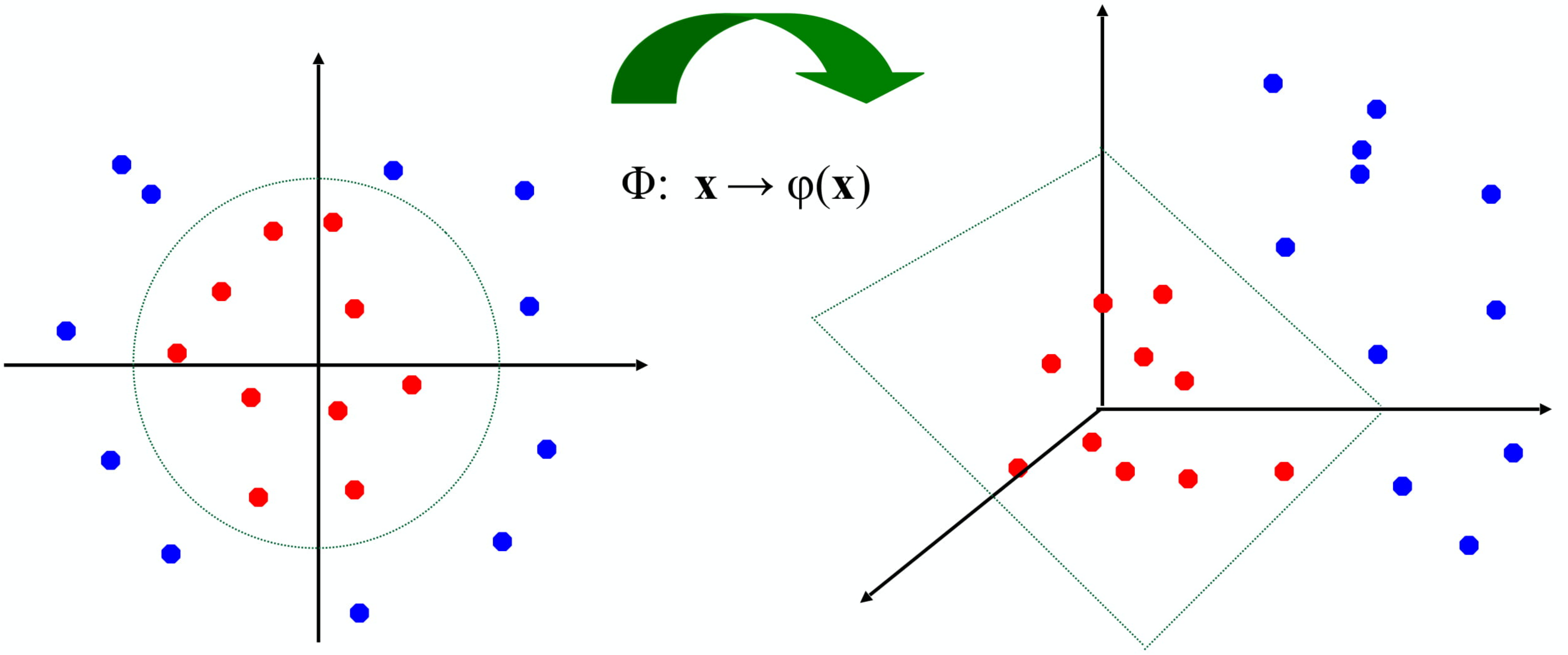
- How about… mapping data to a higher-dimensional space:



This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Non-linear SVMs: Feature Space

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:



This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

  - Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

- In general, functions that satisfy *Mercer's condition* can be kernel functions.

1-3 SVM

LAB1.py

```
# TODO: Checkpoint 3, Train a SVM classification model
# 1. Select appropriate paramter for GridSearchCV
####
print("Fitting SVM to the training set")
param_grid = {
        'kernel': ['rbf', 'linear'],
        'C': [???],
        'gamma': [???],
}
clf = GridSearchCV(SVC(class_weight='balanced'), param_grid, cv=3, iid=False)
clf = clf.fit(X_train, y_train)
print('Best params', clf.best_params_)
```

SVC: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Checkpoint 3
Output:

```
KFold average
precision: 0.9223
recall   : 0.9125
```

1-4 PCA

- Principal Components Analysis
- Data with so many features can easily contain a lot of noisy ones.
- Would it be better if we could select just ones which really capture the trends and the patterns in our data? Here's where PCA comes into play!

1-4 PCA

LAB1.py

```
n_components = ??????
pca = PCA(n_components=n_components, whiten=True).fit(X_train)
eigenpokemons_titles = [
    "eigenpokemon %d" % i
    for i in range(pca.components_.shape[0])]
plot_gallery(pca.components_, eigenpokemons_titles, "PCA", height, width)
print("Projecting the input data on the eigenpokemon orthonormal basis")
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

PCA: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

1-4 PCA

Checkpoint 4
1. Output KFold average
2. PCA.png

```
KFold average
precision: 0.9189
recall   : 0.9125
```