Support Vector Machines in Finance
A gentle introduction

Prof. Dr. Jörg Osterrieder
Prof. Dr. Branka Hadji Misheva

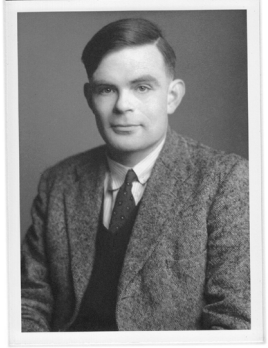
Pre-reading

Forbes

Oct 20, 2018, 11:15am EDT

The Machine Learning Revolution: How Artificial Intelligence Could Transform Your Business

Turing 1950

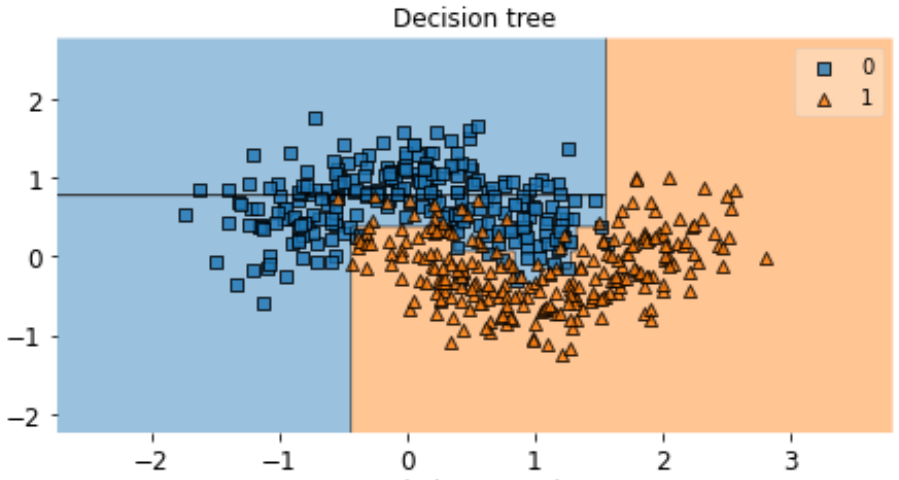


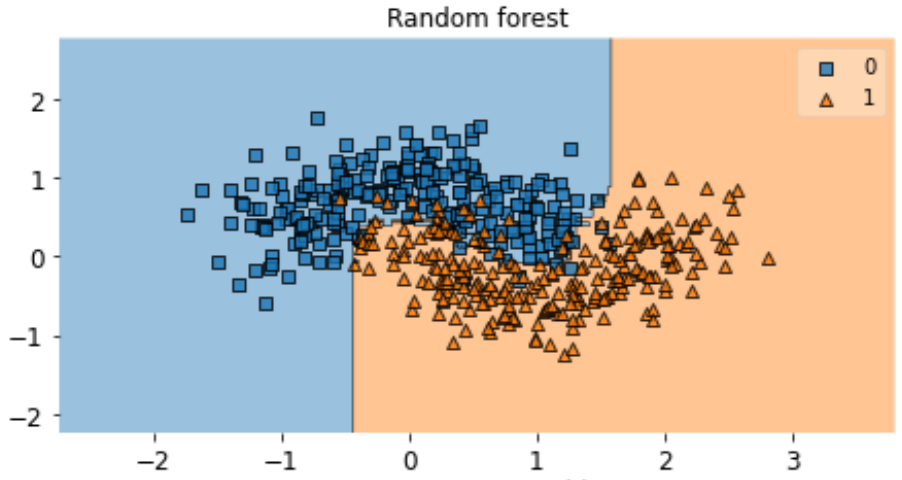
Machine Learning is an application of artificial intelligence where a computer/machine learns from the past experiences (input data) and makes future predictions. The performance of such a system should be at least human level.

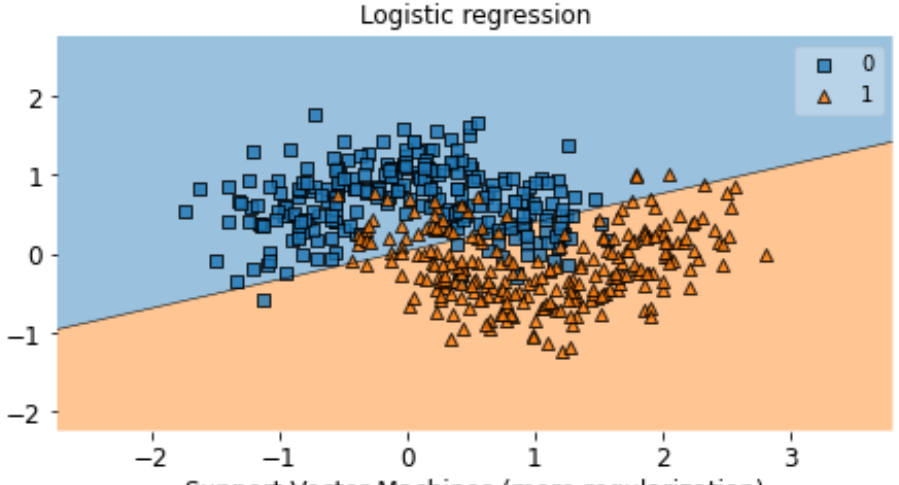
Mitchell 1997

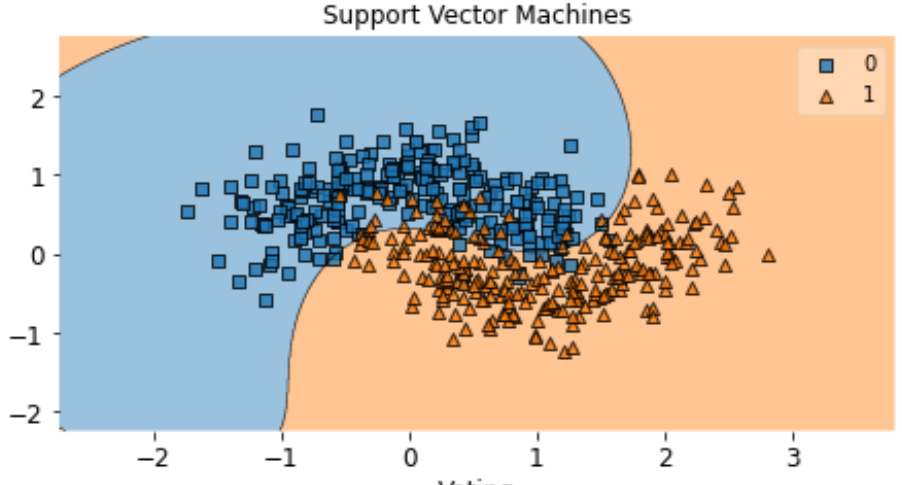
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

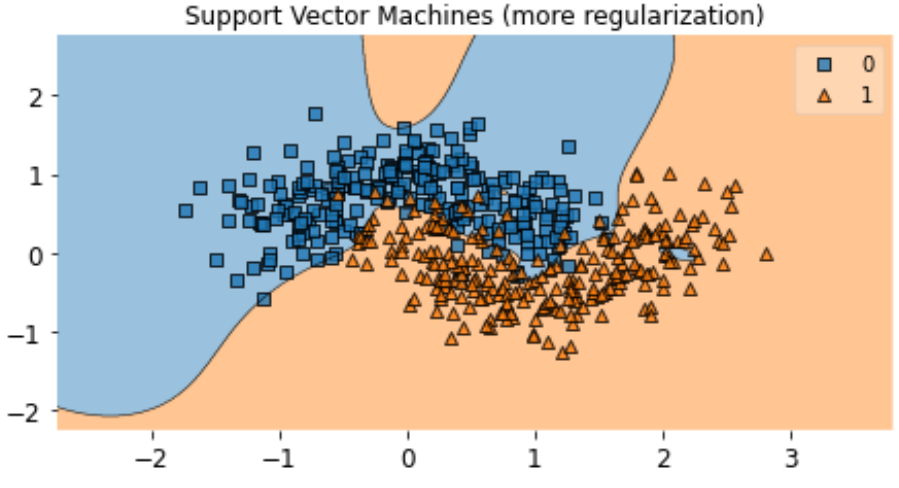
A handwriting recognition learning problem:
Task T: recognizing and classifying handwritten words within images
Performance measure P: percent of words correctly classified, accuracy
Training experience E: a data-set of handwritten words with given classifications

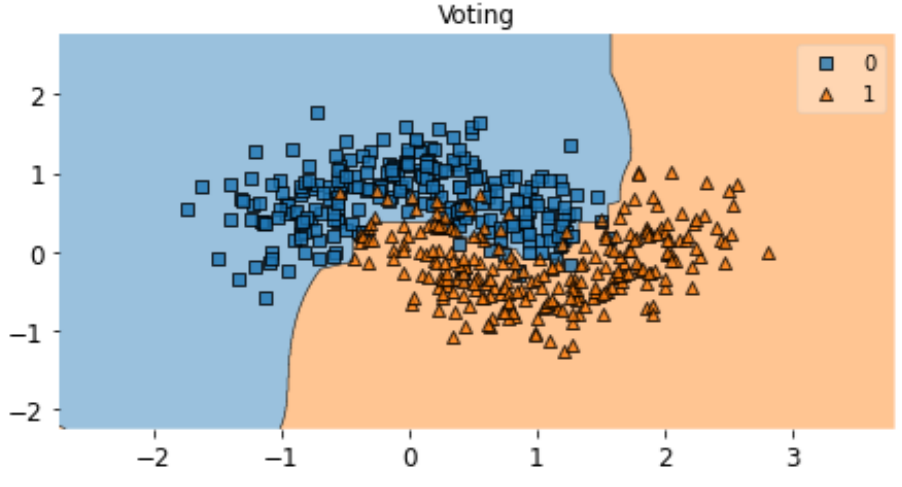

Decision tree
0
1


Random forest
0
1


Logistic regression
0
1


Support Vector Machines
0
1

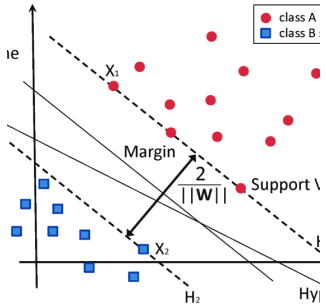

Support Vector Machines (more regularization)
0
1


Voting
0
1

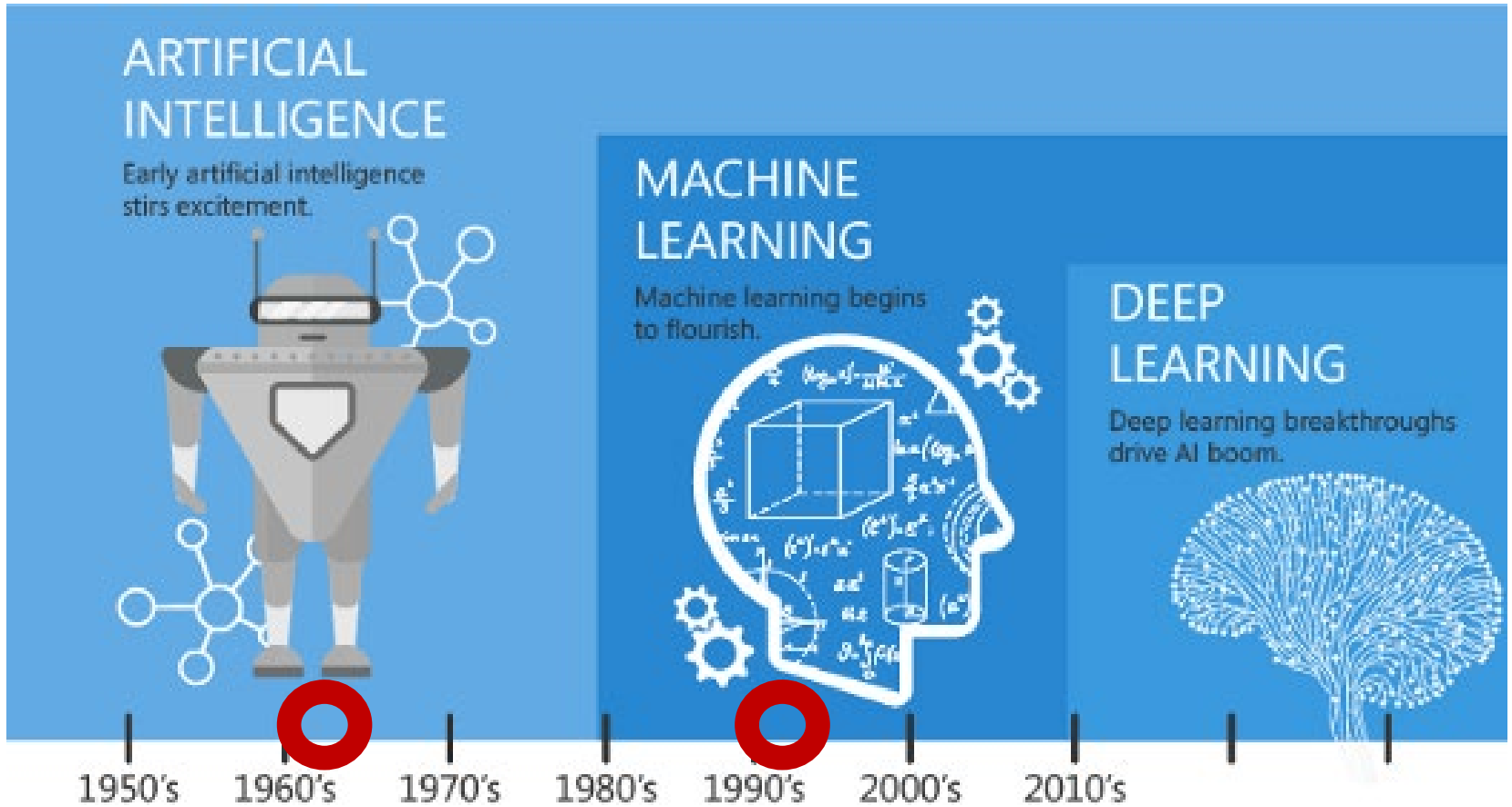
Agenda

AI in Finance

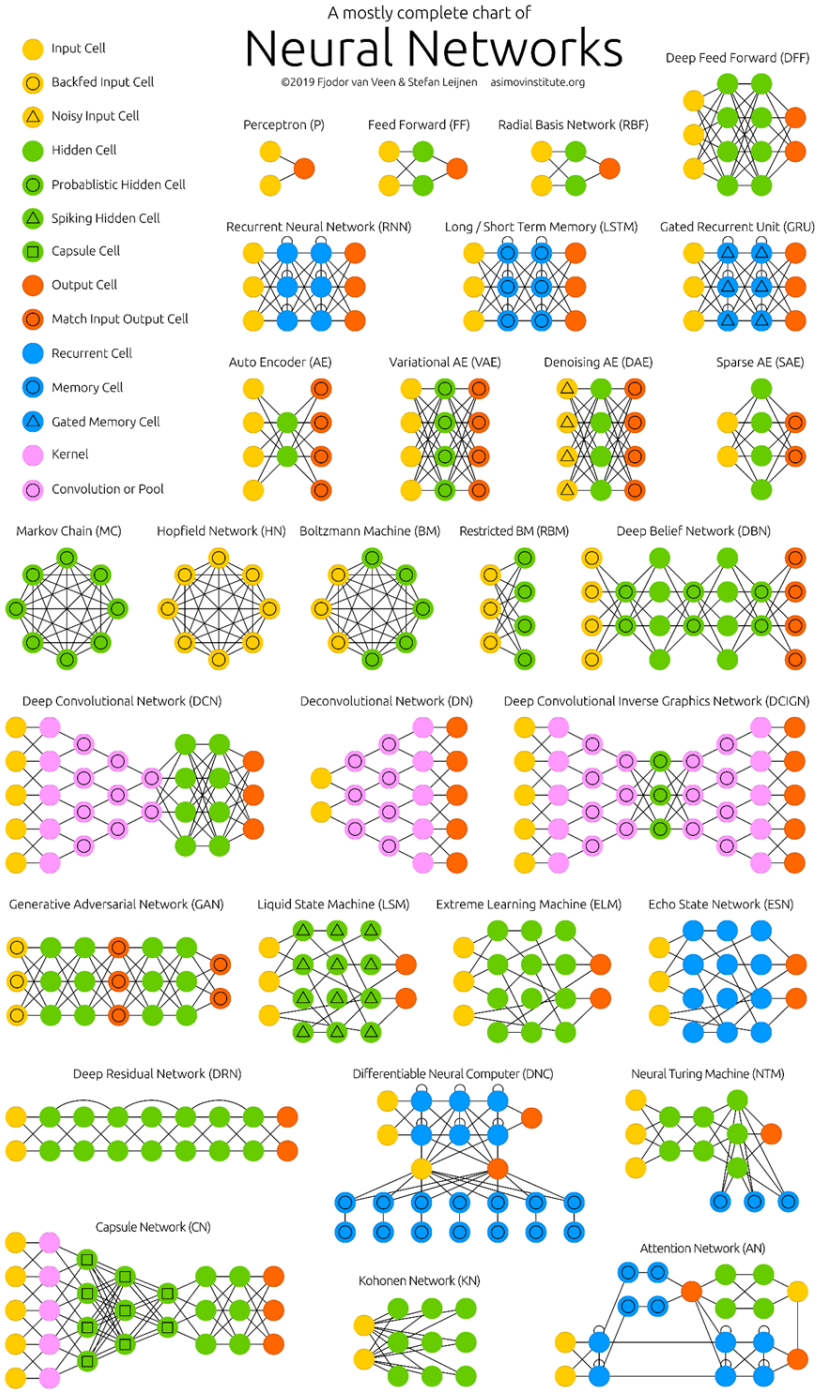
Why AI?



Support Vector Machines



Support Vector Machines[1]



[1] Vladimir Vapnek et al (1963, 1992, 1993, 1997)

Artificial intelligence and Machine Learning is everywhere in Finance

Credit Scoring

Corporate credit ratings[1]

Retail loan applications[2]

Fraud Detection

Unusual patterns[3]

Trade execution

Optimal trade execution policies

Minimize market impact

Double deep Q-learning[4]

Derivatives hedging

Execute hedge transactions

Reinforcement learning for deep hedging[5]

Portfolio management

Compose and rebalance portfolios of financial instruments[6]

Customer service

Process natural languages

Chat bots[7]

[1] Golbayani et al., 2020, «Application of Deep Neural Networks to Assess Corporate Credit Rating»
[2] Babaev et al., 2019, «E.T.-RNN: Applying Deep Learning to Credit Loan Applications»
[3] Yousefi et al., 2019, «A Comprehensive Survey on Machine Learning Techniques and User Authentication Approaches for Credit Card Fraud Detection»
[4] Ning et al., 2020, «Double Deep Q-Learning for Optimal Execution»
[5] Buehler et al., 2019, «Deep Hedging: Hedging Derivatives Under Generic Market Frictions Using Reinforcement Learning»
[6] Lopez de Prado, 2020, «Advances in Financial Machine Learning»
[7] Yu et al., 2020, «AVA: A Financial Service Chatbot based on Deep Bidirectional Transformers»

Machine, deep and reinforcement learning work better than traditional methods

Big Data

Neural networks benefit when being trained on larger data sets

Instability

Financial markets do not follow constant laws as in physics

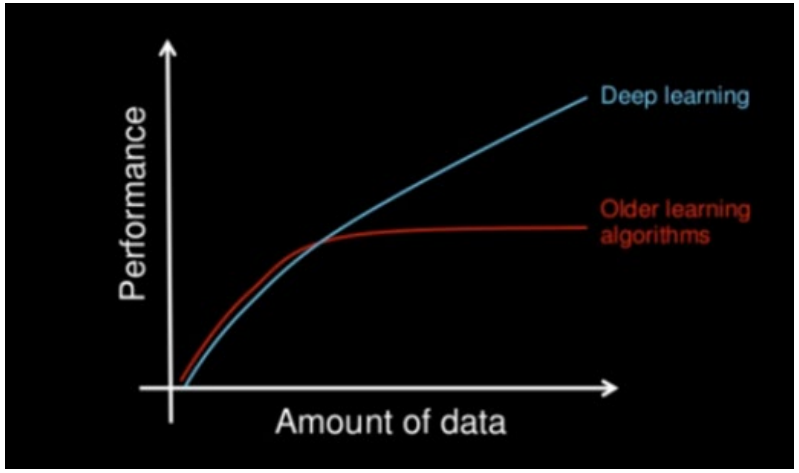
Neural networks use incremental updates through online learning

Non-linearity

Neural networks can deal better with nonlinear relationships

Non-normality

Financial econometrics often assumes normally distributed variables. AI algorithms not so



High dimensionality

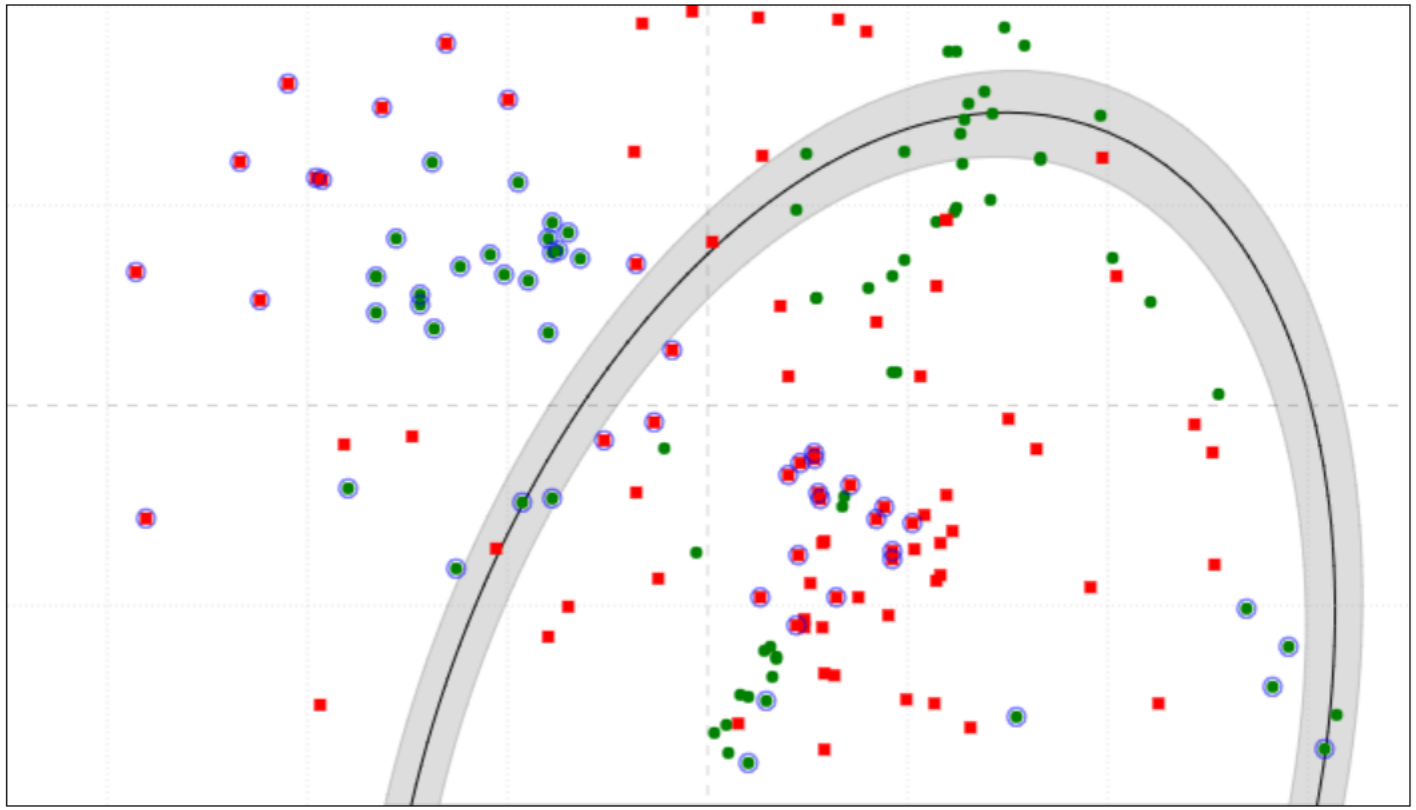
Traditional econometrics usually has a lower number of features (e.g. CAPM); AI algorithms can be very high-dimensional

Classification problem

Traditional econometrics more suitable for estimation (regression) problems; Classification problems are equally as important

An interactive demo

- https://jgreitemann.github.io/svm-demo



Applications of Support Vector Machines

Banking

- In the banking sector, a random forest algorithm is widely used in two main applications. These are for finding loyal customers and finding fraud customers.

Medicine

- In the medicine field, a random forest algorithm is used to identify the correct combination of the components to validate the medicine. Random forest algorithms are also helpful for identifying the disease by analyzing the patient's medical records.

Stock Market

- In the stock market, a random forest algorithm is used to identify the stock behavior as well as the expected loss or profit by purchasing the particular stock.

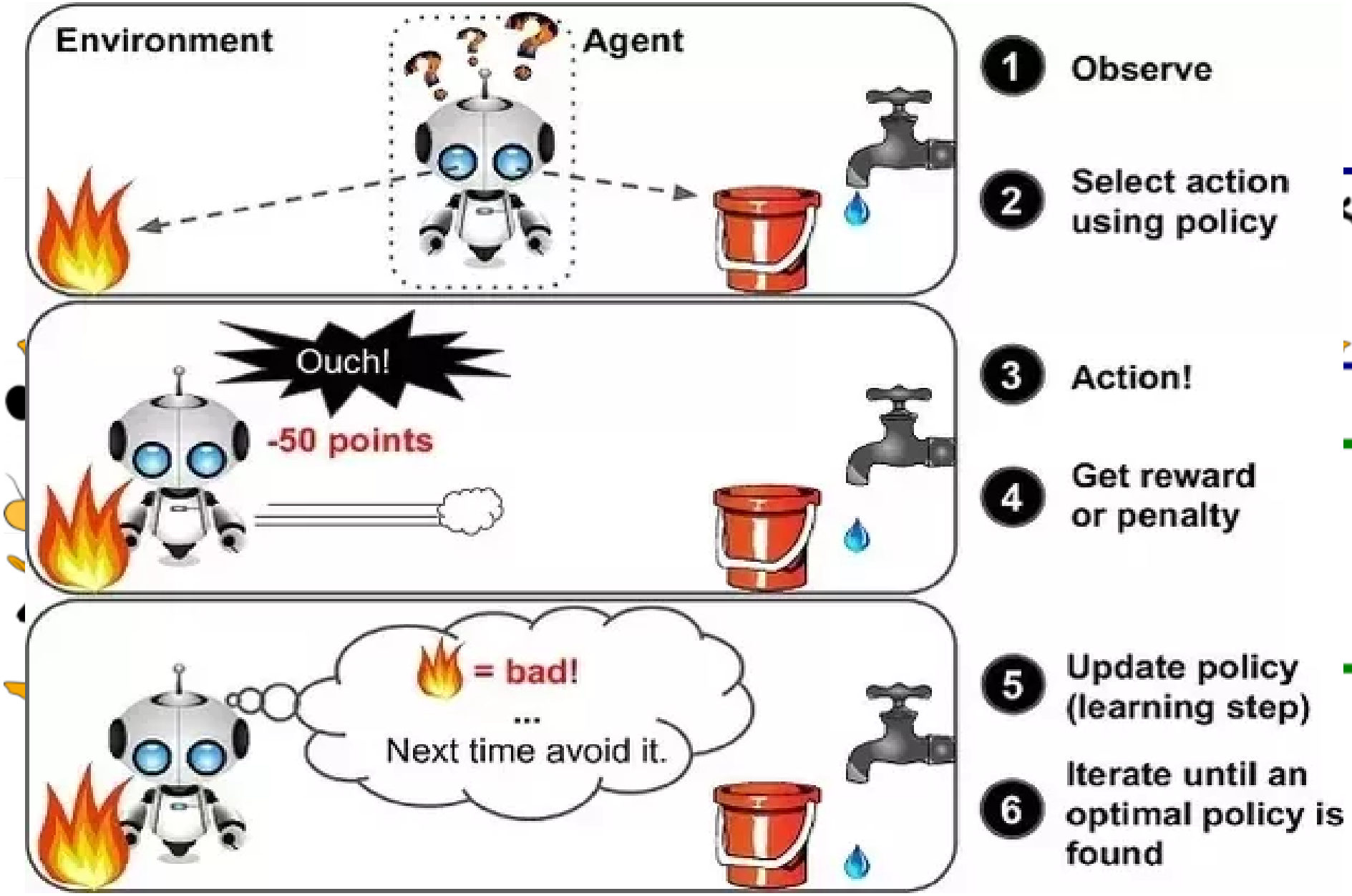
E-commerce

- In e-commerce, the random forest is used in the segment of the recommendation engine for identifying the likelyhood of customers liking the recommend products based on similar kinds of customers.

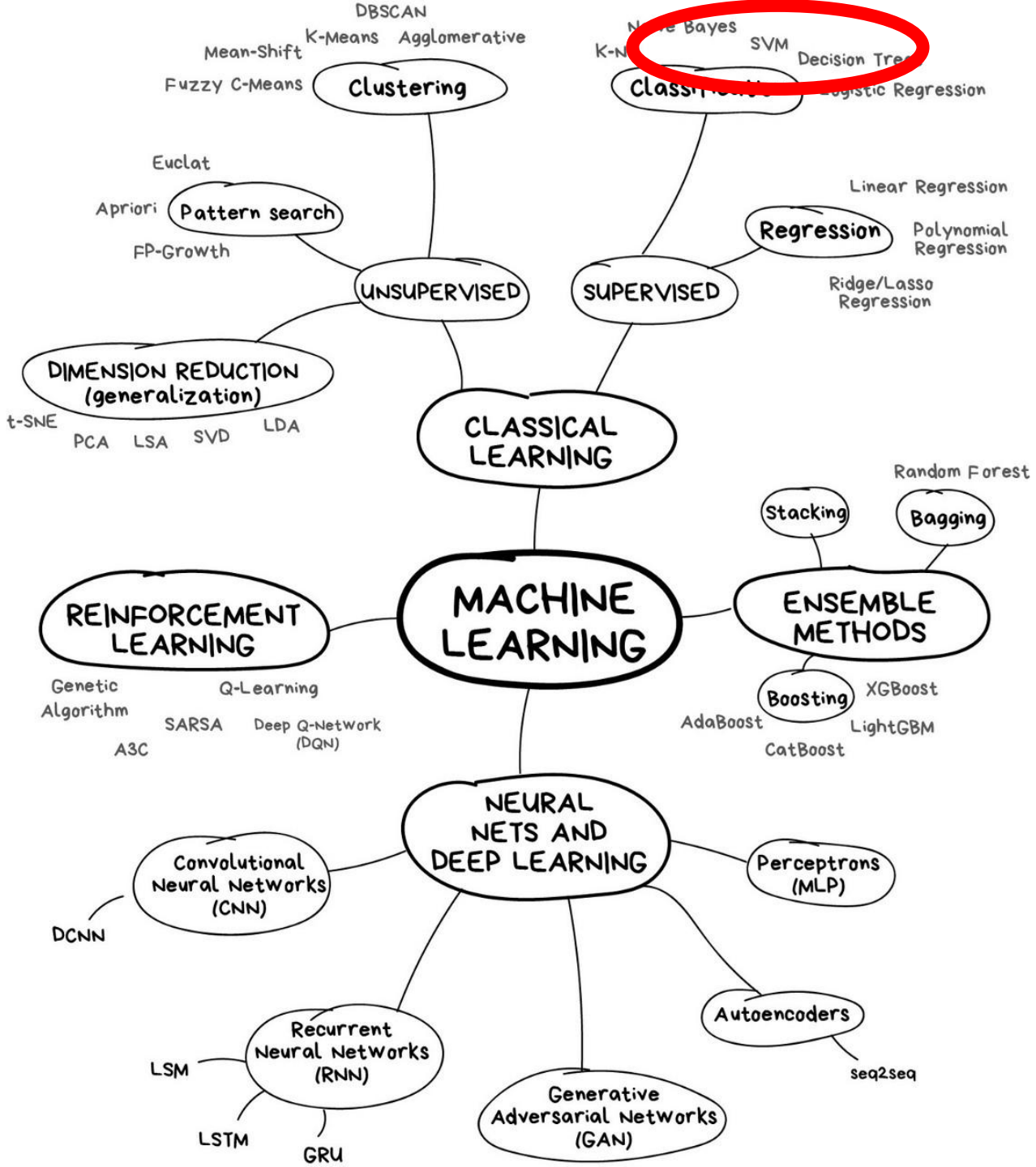
Use cases of Support Vector Machines in Finance

- Stock price prediction
- Derivative pricing
- Investor Risk Tolerance and Robo-Advisors
- Yield Curve Prediction
- Fraud Detection
- Loan Default Probability
- Bitcoin Trading Strategy

A detour – Machine Learning



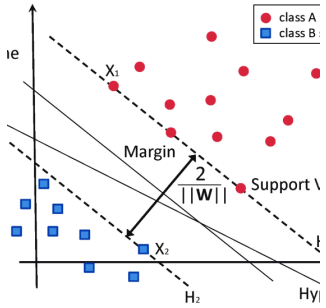
A detour – Machine Learning



Agenda

AI in Finance

Why AI?

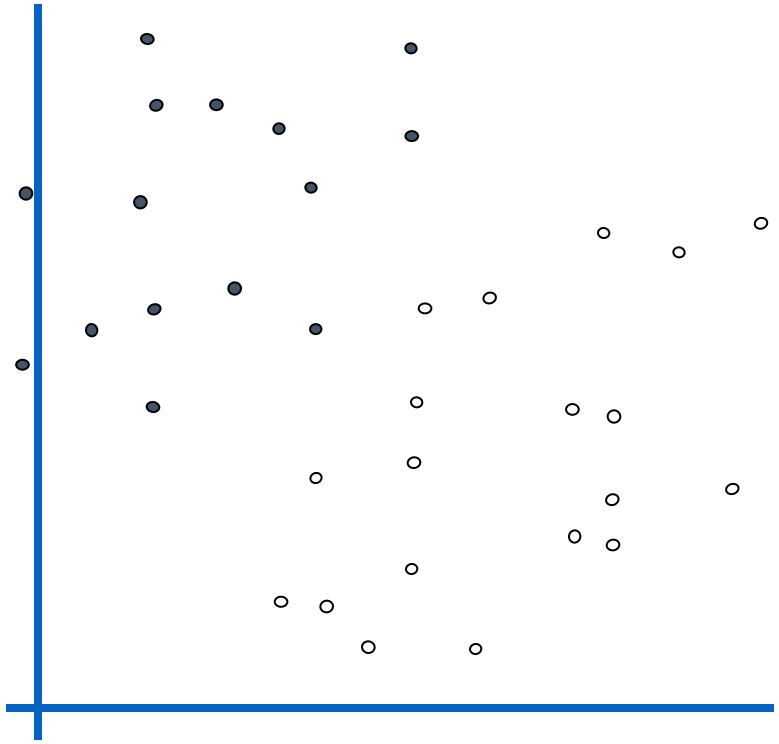


Support Vector Machines

What is a Support Vector Machine?

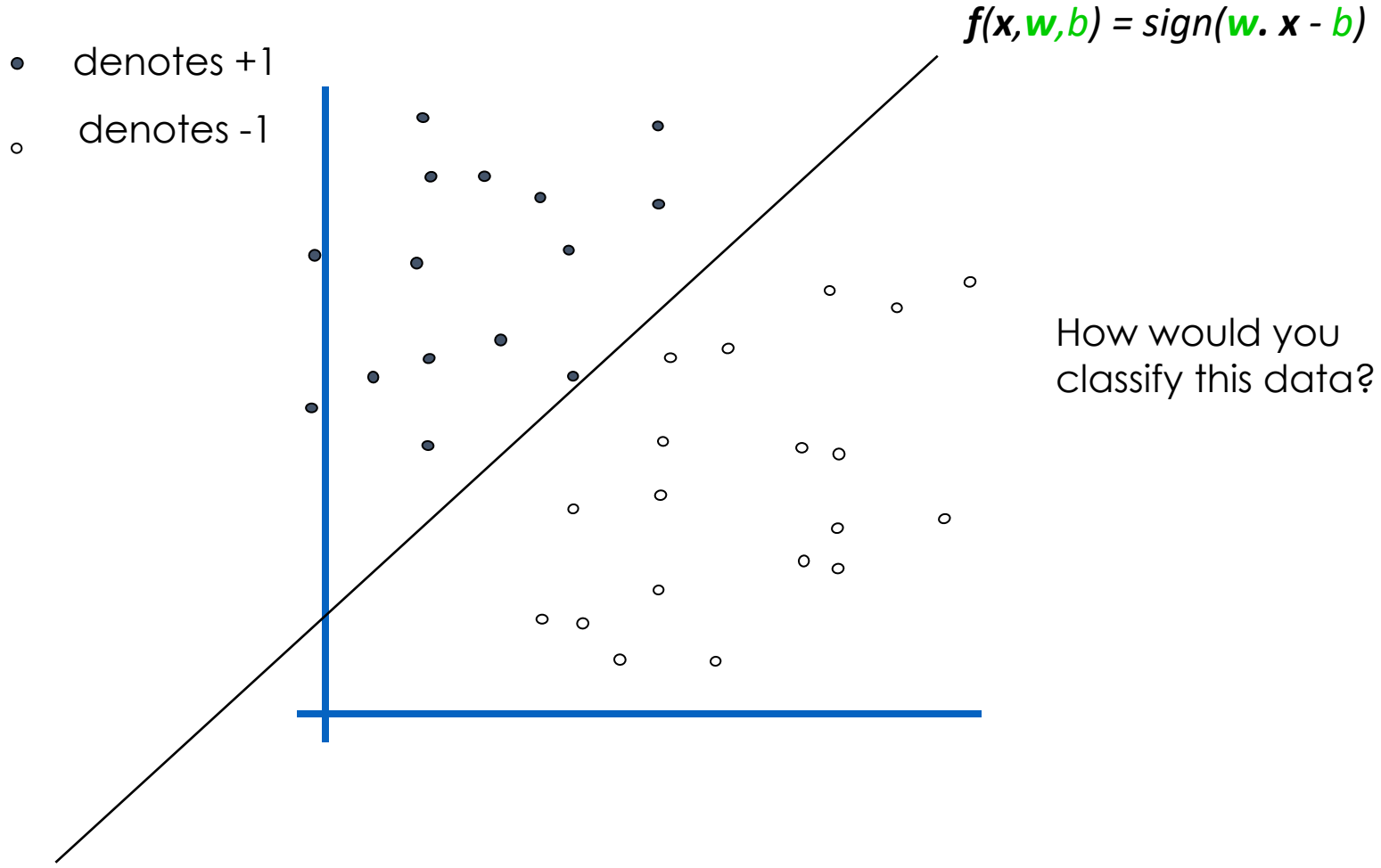
- Supervised machine learning algorithm
- Groups different objects into classes
- Used in classification and regression
- Standard usage for linear classification; use the “kernel trick” for non-linear classification
- Goal: Find widest possible area between different classes

Linear Classifiers



How would you classify this data?

Linear Classifiers

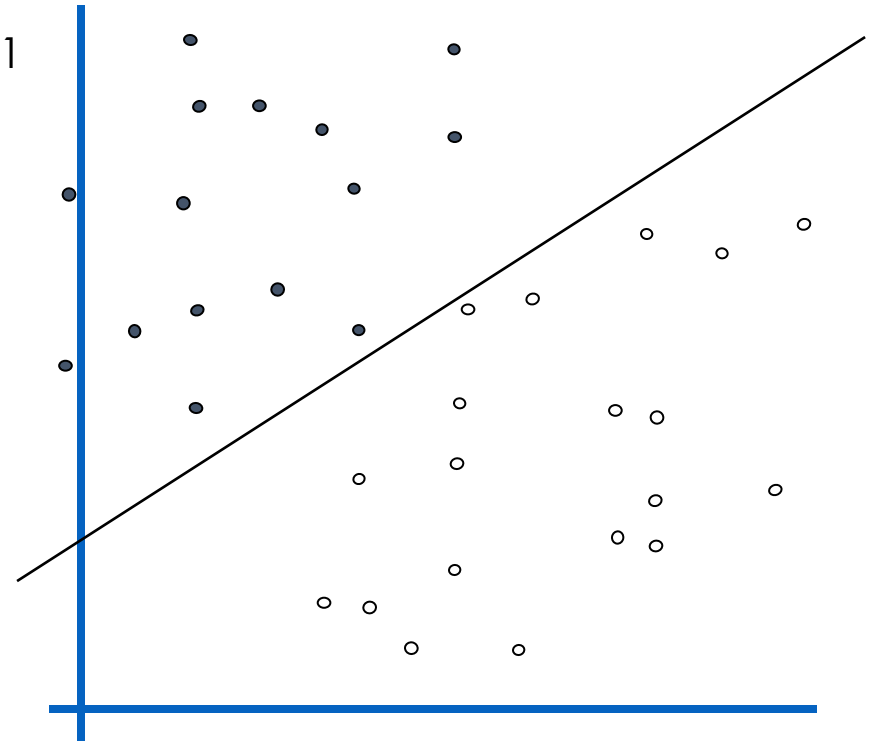


Linear Classifiers

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

denotes +1

denotes -1

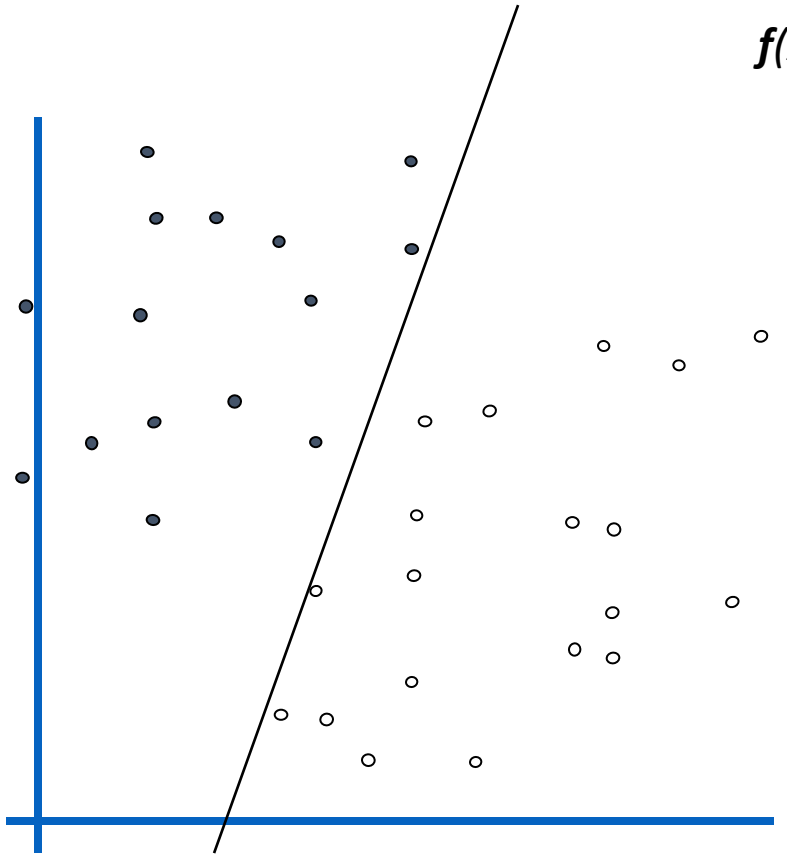


How would you classify this data?

Linear Classifiers

denotes +1

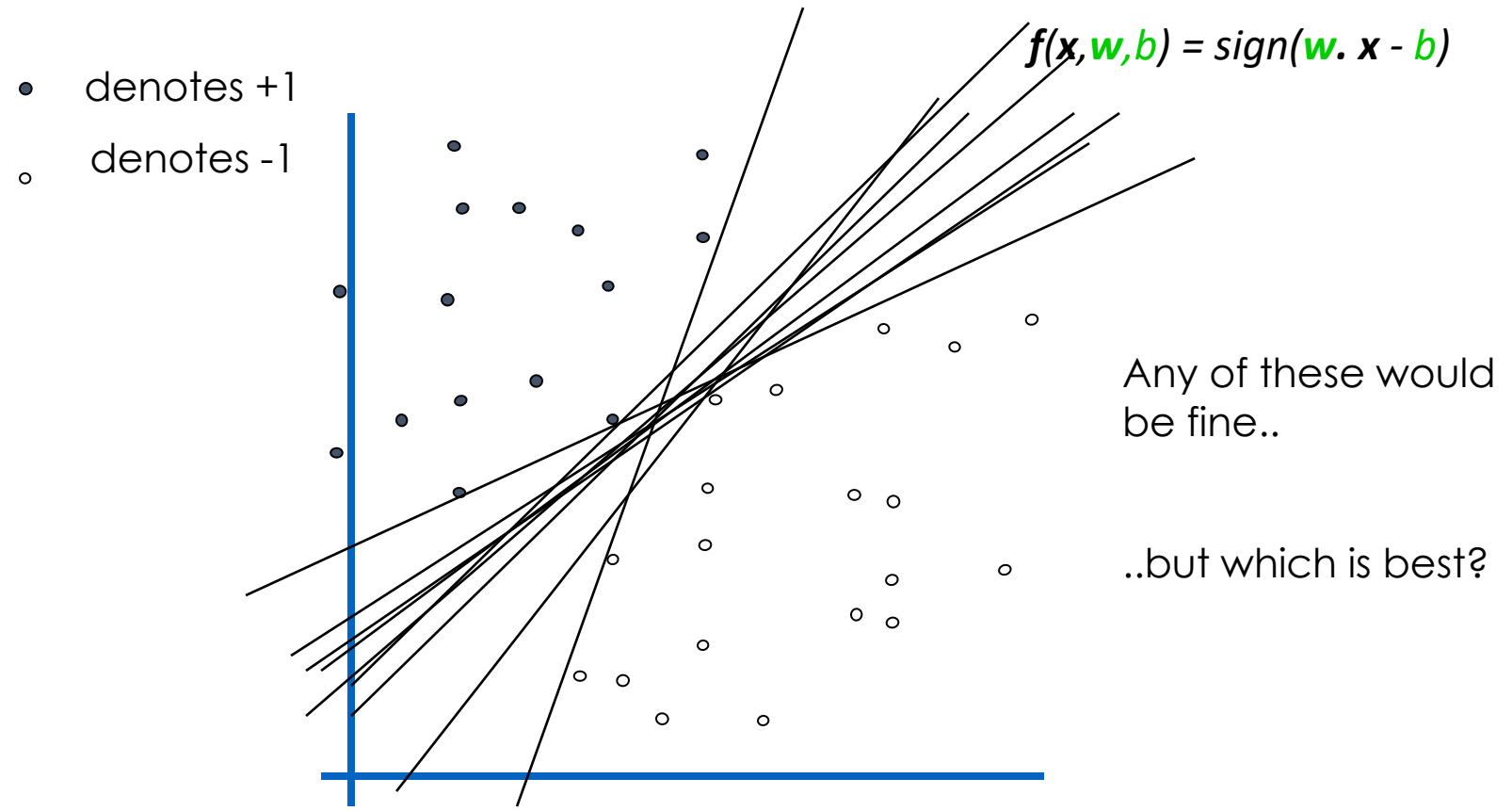
denotes -1



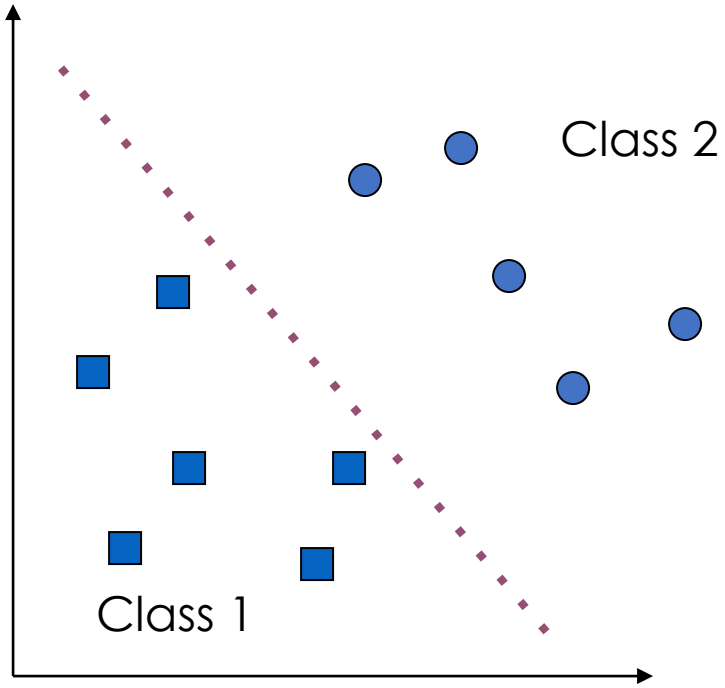
$f(x,w,b) = sign(w.\ x - b)$

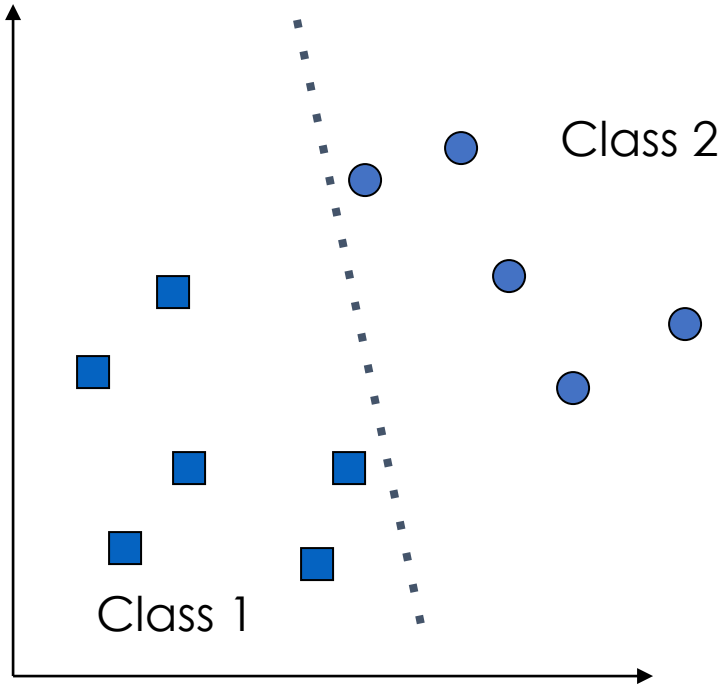
How would you classify this data?

Linear Classifiers



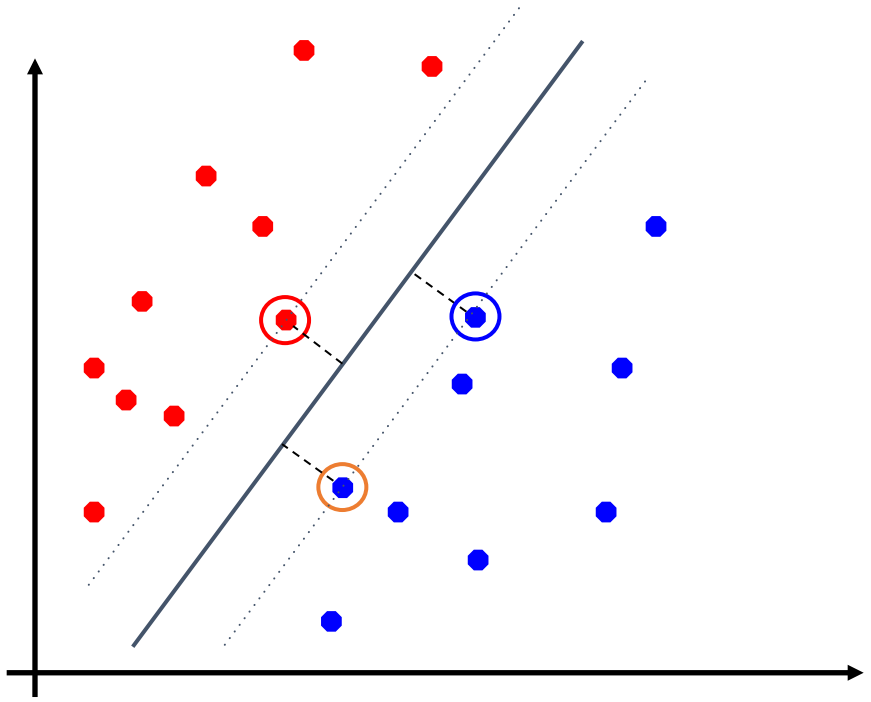
Example of Bad Decision Boundaries





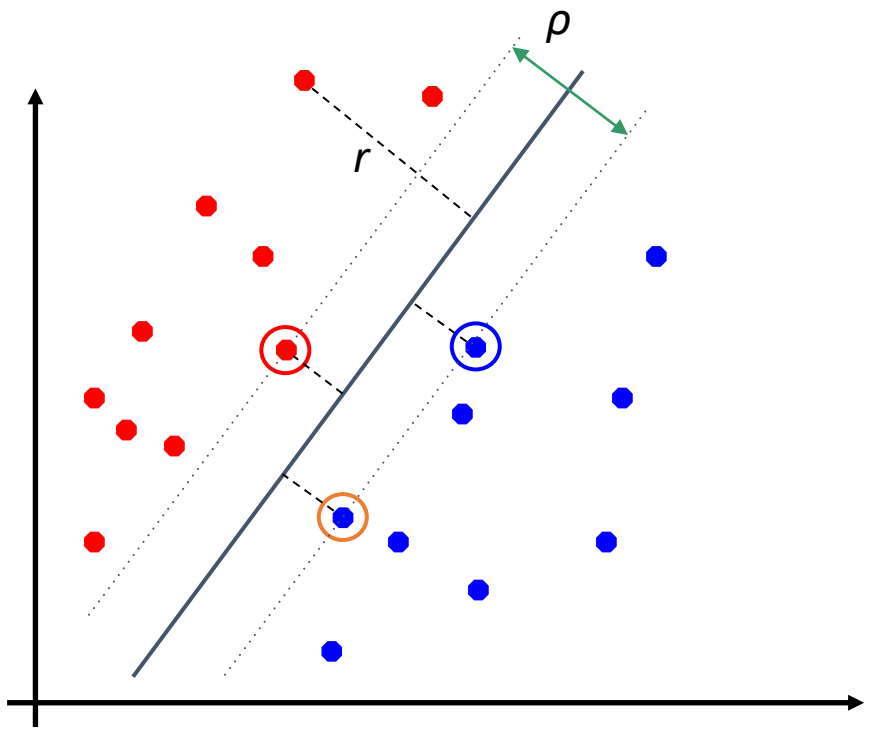
Maximum Margin Classification

- Maximizing the margin is good according to intuition and theory.
- Implies that only support vectors matter; other training examples are ignorable.



Classification Margin

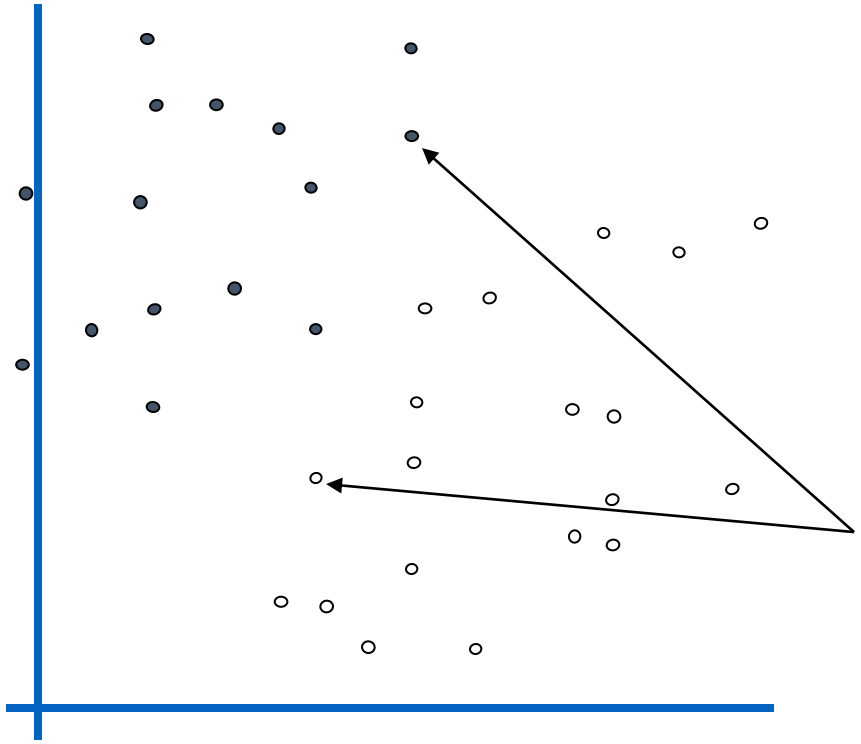
- Distance from example $\mathbf{x}_i$ to the separator is $r = \dfrac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



Classifier Margin

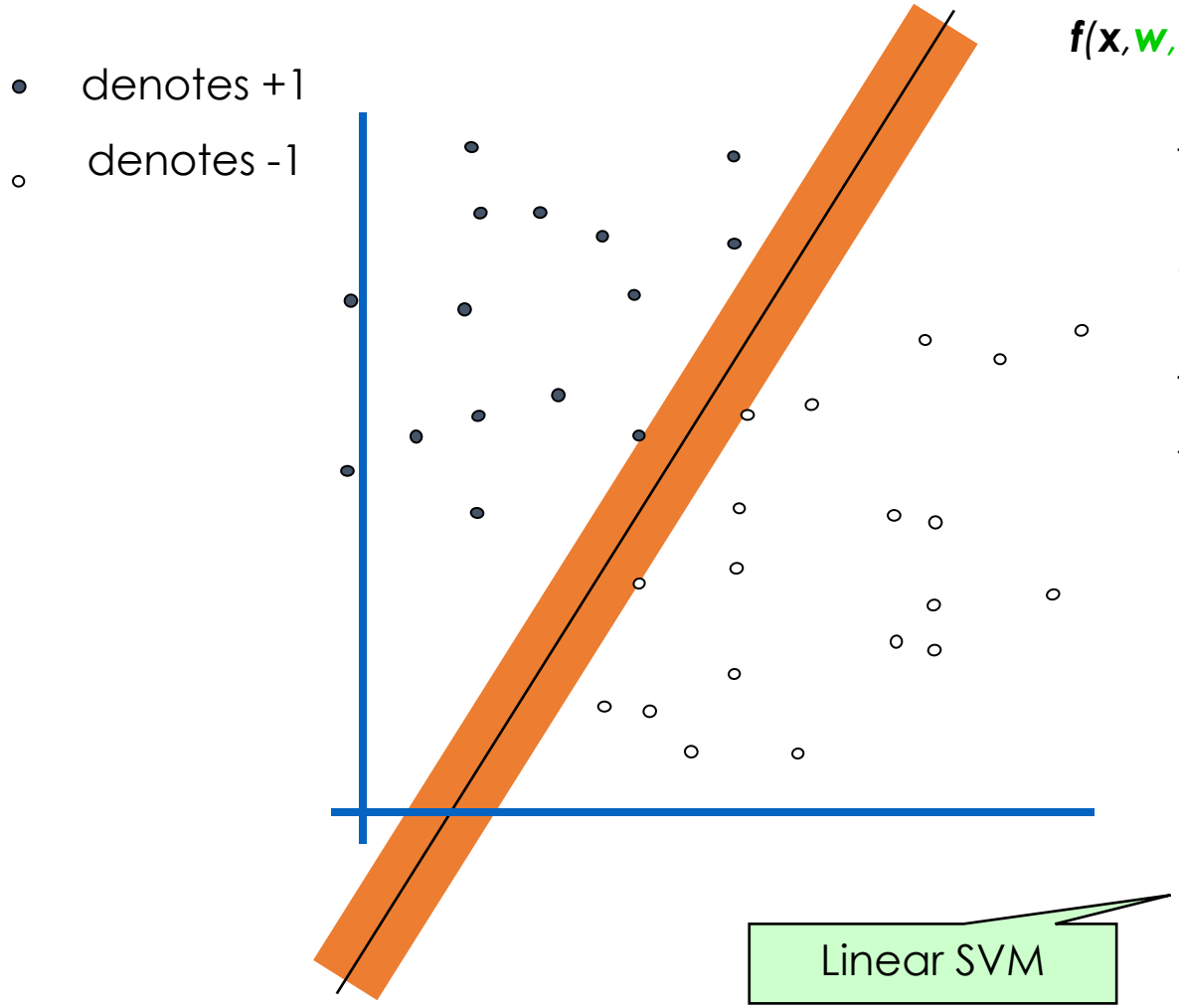
denotes +1

denotes -1



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin

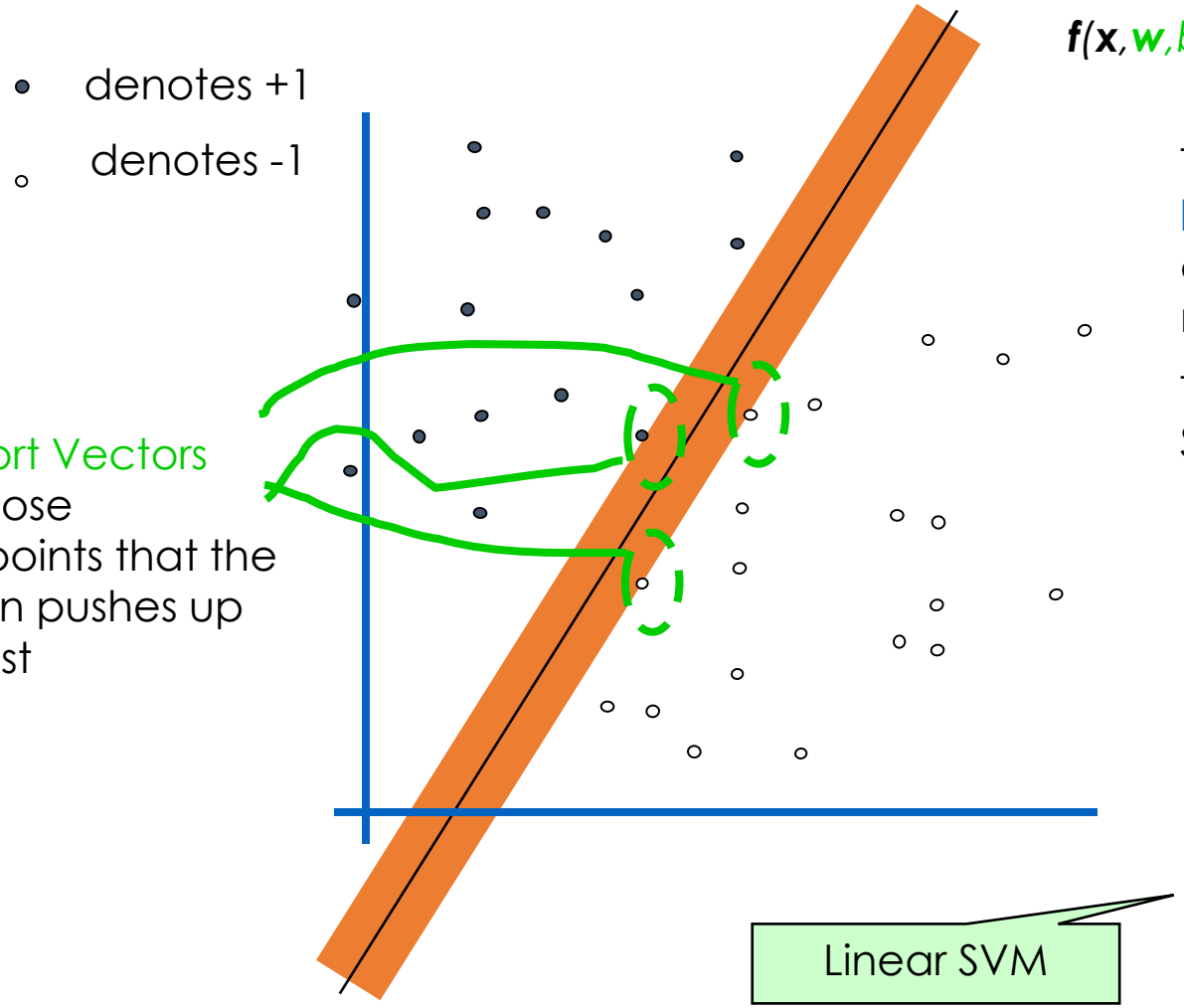


$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

The maximum margin linear classifier is the linear classifier with the maximum margin.

This is the simplest kind of SVM (called an LSVM)

Maximum Margin



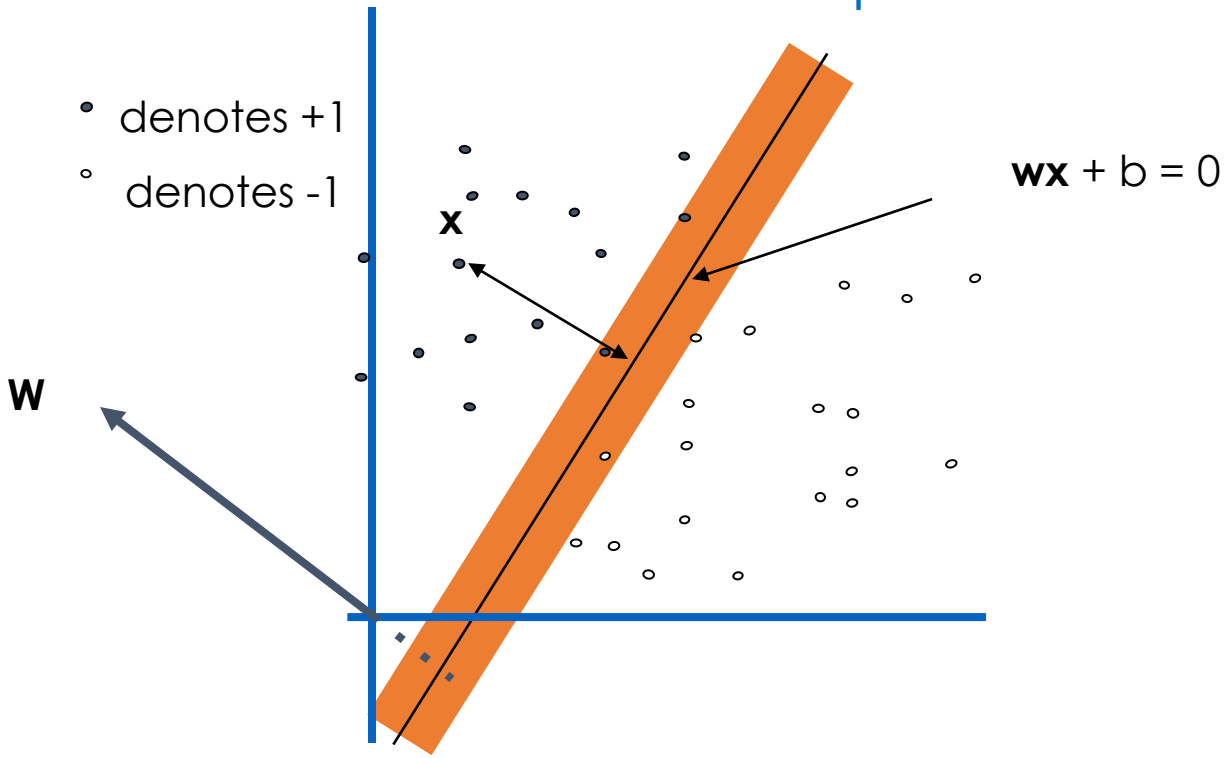
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} + b)$

The maximum margin linear classifier is the linear classifier with the maximum margin.

This is the simplest kind of SVM (called an LSVM)

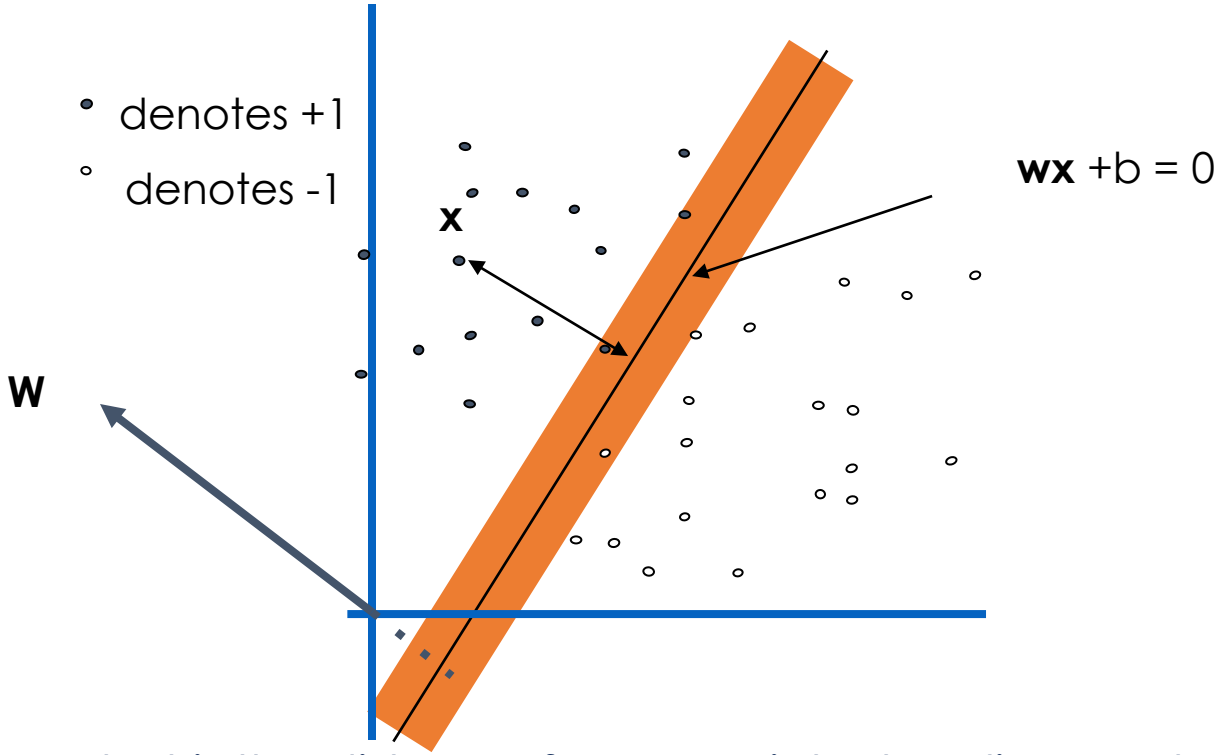
Support Vectors are those datapoints that the margin pushes up against

How to calculate the distance from a point to a line?



- In our case, $w_1*x_1+w_2*x_2+b=0$,

Estimate the Margin

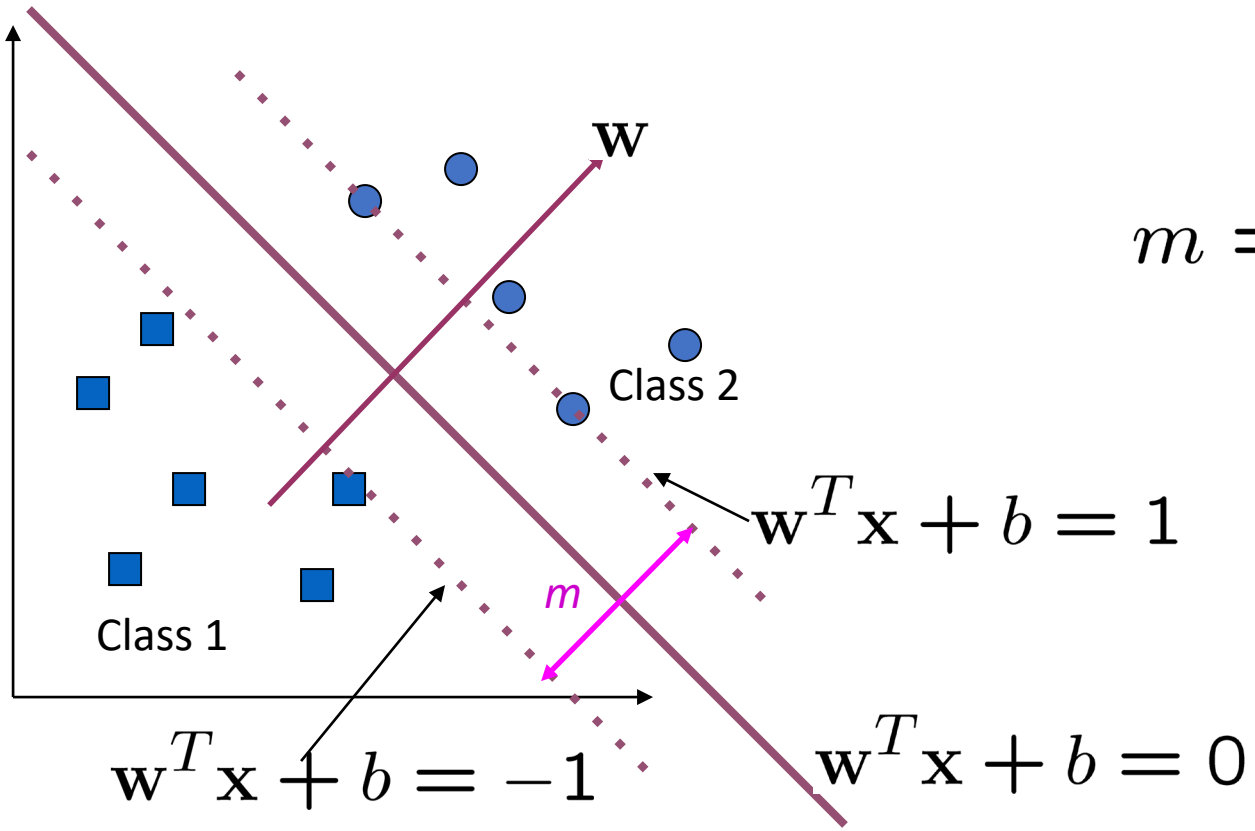


- What is the distance from a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*
  - Distance between the origin and the line $\mathbf{w}^t\mathbf{x}$=-b is b/| |$\mathbf{w}$| |



$$m = \frac{2}{||\mathbf{w}||}$$

Finding the Decision Boundary

- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly ⇒ $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$
- To see this: when y=-1, we wish (wx+b)<1, when y=1, we wish (wx+b)>1. For support vectors, we wish y(wx+b)=1.
- The decision boundary can be found by solving the following constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

Next steps

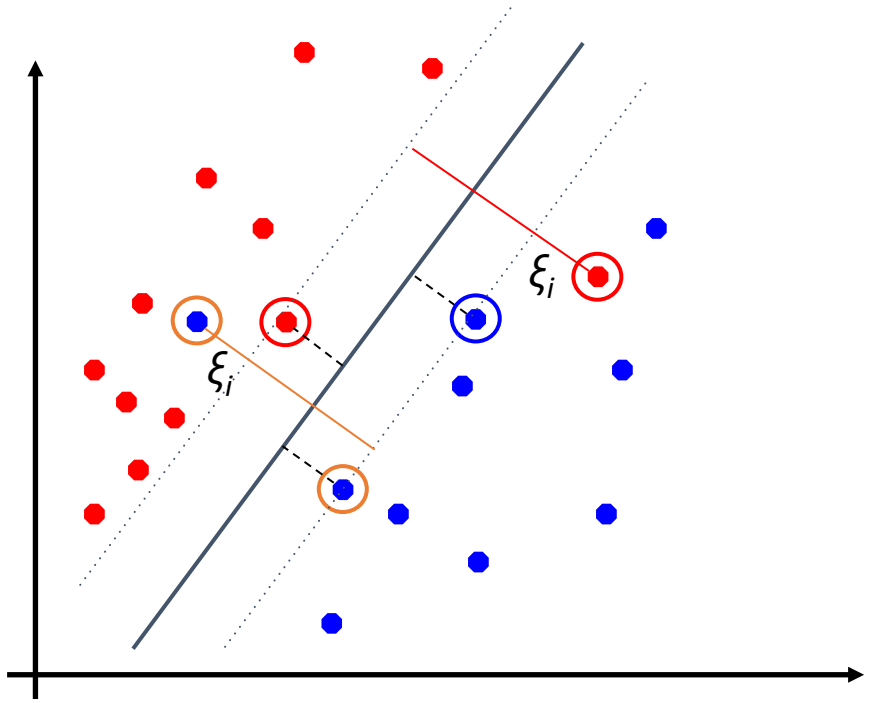
- Converting SVM to a form we can solve
  - Dual form
- Allowing a few errors
  - Soft margin
- Allowing nonlinear boundary
  - Kernel functions

Linear SVMs: Overview

- The classifier is a ***separating hyperplane***.
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers a_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products.

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples



Soft Margin Hyperplane

- Introduce slack variables ξ_i=0

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$

- C : tradeoff parameter between error and margin
- The optimization problem becomes

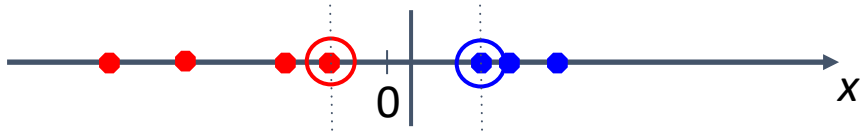
$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Extension to a non-linear Decision Boundary

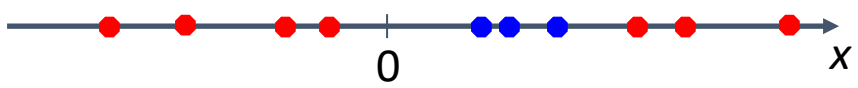
- So far, we have only considered large-margin classifiers with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space
  - Input space: the space where the points $\mathbf{x}_i$ are located
  - Feature space: the space of $\varphi(\mathbf{x}_i)$ after transformation

Non-linear SVMs

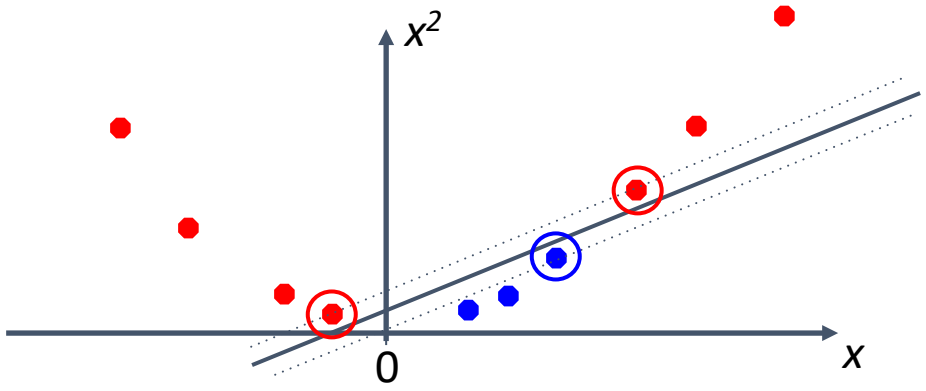
- Datasets that are linearly separable with some noise work out great:



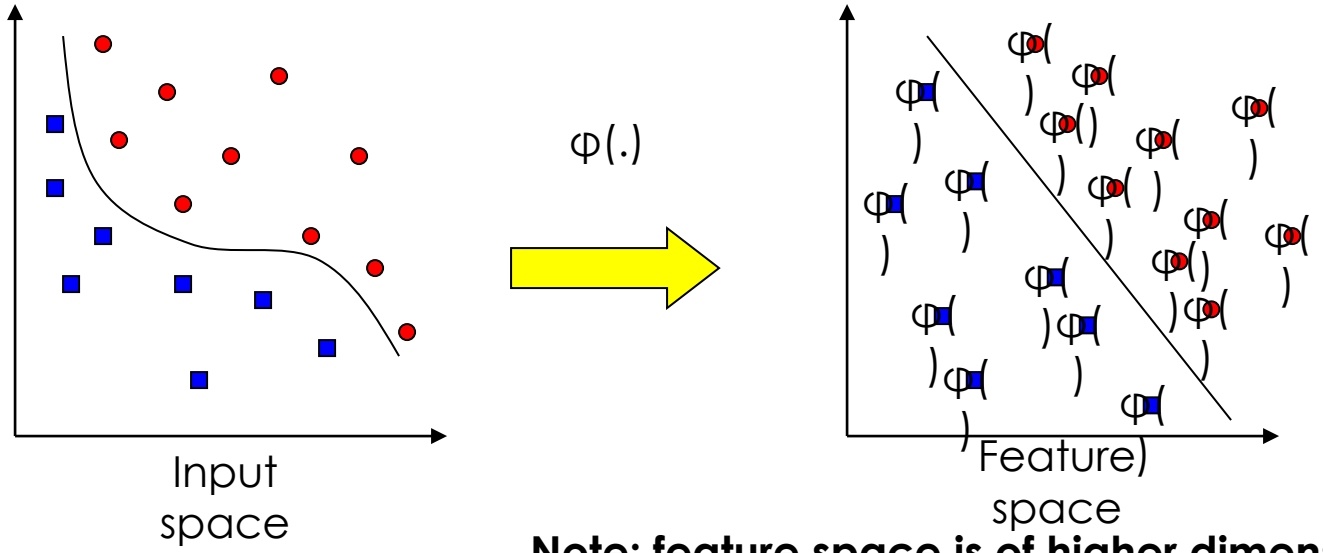
- But what are we going to do if the dataset is just too hard?



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



Transforming the Data

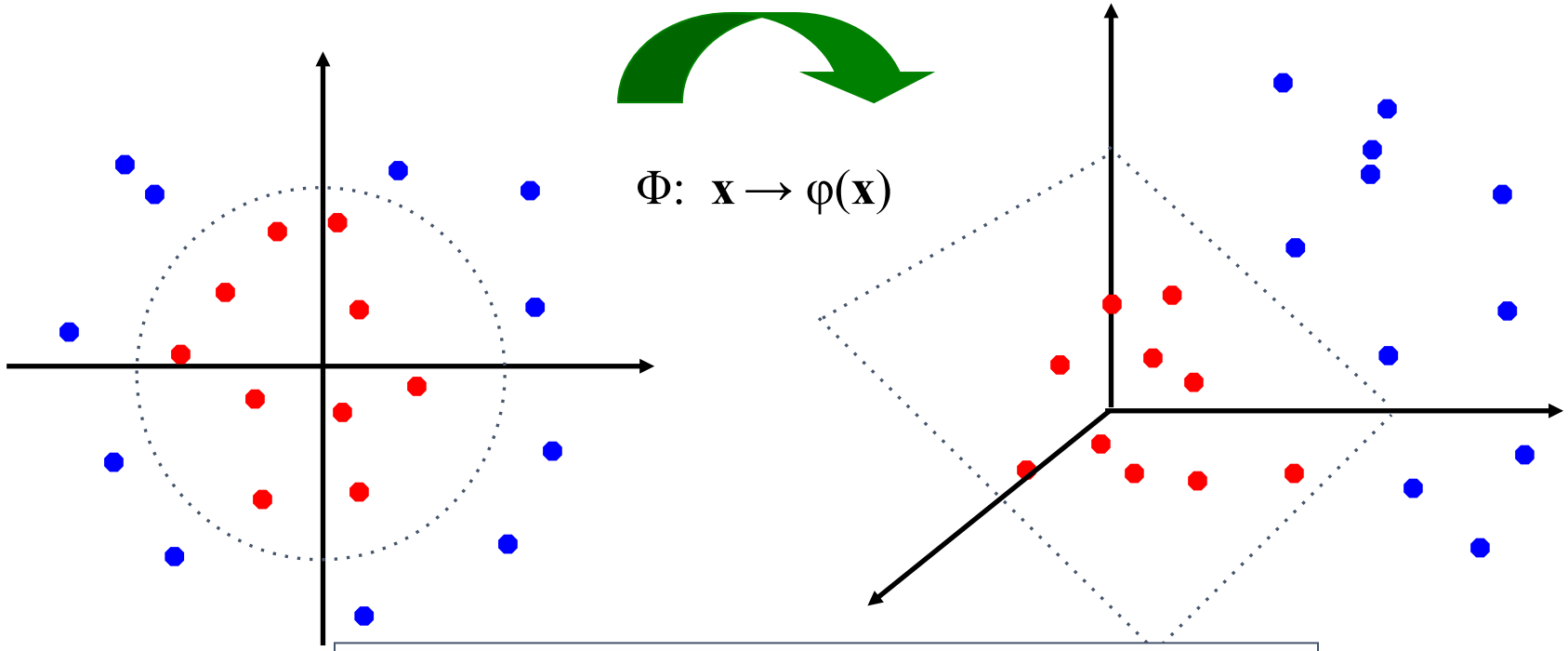


Note: feature space is of higher dimension than the input space in practice

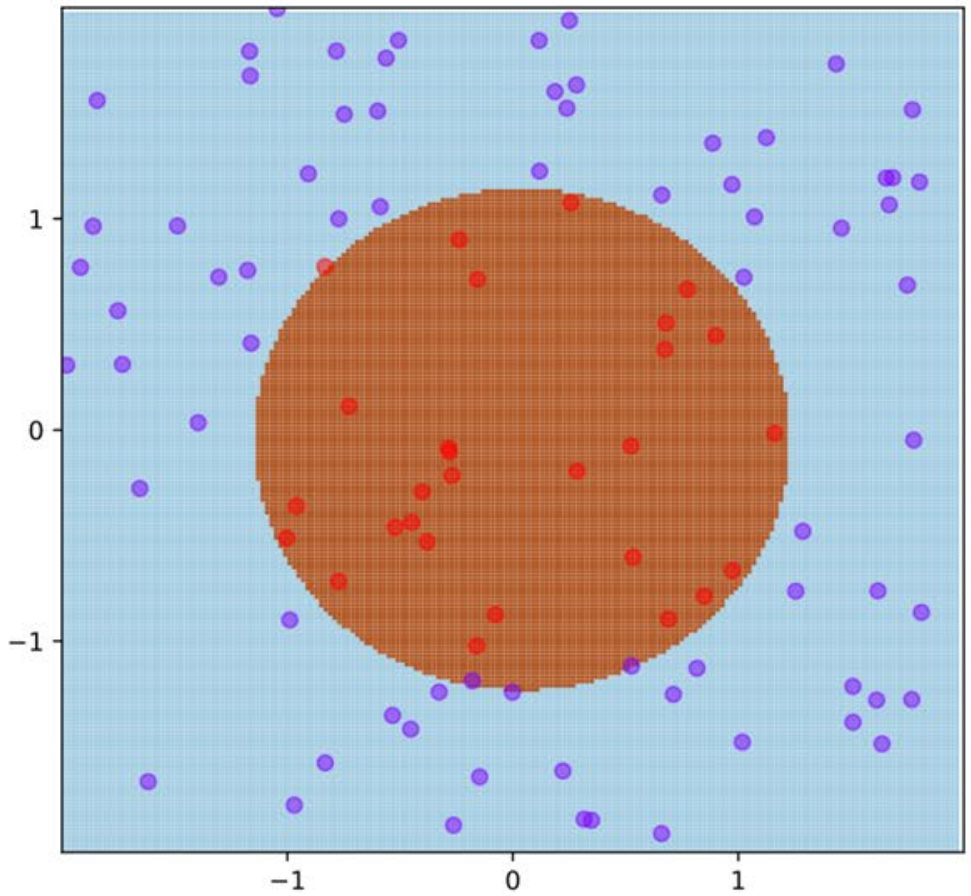
- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

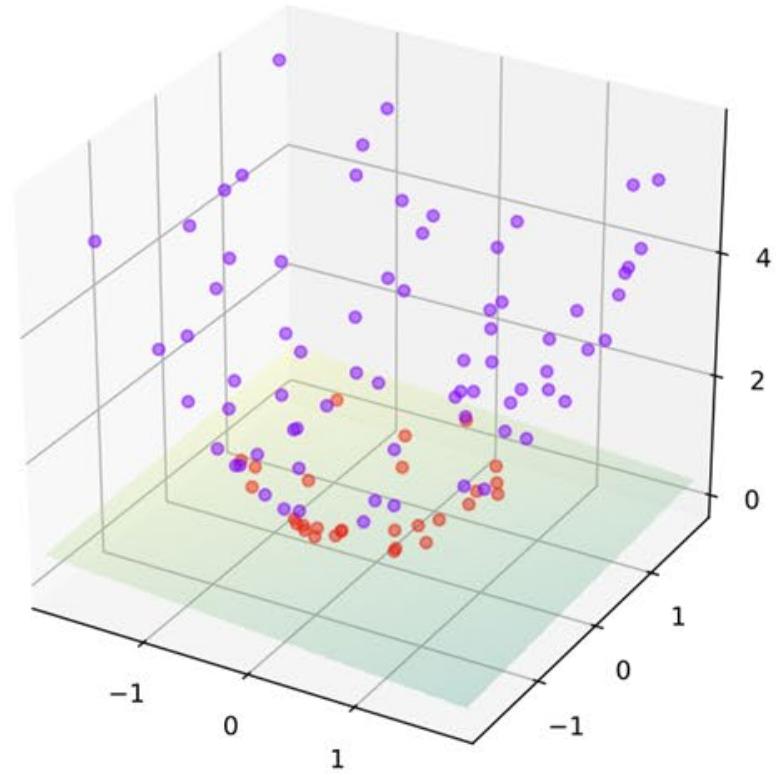
Non-linear SVMs: Feature spaces

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

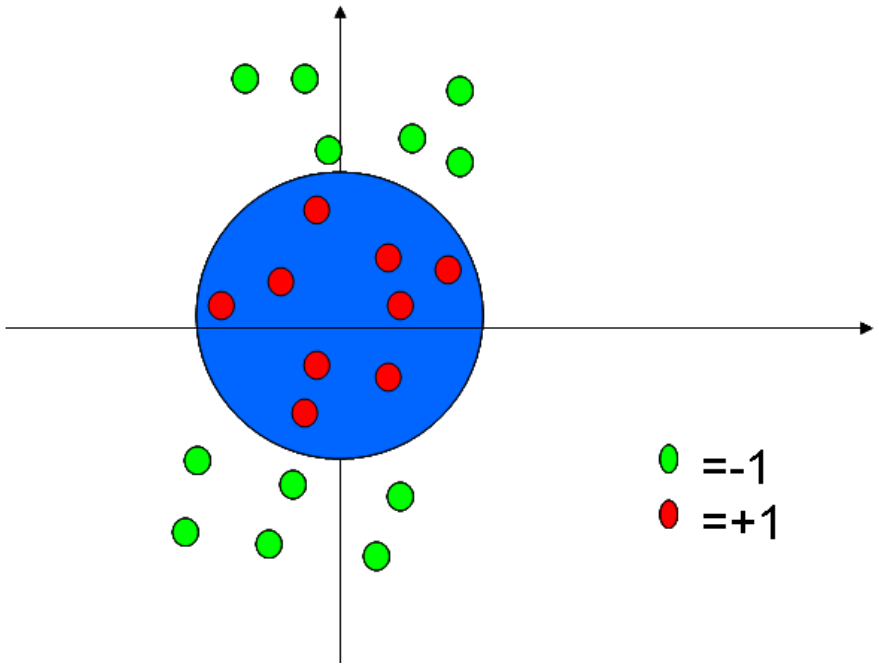


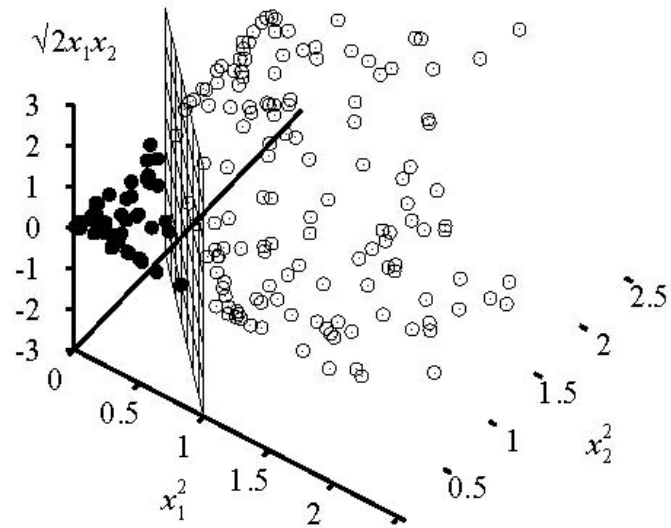
$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$


1
0
−1
−1
0
1


4
2
0
−1
0
1
1
0
−1

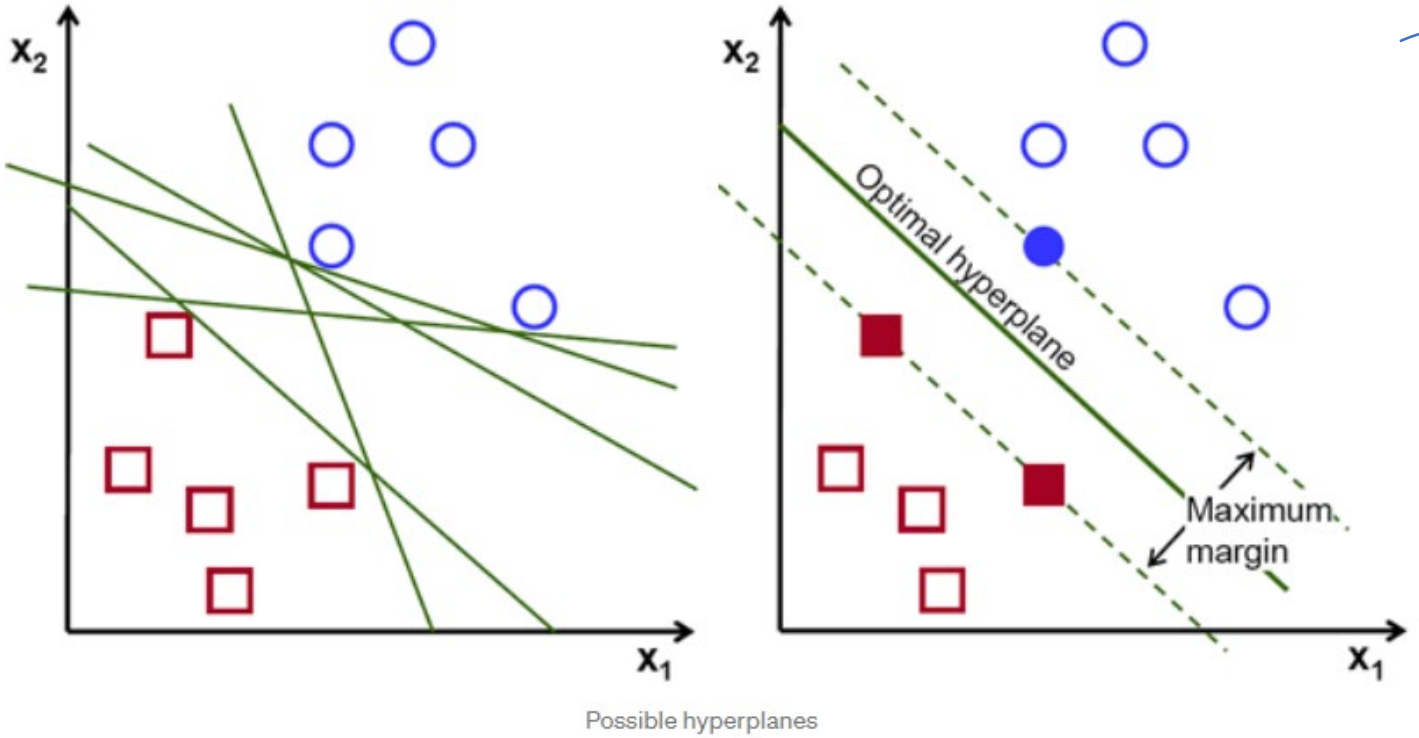
Kernel Trick





Data points are linearly separable in the space $(x_1^2, x_2^2, \sqrt{2}x_1x_2)$

Support Vector Machine



How: we find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

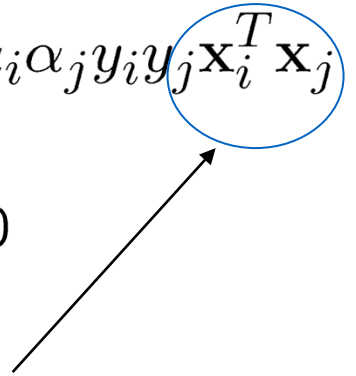
Image Source: Towards Data Science

The Kernel Trick

- Recall the SVM optimization problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$



- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function *K* by $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

An Example for φ(.) and K(.,.)

- Suppose φ(.) is given as follows

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out φ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out φ(.) explicitly is known as the kernel trick

Kernel Functions

- Not all similarity measures can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer function, i.e., the function is "positive-definite"
- This implies that
    - the *n* by *n* kernel matrix,
    - in which the (i,j)-th entry is the $K(\mathbf{x}_i, \mathbf{x}_j)$, is always positive definite
- This also means that optimization problem can be solved in polynomial time!

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

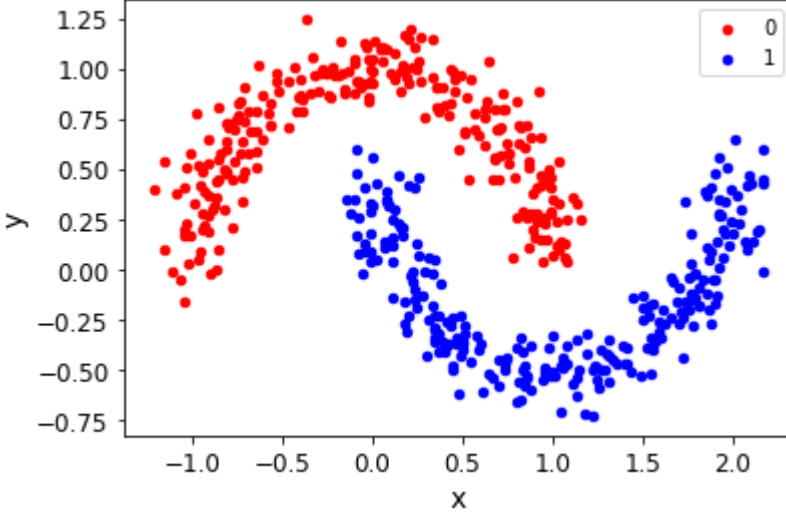
  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional

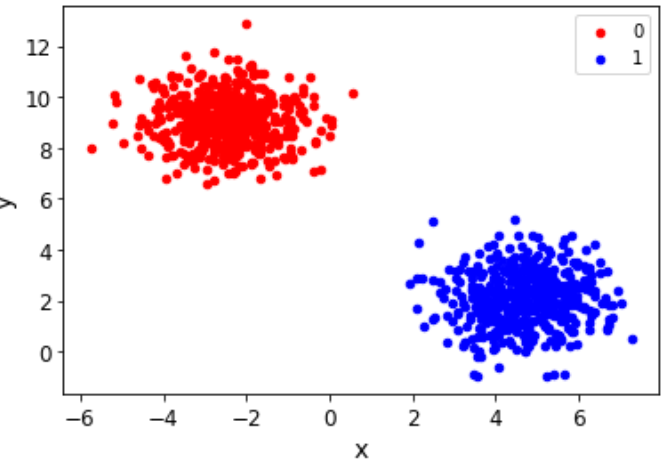
- Sigmoid with parameter κ and θ

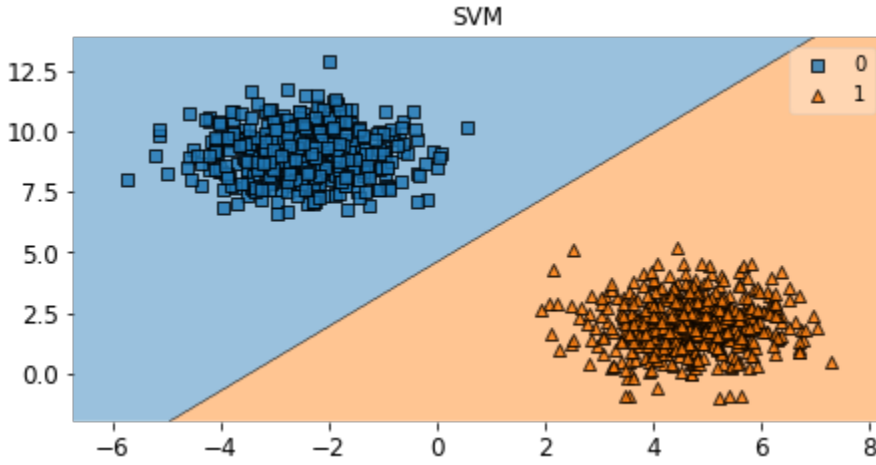
$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

SVMs in Python

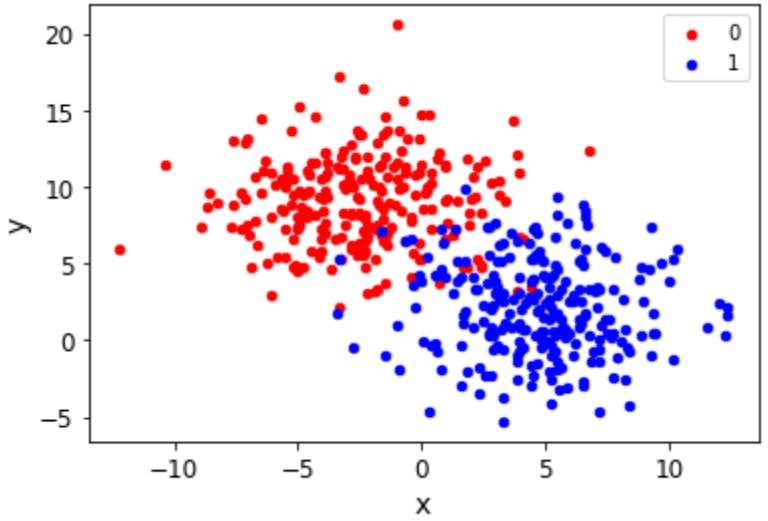
- **Generate data**
  - X, y = make_moons(n_samples=500, noise=0.1)
- **Split into test and training data**
  - X_train, X_test, y_train, y_test = train_test_split(X, y)
- **Support vector machine**
  - svm_clf = SVC(kernel="linear")
  - svm_clf.fit(X_train, y_train)
  - plot_decision_regions(X, y, svm_clf)





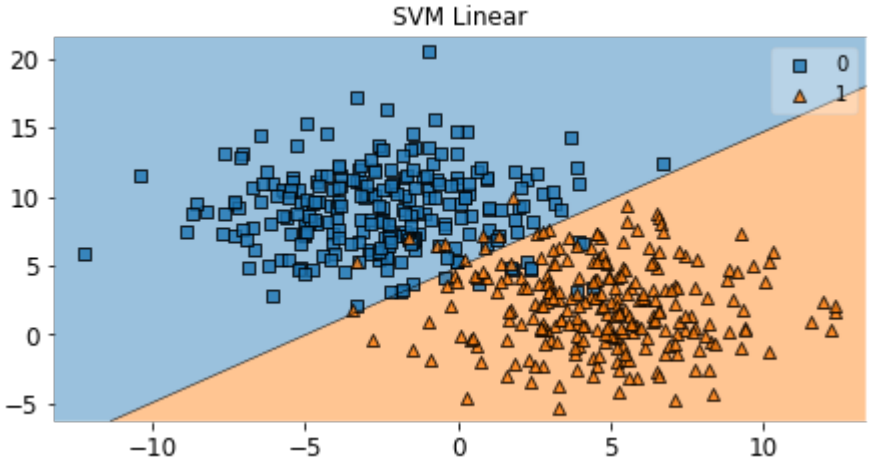


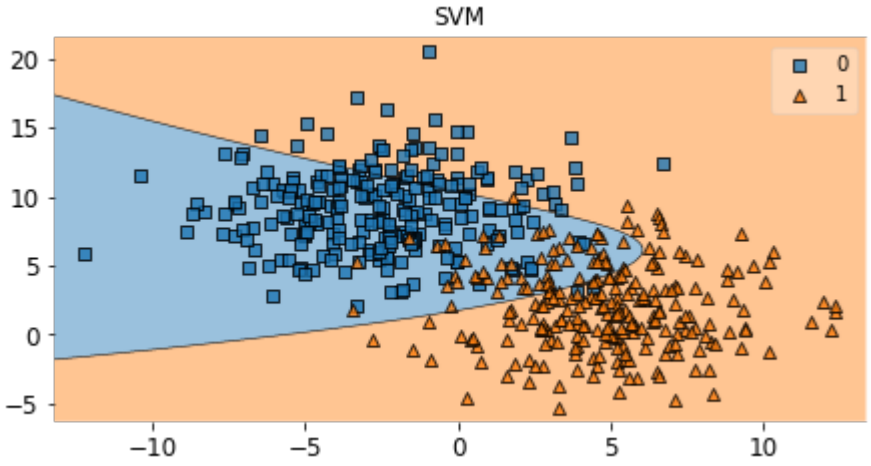
SVMs in Python



Non-linear classification vs linear soft-margin classification

- svm_linear_clf = SVC(kernel= "linear", C=100)
- svm_clf = SVC(kernel= "sigmoid", C=100, tol=0.00001)





Exercise – In-sample vs Out-of-sample

Colab SVM Implementation:
https://colab.research.google.com/drive/1BOebDvNWX4rOM4BbEqvlmQ4Sy_5bCQ0k?usp=sharing

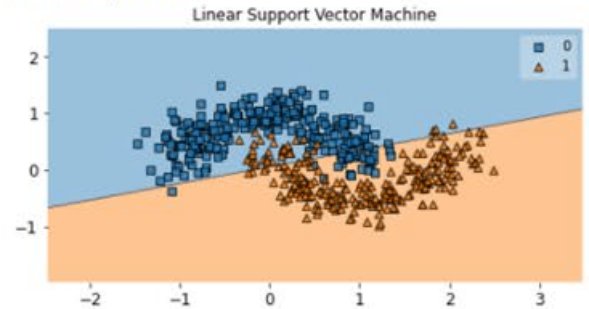
▾ Linear Support Vector Machine

```
[20] # define SVC classifier
     svm_clf = SVC(random_state=42, kernel="linear")
```

Python code for a SVM with linear kernel.

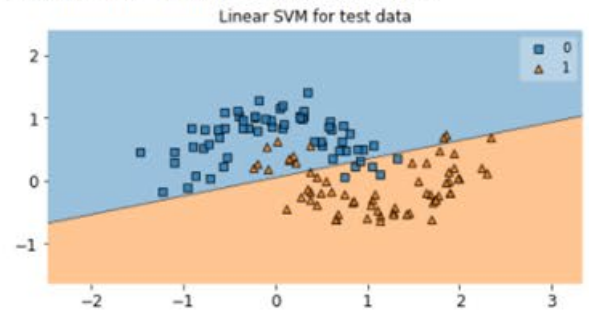
```
#### Plot the decision boundary for train data
plt.figure(figsize=(16,12))
plt.subplot(321)
plot_decision_regions(X, y,svm_clf)
plt.title("Linear Support Vector Machine")

y_pred = svm_clf.predict(X_train)
print("in sample", accuracy_score(y_train, y_pred))
y_pred = svm_clf.predict(X_test)
print("out-of-sample", accuracy_score(y_test, y_pred))
```

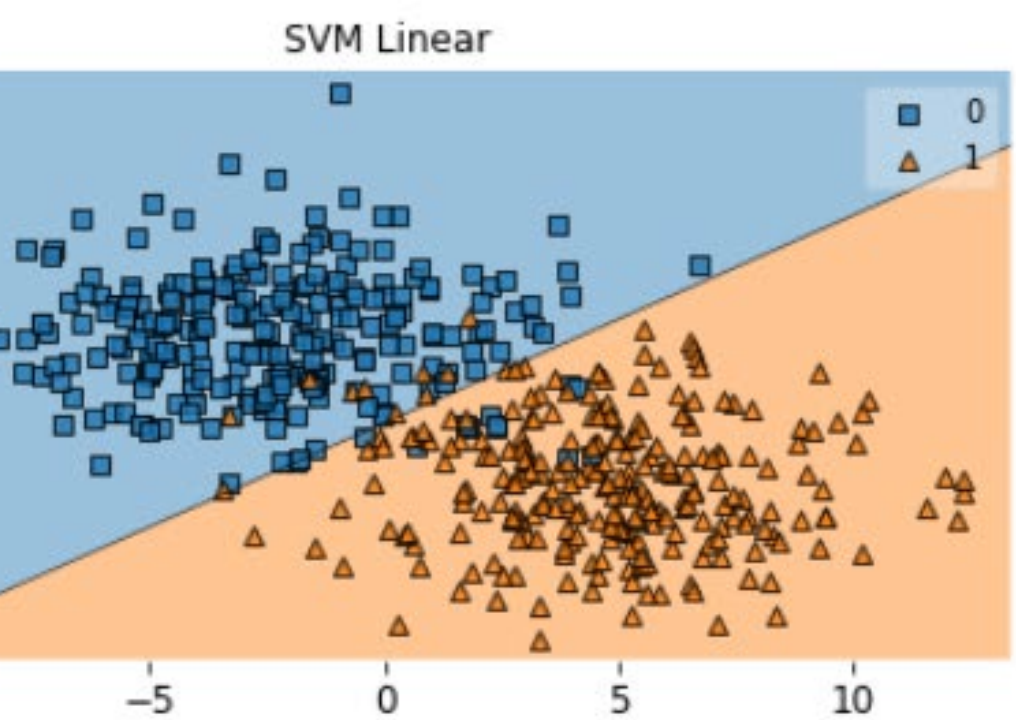
```
in sample 0.8586666666666667
out-of-sample 0.856
```

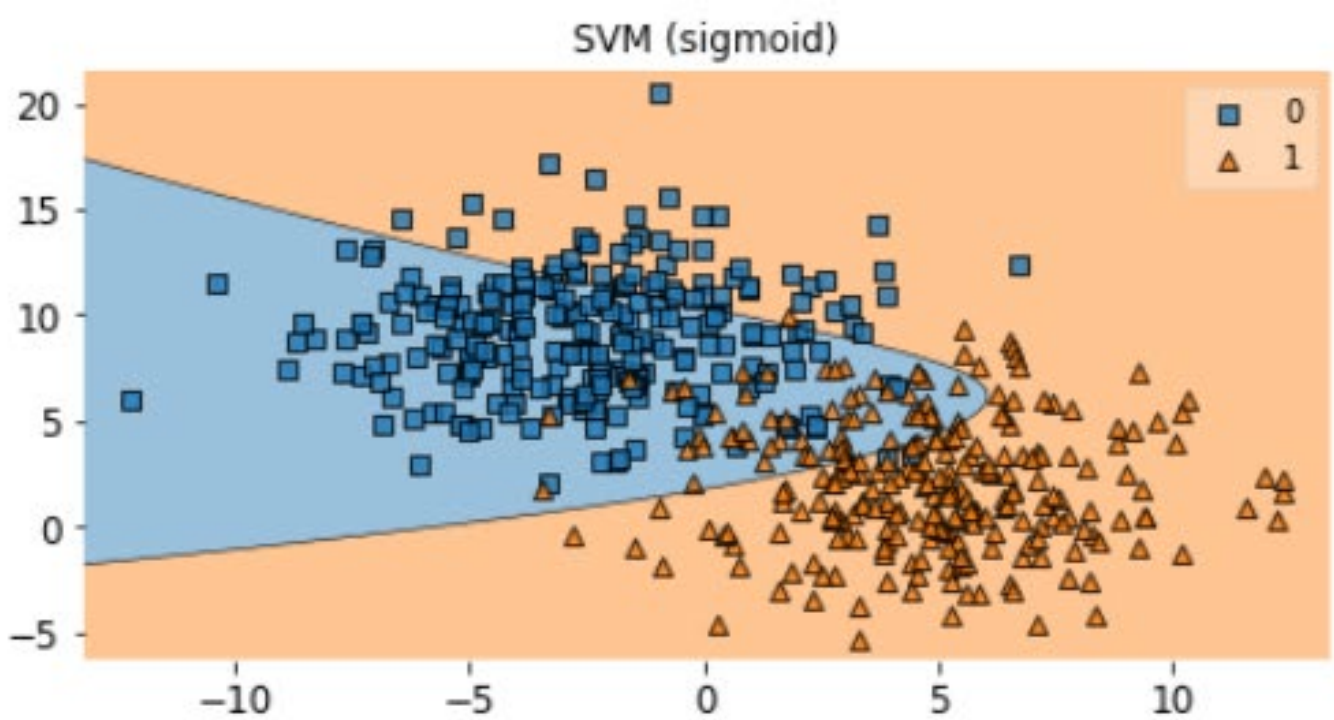


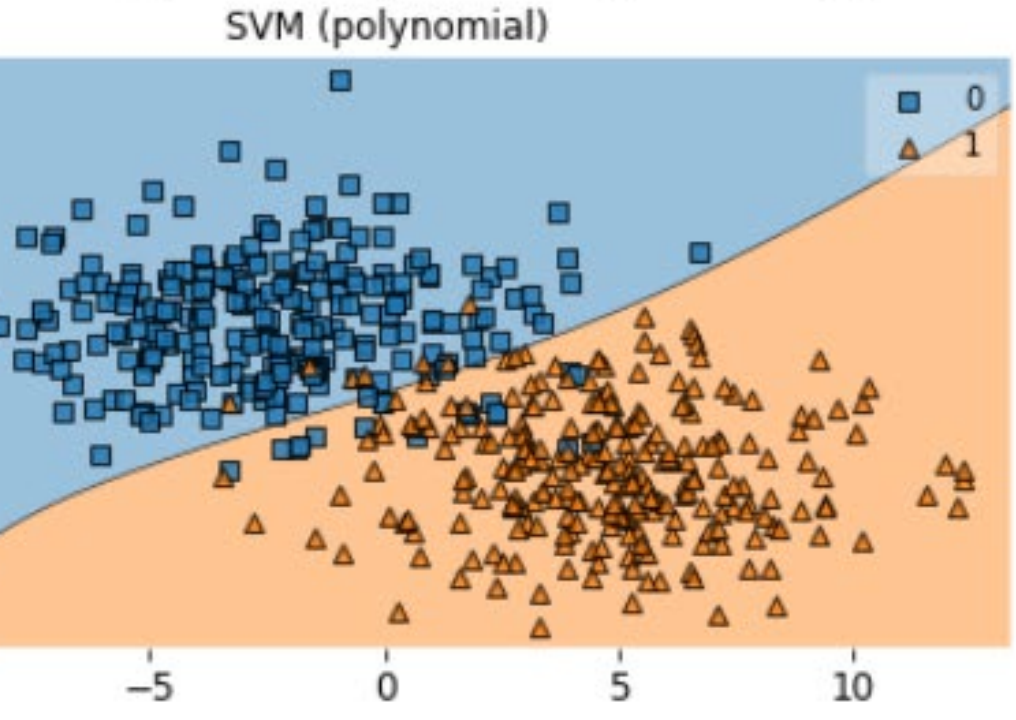
[22]
```
#plot decision boundary for test data
plt.figure(figsize=(16,12))
plt.subplot(321)
plot_decision_regions(X_test, y_test, svm_clf)
plt.title("Linear SVM for test data")
```

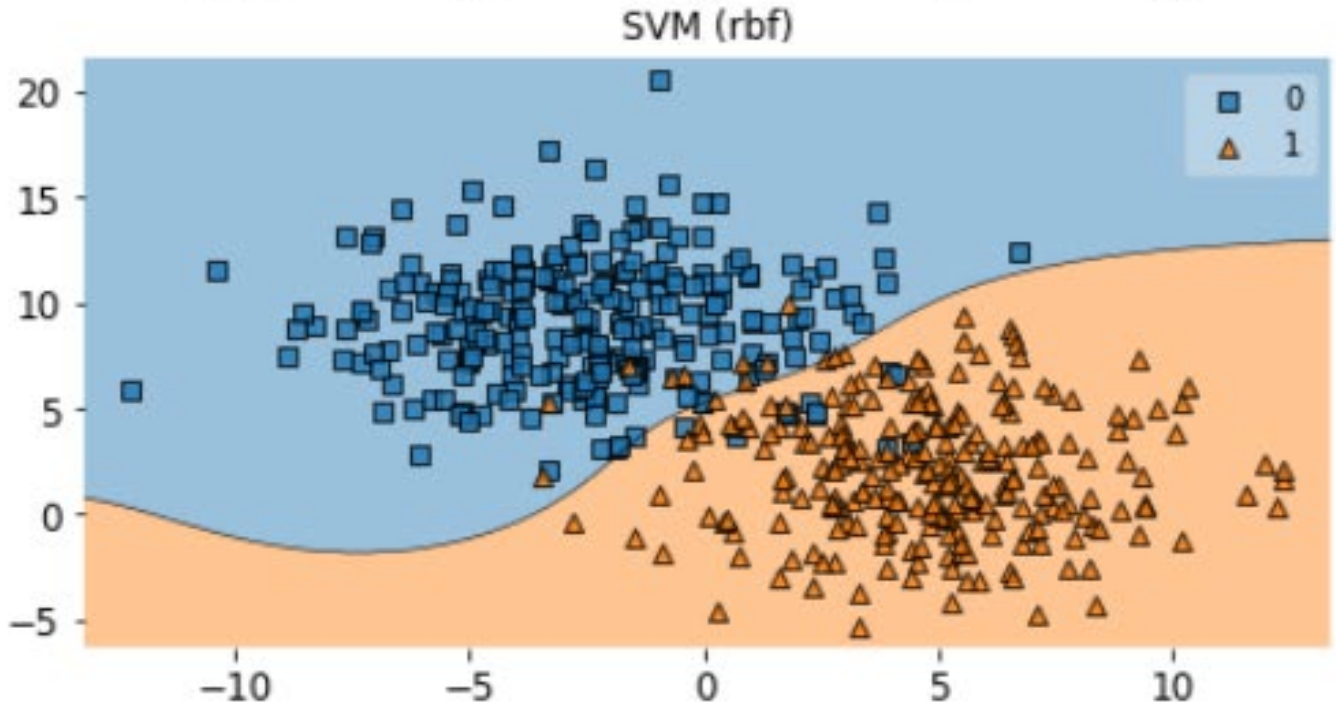
```
Text(0.5, 1.0, 'Linear SVM for test data')
```




SVM Linear
0
1
−5
0
5
10


SVM (sigmoid)
0
1
20
15
10
5
0
−5
−10
−5
0
5
10


SVM (polynomial)
0
1
−5
0
5
10


SVM (rbf)
0
1
20
15
10
5
0
−5
−10
−5
0
5
10

SVMs in Python

Accuracy

- y_pred = svm_clf.predict(X_train)
- print(svm_clf.__class__.__name__, accuracy_score(y_train, y_pred)) # in sample
- y_pred = svm_clf.predict(X_test)
- print(svm_clf.__class__.__name__, accuracy_score(y_test, y_pred)) # out of sample

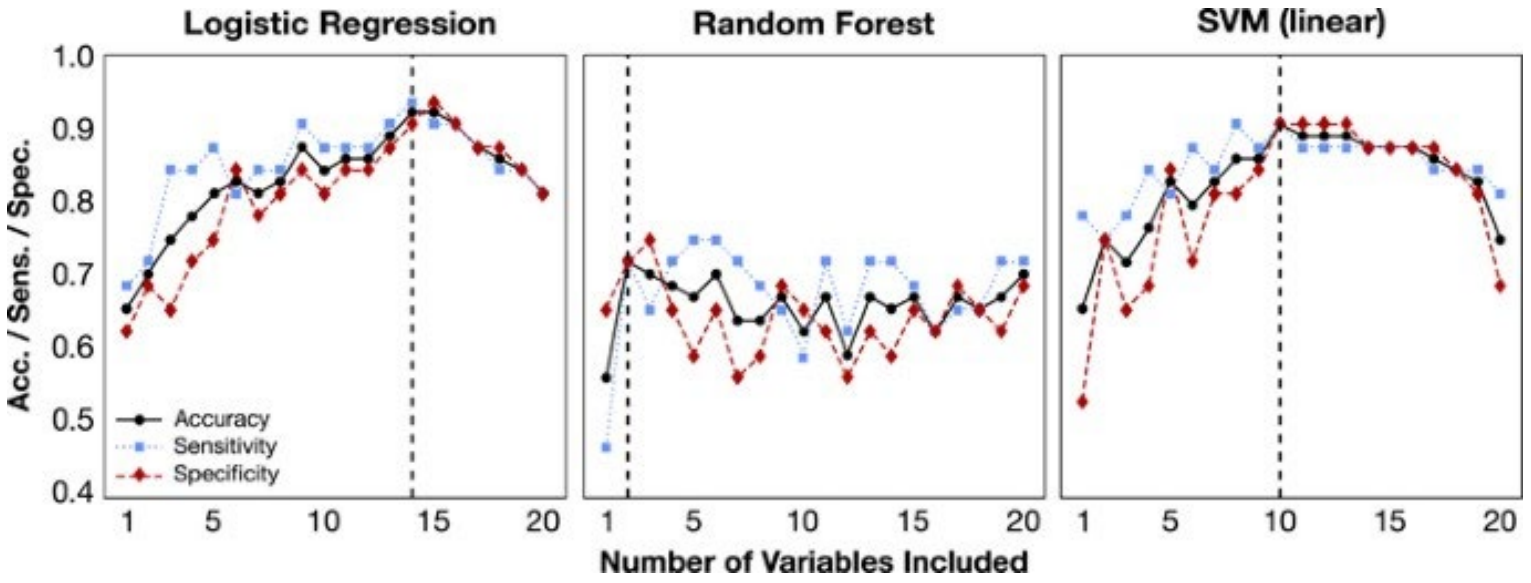
```
SVC 0.952
SVC 0.968
```

How should they look like?

In sample accuracy should normally be larger than out of sample accuracy.

How to increase model accuracy

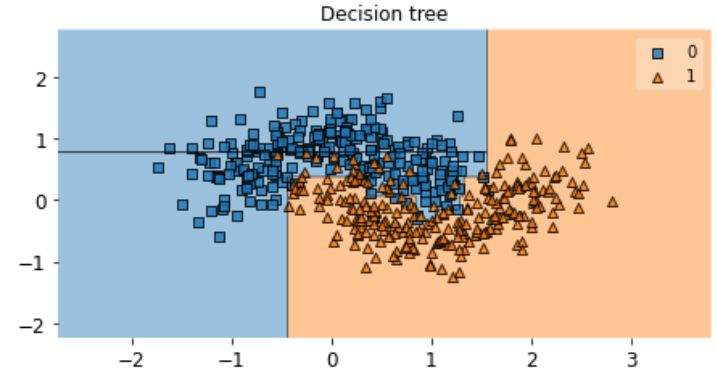
1. Add more data. Having more data is always a good idea. (depends on model, see graphic)
2. Treat missing and Outlier values. ...
3. Feature Engineering. ...
4. Feature Selection. ...
5. Multiple algorithms. ...
6. Algorithm Tuning. ...
7. Ensemble methods.

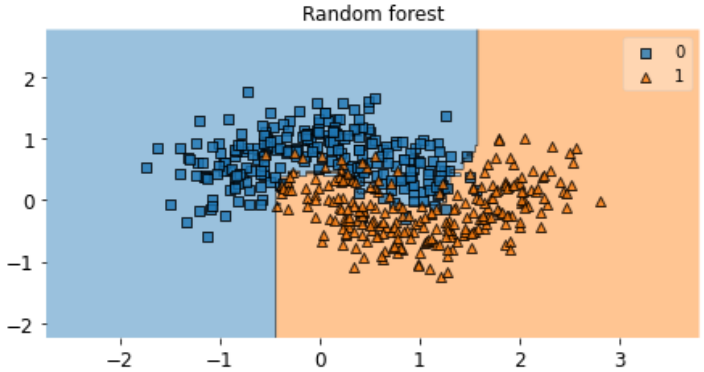


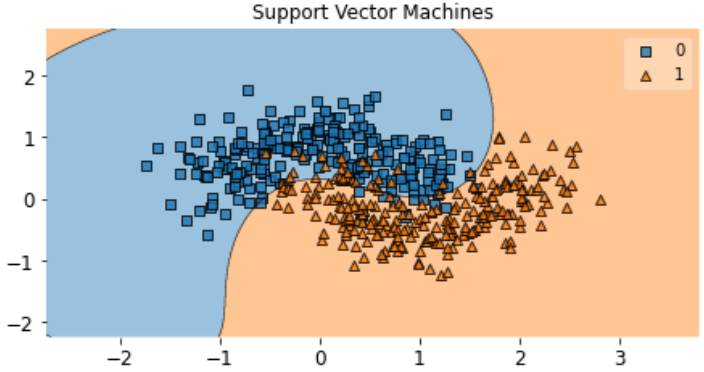
Conclusion

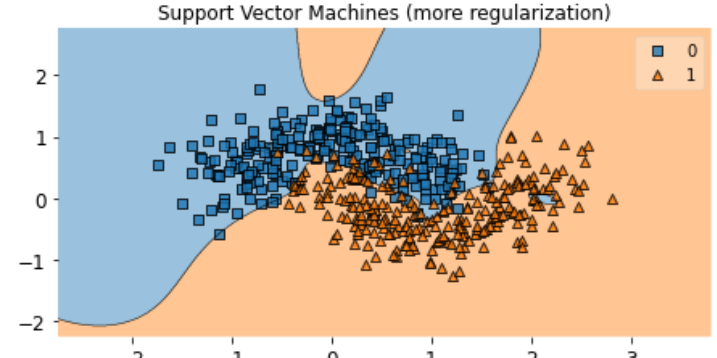
- SVM is a useful alternative to neural networks
- Two key concepts of SVM: maximize the margin and use the kernel trick for non-linear classification
- A lot of active research is taking place in areas related to SVMs
- Many SVM implementations are available on the web for you to try on your data set!

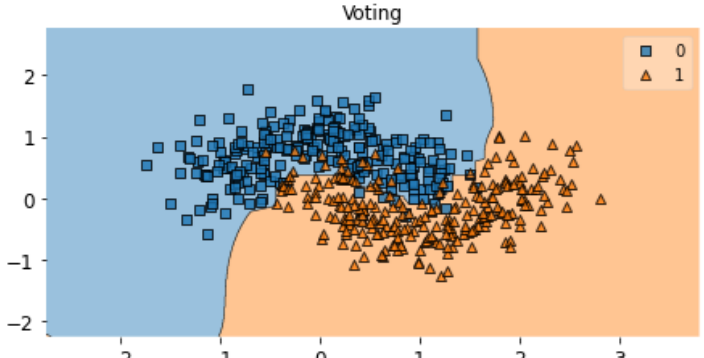
Summary – Support Vector Machines – A machine learning technique











What next?

Dynamic portfolio optimization: Computing an optimal trajectory for a portfolio under realistic assumptions is a NP-Complete problem. See an example here: http://ssrn.com/abstract=2649376

Clustering: Clustering algorithm
rely on heuristics. It would be
replace some of these heurist
brute force search over an
unfathomably large number
combinations. Good clustering methods have applications on risk management and regression analysis. See an example here: http://ssrn.com/abstract=2708678

copulas, are too unrealistic/restrictive.

Option pricing: Some complex derivatives are path-dependent. Evaluating a large number of paths can be computationally expensive.

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

https://colab.research.google.com/drive/1HNipWjKxUZWJfgbxVzE7adjZLkZ3LjuP?usp=sharing#scrollTo=LSMN3am9s65M

Appendix

The Dual Problem (we ignore the derivation)

- The new objective function is in terms of a_i only
- It is known as the dual problem: if we know **w**, we know all a_i; if we know all a_i, we know **w**
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized!

- The dual problem is therefore:

$$\max. \; W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Properties of a_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found

- **w** can be recovered by

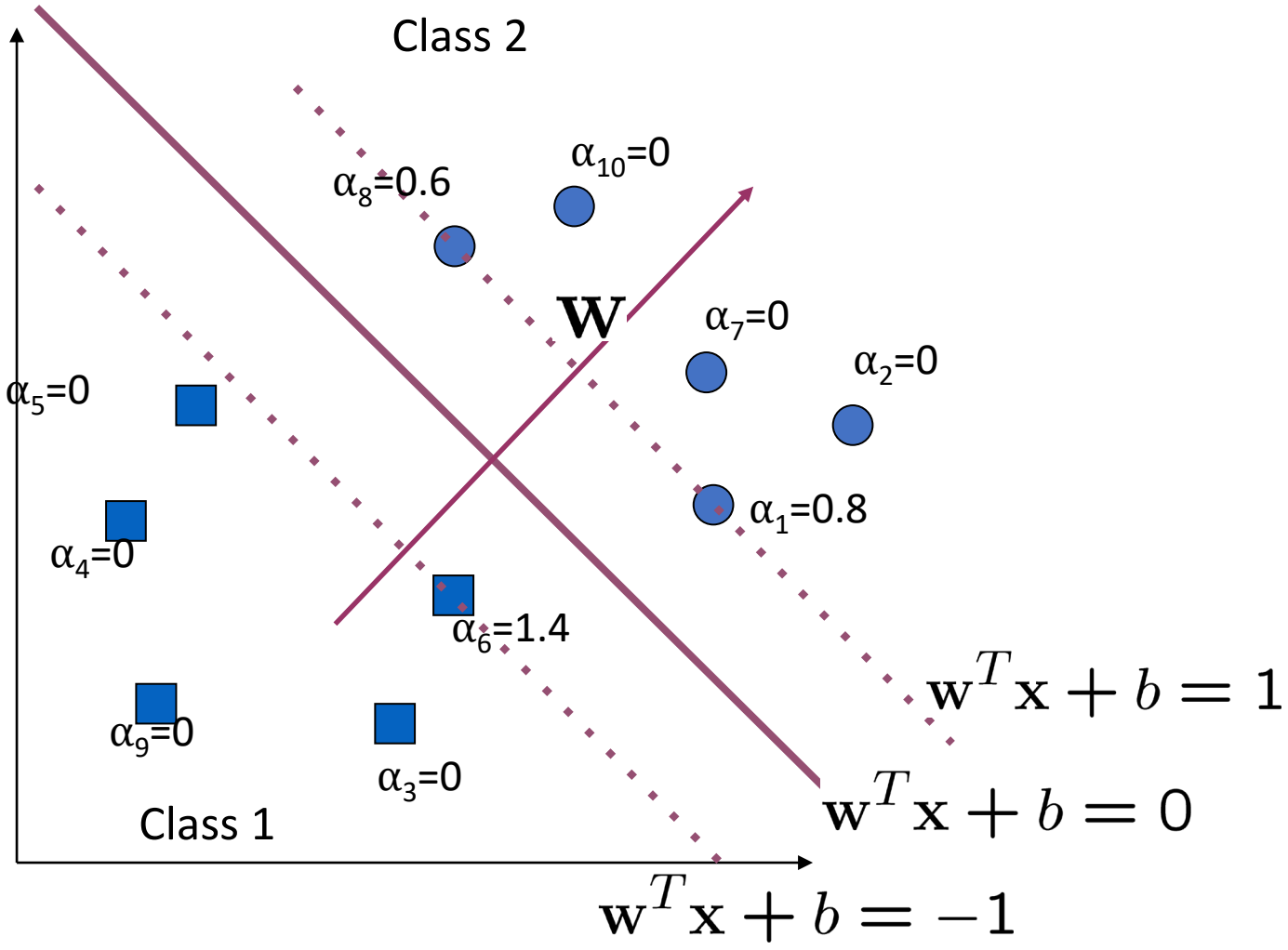
$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Characteristics of the Solution

- Many of the α_i are zero (see next page for example)
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise
  - Note: **w** need not be formed explicitly

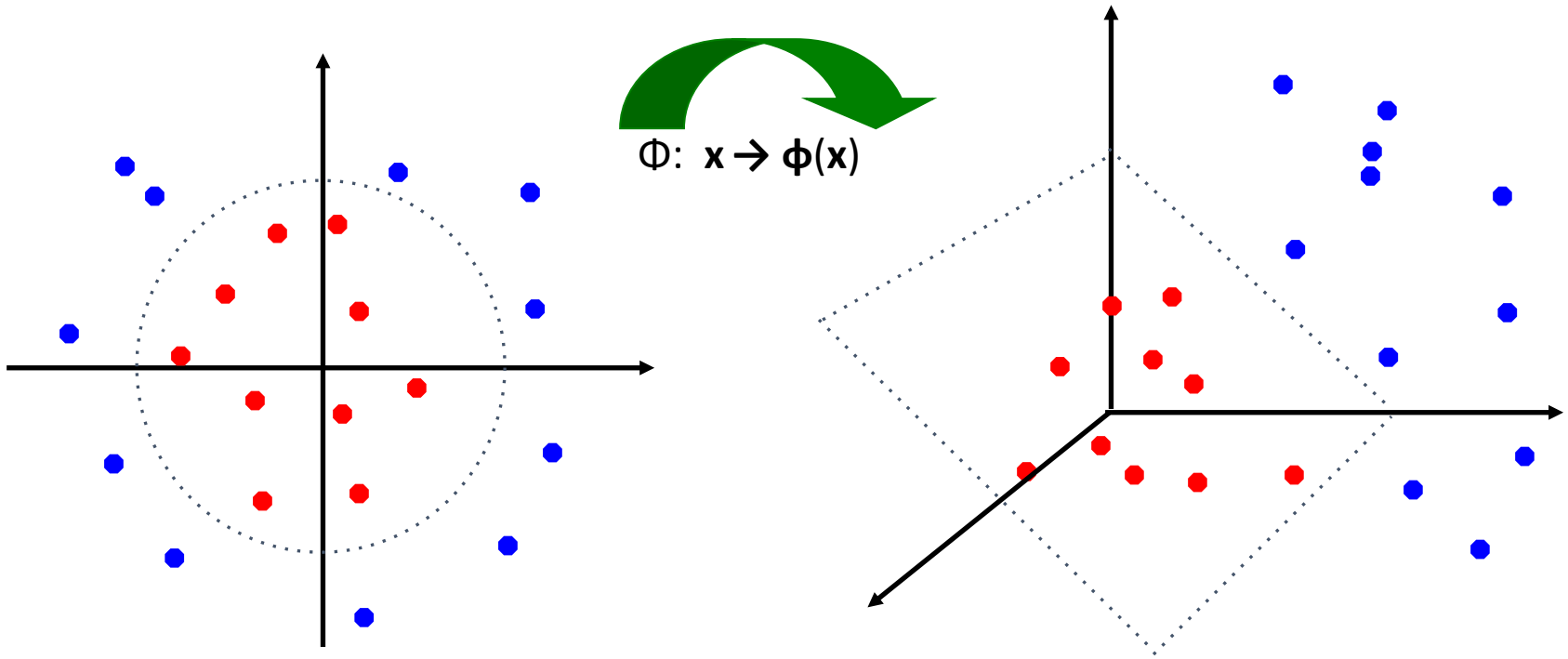
$$\mathbf{w} = \sum_{i=1}^{s} \alpha_{t_i} y_{t_i} \mathbf{x}_{t_i}$$

A Geometrical Interpretation



Non-linear SVMs: Feature spaces

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



Soft Margin Hyperplane

- Define ξ_i=0 if there is no error for x_i
  - ξ_i are just "slack variables" in optimization theory

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

- We want to minimize
  - C : tradeoff parameter between error and margin
- The optimization problem becomes $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Optimization Problem

- The dual of the problem is

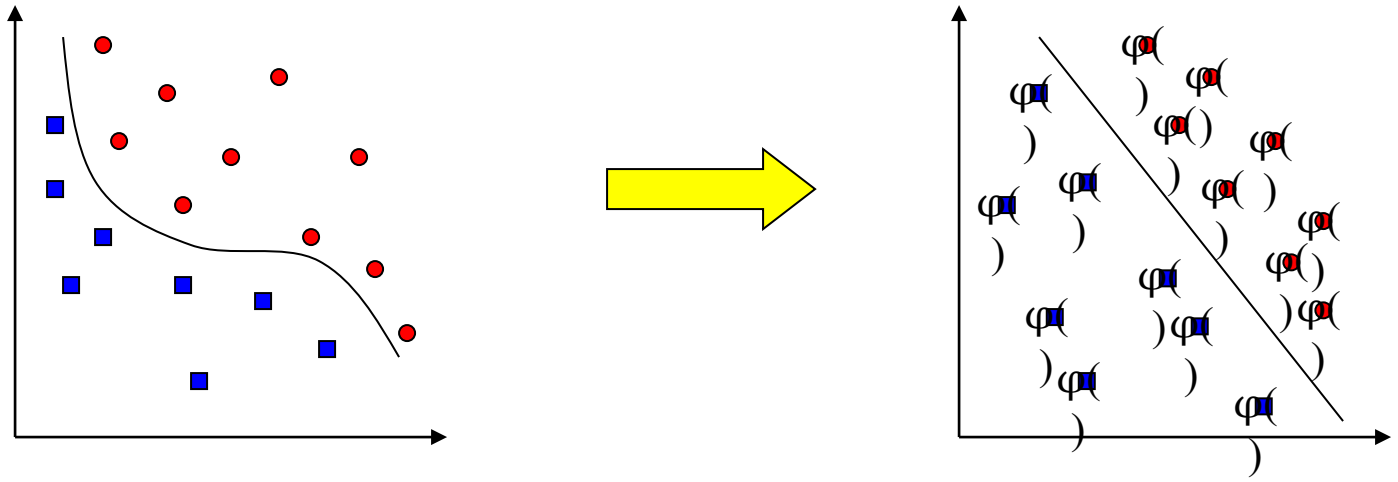
$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- **w** is also recovered as $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- The only difference with the linear separable case is that there is an upper bound C on a_i
- Once again, a QP solver can be used to find a_i

Why does a RBF network work?

- The hidden layer applies a nonlinear transformation from the input space to the hidden space
- In the hidden space a linear discrimination can be performed

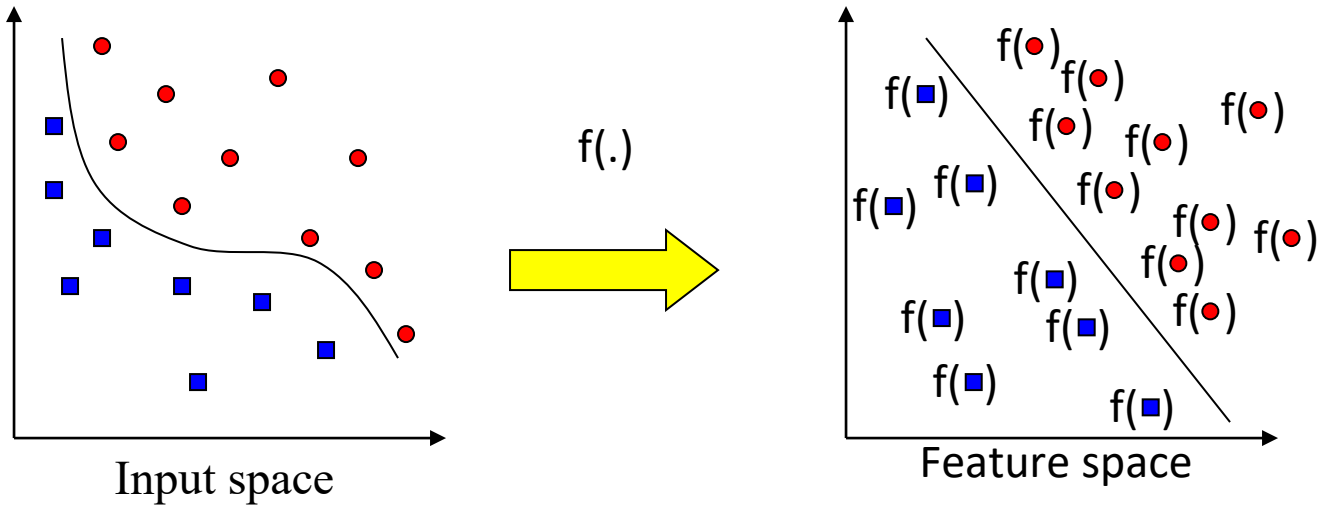


Extension to Non-linear Decision Boundary

- **Key idea:** transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - **Input space:** the space $\mathbf{x}_i$ are in
  - **Feature space:** the space of $f(\mathbf{x}_i)$ after transformation
- **Why transform?**
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - The classification task can be "easier" with a proper transformation. Example: XOR

Extension to Non-linear Decision Boundary

- Possible problem of the transformation
  - High computational burden and hard to get a good estimate
- SVM solves these two issues simultaneously
  - Kernel tricks for efficient computation
  - Minimize $||w||^2$ can lead to a "good" classifier



Example Transformation

- Define the kernel function *K* (**x**,**y**) as $K(\mathbf{x}, \mathbf{y}) = (1 + x_1 y_1 + x_2 y_2)^2$

- Consider the following transformation

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$

$$\phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) = (1, \sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, \sqrt{2}y_1 y_2)$$

- The inner product can be computed by *K* without going through the map f(.)

$$\begin{aligned} \langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle &= (1 + x_1 y_1 + x_2 y_2)^2 \\ &= K(\mathbf{x}, \mathbf{y}) \end{aligned}$$

Kernel Trick

- The relationship between the kernel function *K* and the mapping f(.) is

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$$

  - This is known as the kernel trick
- In practice, we specify *K*, thereby specifying f(.) indirectly, instead of choosing f(.)
- Intuitively, *K* (**x**,**y**) represents our desired notion of similarity between data **x** and **y** and this is from our prior knowledge
- *K* (**x**,**y**) needs to satisfy a technical condition (Mercer condition) in order for f(.) to exist

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width s

  - Closely related to radial basis function neural networks

- Sigmoid with parameter k and q

$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - It does not satisfy the Mercer condition on all k and q

- Research on different kernel functions in different applications is very active

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$