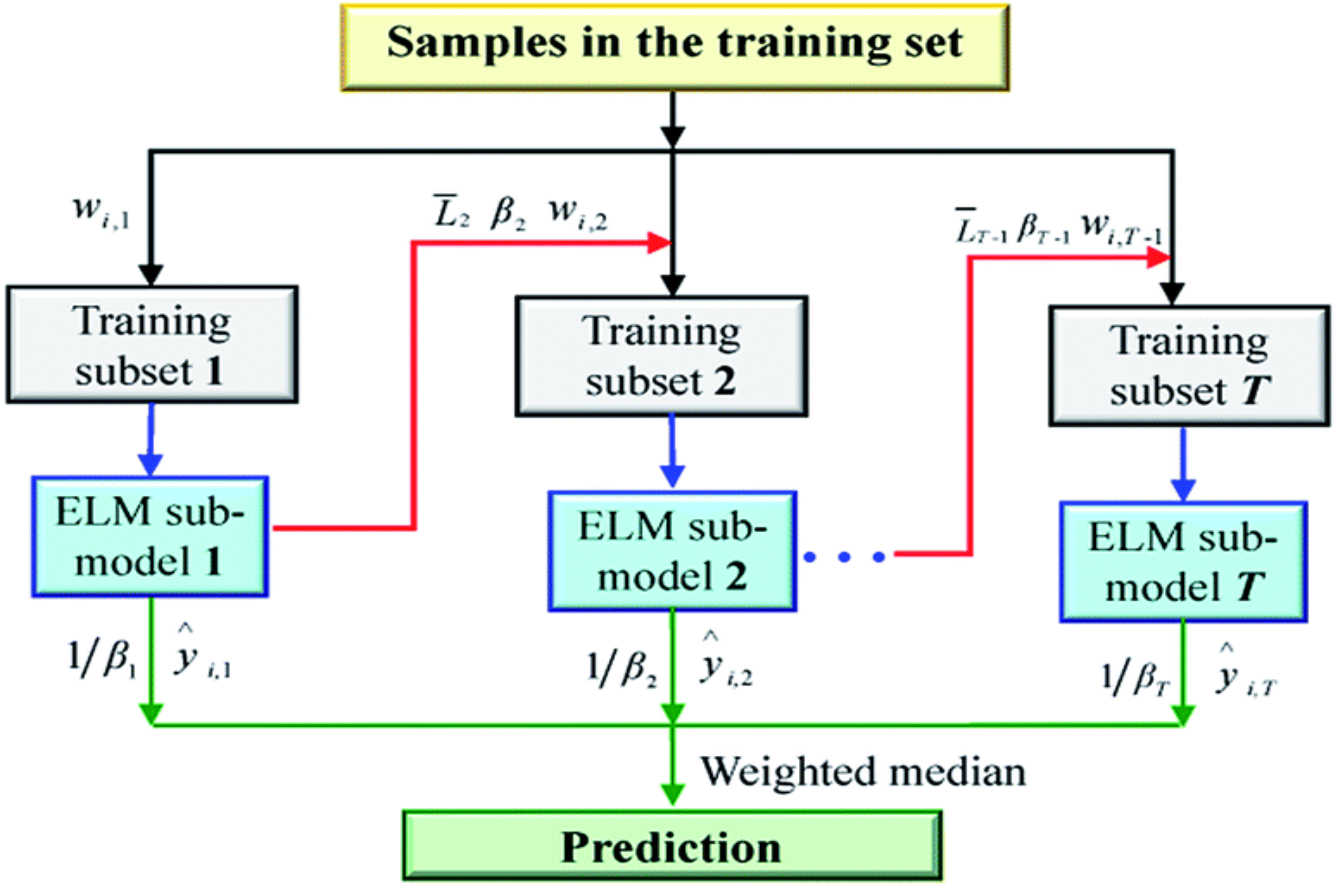
Mohamed Noordeen Alaudeen

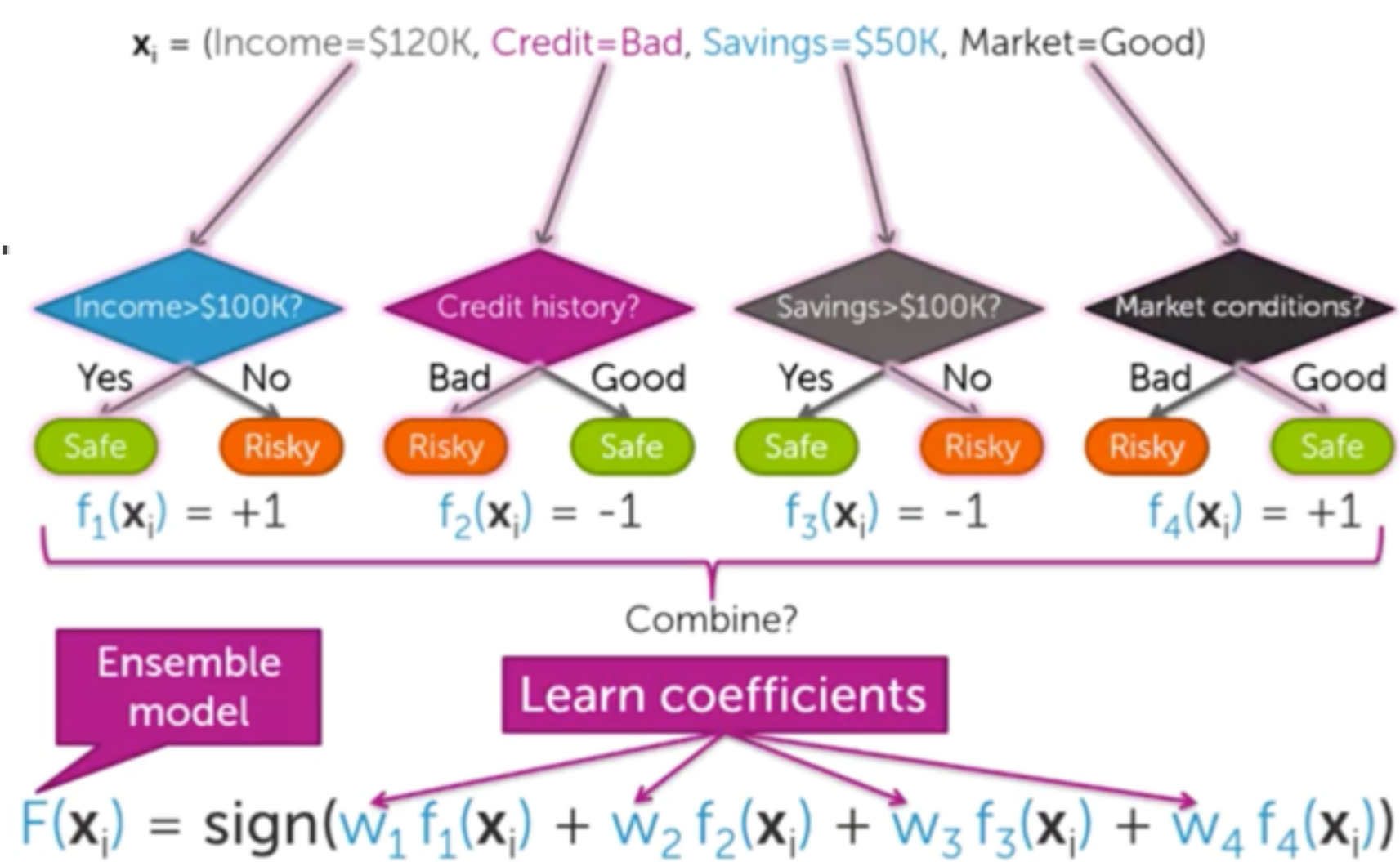
Senior Data Scientist – Logitech Boosting

Boosting

Competition winning algorithm

Boosting


Samples in the training set
$w_{i,1}$
$\overline{L}_2\ \beta_2\ w_{i,2}$
$\overline{L}_{T-1}\ \beta_{T-1}\ w_{i,T-1}$
Training subset 1
Training subset 2
Training subset T
ELM sub-model 1
ELM sub-model 2
ELM sub-model T
$1/\beta_1$ $\hat{y}_{i,1}$
$1/\beta_2$ $\hat{y}_{i,2}$
$1/\beta_T$ $\hat{y}_{i,T}$
Weighted median
Prediction


x_i = (Income=$120K, Credit=Bad, Savings=$50K, Market=Good)
Income>$100K?
Yes
No
Safe
Risky
Credit history?
Bad
Good
Risky
Safe
Savings>$100K?
Yes
No
Safe
Risky
Market conditions?
Bad
Good
Risky
Safe
$f_1(x_i) = +1$
$f_2(x_i) = -1$
$f_3(x_i) = -1$
$f_4(x_i) = +1$
Combine?
Ensemble model
Learn coefficients
$F(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))$

Prediction with ensemble

Combine?

Ensemble model

Learn coefficients

$$F(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))$$

w_1	2
w_2	1.5
w_3	1.5
w_4	0.5

Prediction with ensemble

Ensemble model

Learn coefficients

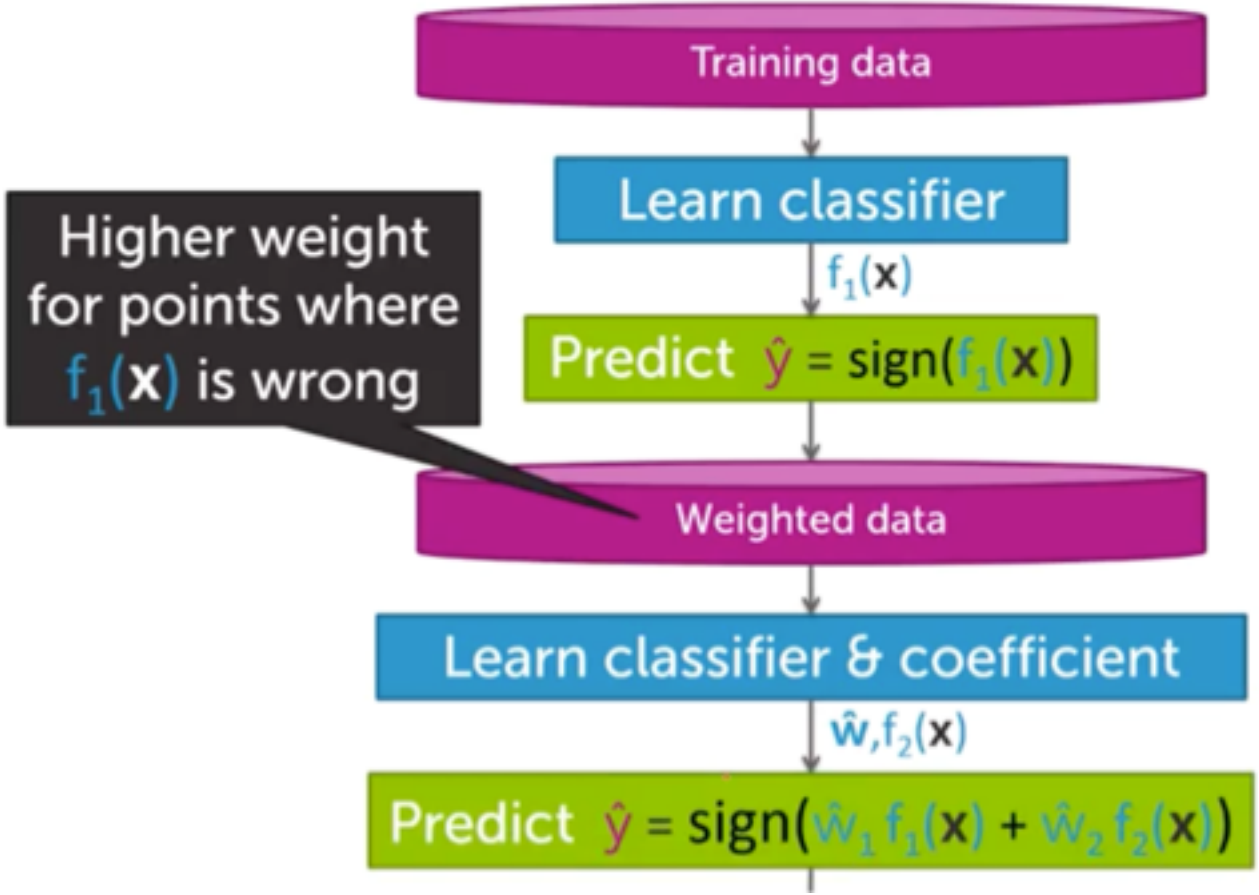
$$\hat{F}(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))$$

$= \text{sign}(2 \times +1 + 1.5 \times -1 + 1.5 \times -1 + 0.5 \times +1)$

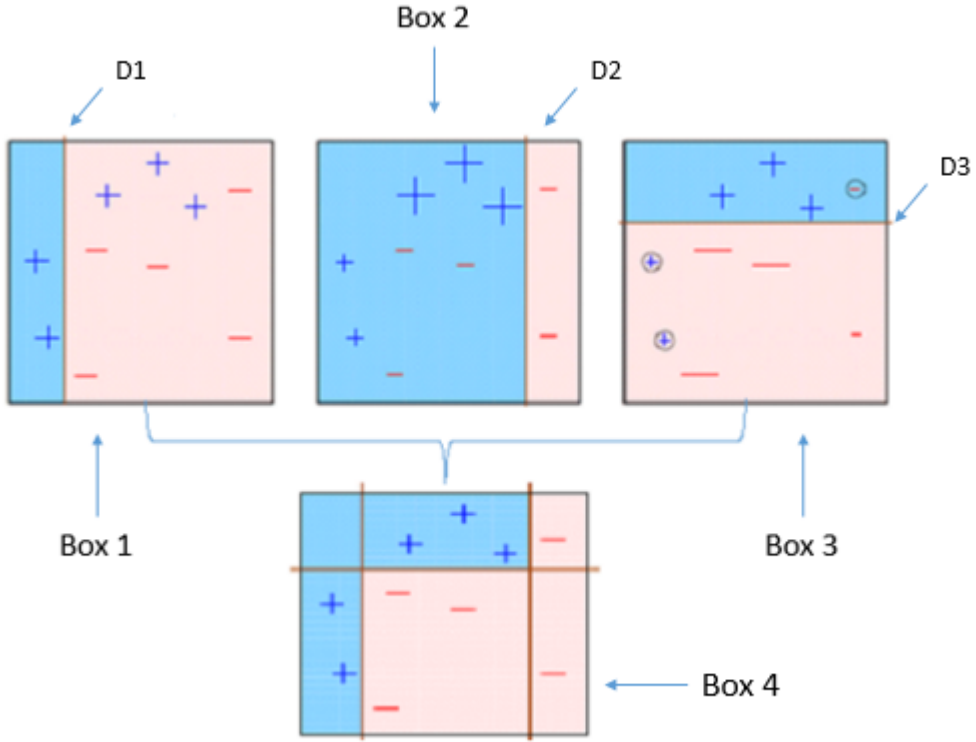
$= \text{sign}(-0.5) \Rightarrow \hat{y}_i = -1$

w_1	2
w_2	1.5
w_3	1.5
w_4	0.5

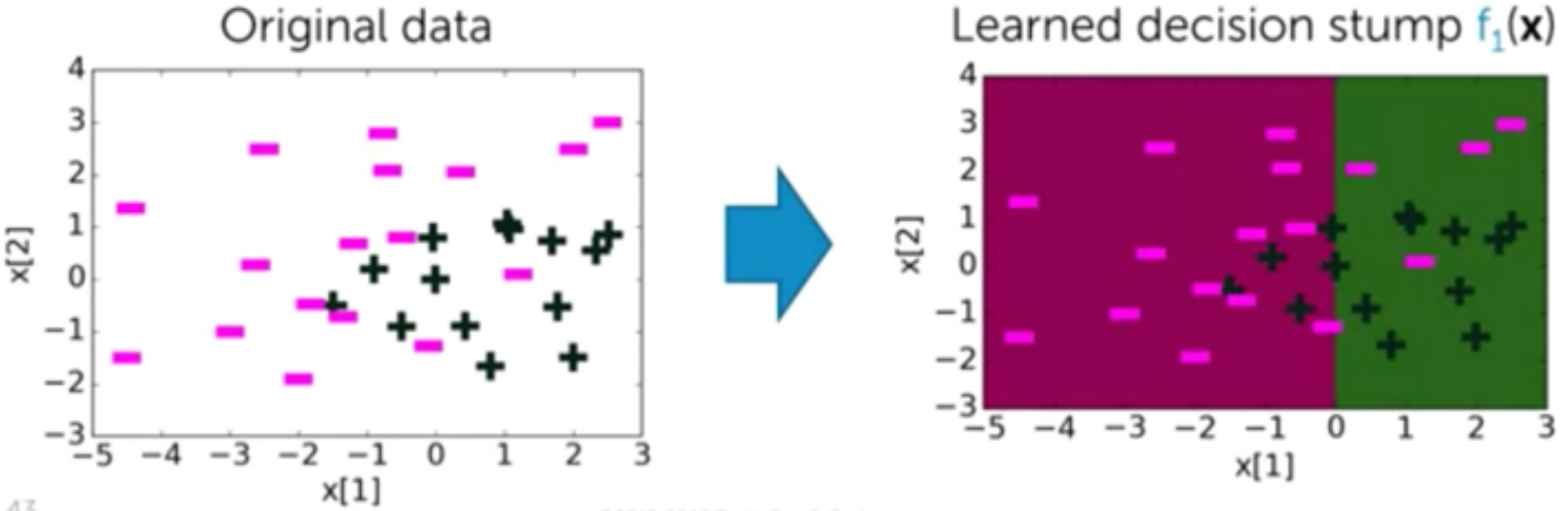
Boosting = Greedy learning ensembles from data



AdaBoost

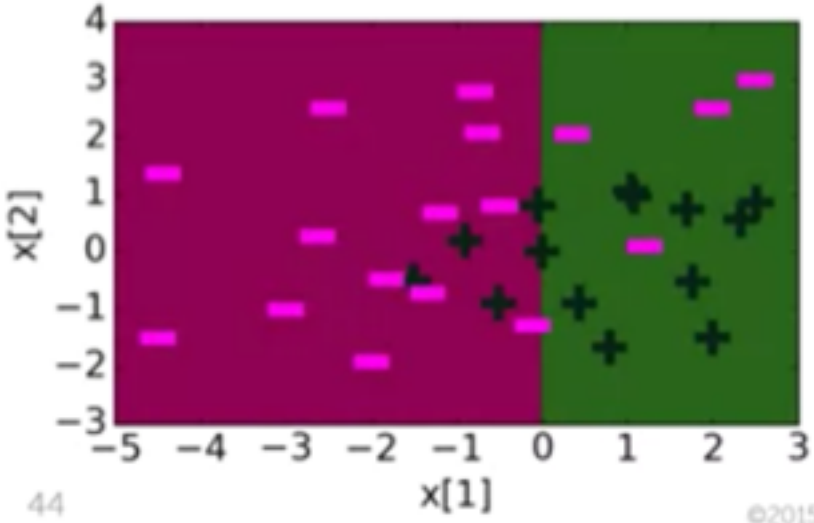


Iteration = 1

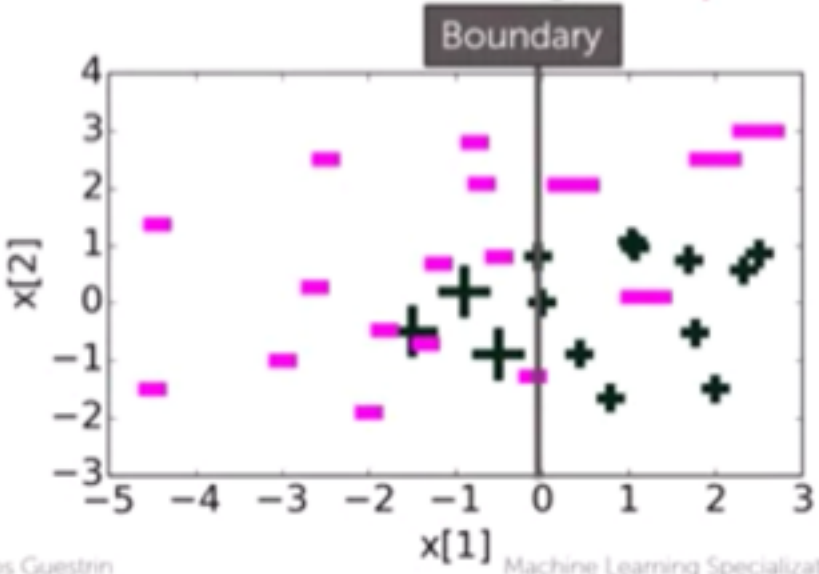


Updated Weights

Learned decision stump $f_1(\mathbf{x})$

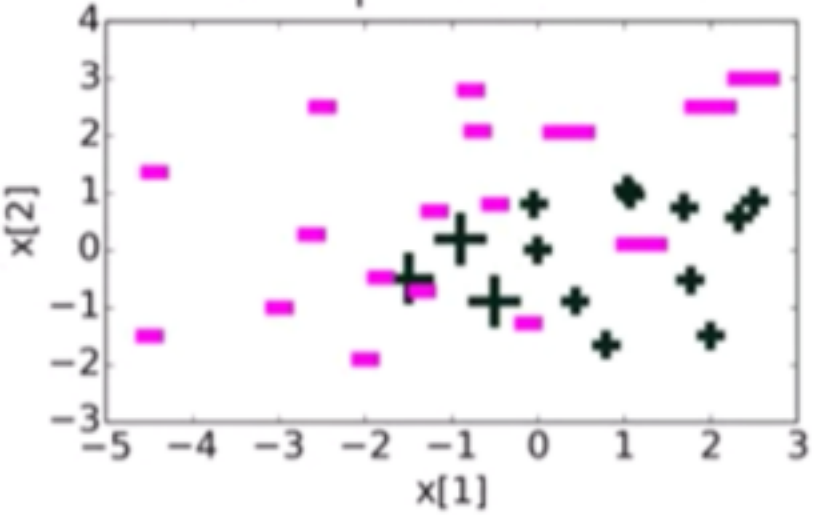


New data weights α_i

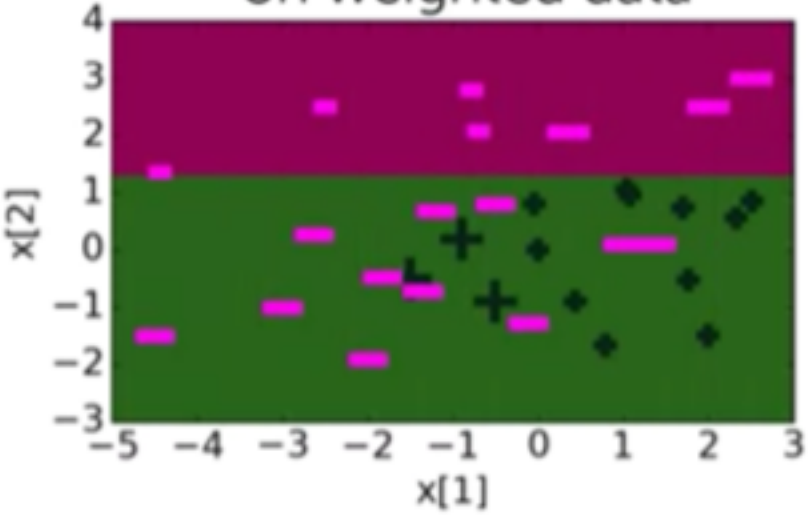


Iteration = 2

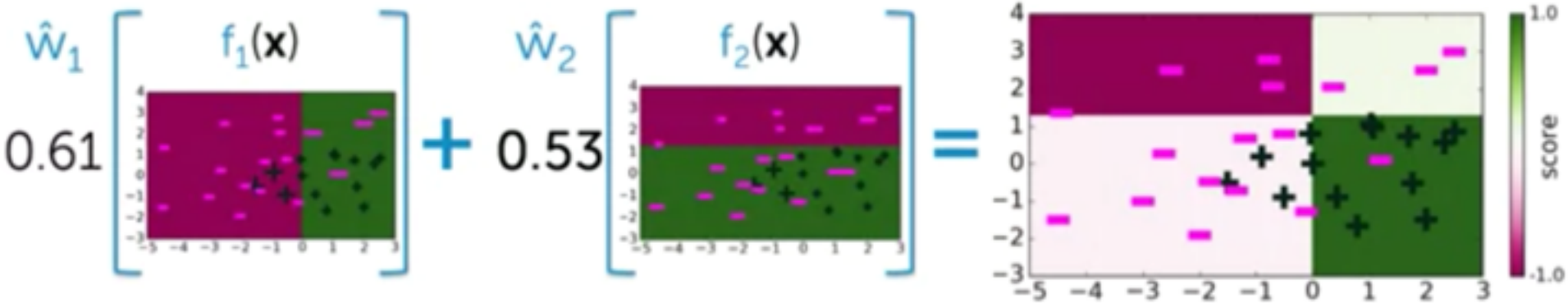
Weighted data: using α_i chosen in previous iteration



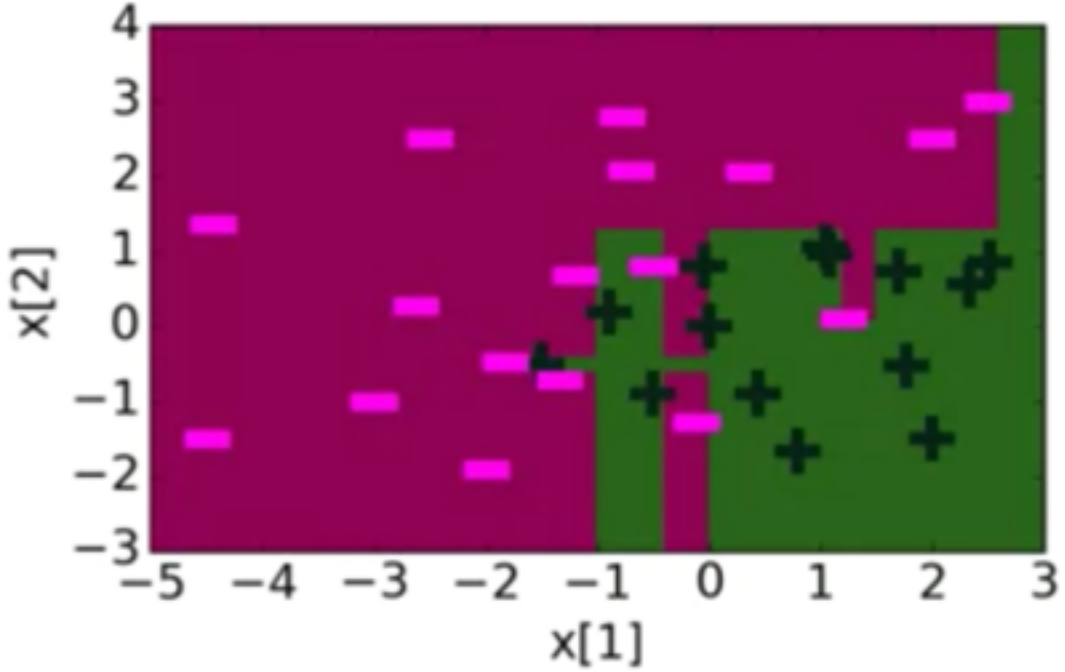
Learned decision stump $f_2(\mathbf{x})$ on weighted data



Ensemble becomes Weighted sum of learned classifiers



Decision boundary of ensemble classifier after 30 iterations



training_error = 0

AdaBoost: learning ensemble

- Start same weight for all points: $\alpha_i = 1/N$

- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i

- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

Finding best next decision stump

Consider splitting on each feature:

Finding best next decision stump

Consider splitting on each feature:

weighted_error
= 0.2

Finding best next decision stump

Consider splitting on each feature:

weighted_error = 0.2

weighted_error = 0.35

Finding best next decision stump

Consider splitting on each feature:

weighted_error = 0.2

weighted_error = 0.35

weighted_error = 0.3

Finding best next decision stump

Consider splitting on each feature:

weighted_error = 0.2

weighted_error = 0.35

weighted_error = 0.3

weighted_error = 0.4

Finding best next decision stump

Consider splitting on each feature:

Finding best next decision stump

Consider splitting on each feature:

$$\hat{W}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right) = 0.69$$

Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

Updating weights α_i

Credit	Income	y
A	$130K	Safe
B	$80K	Risky
C	$110K	Risky
A	$110K	Safe
A	$90K	Safe
B	$120K	Safe
C	$30K	Risky
C	$60K	Risky
B	$95K	Safe
A	$60K	Safe
A	$98K	Safe

Updating weights α_i

Credit	Income	y	ŷ
A	$130K	Safe	Safe
B	$80K	Risky	Risky
C	$110K	Risky	Safe
A	$110K	Safe	Safe
A	$90K	Safe	Risky
B	$120K	Safe	Safe
C	$30K	Risky	Risky
C	$60K	Risky	Risky
B	$95K	Safe	Risky
A	$60K	Safe	Risky
A	$98K	Safe	Risky

Updating weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	
B	$80K	Risky	Risky	1.5	
C	$110K	Risky	Safe	1.5	
A	$110K	Safe	Safe	2	
A	$90K	Safe	Risky	1	
B	$120K	Safe	Safe	2.5	
C	$30K	Risky	Risky	3	
C	$60K	Risky	Risky	2	
B	$95K	Safe	Risky	0.5	
A	$60K	Safe	Risky	1	
A	$98K	Safe	Risky	0.5	

Updating weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} = \alpha_i e^{-0.69} = \alpha_i / 2 & \text{, if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t} = \alpha_i e^{0.69} = 2\,\alpha_i & \text{, if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	
B	$80K	Risky	Risky	1.5	
C	$110K	Risky	Safe	1.5	
A	$110K	Safe	Safe	2	
A	$90K	Safe	Risky	1	
B	$120K	Safe	Safe	2.5	
C	$30K	Risky	Risky	3	
C	$60K	Risky	Risky	2	
B	$95K	Safe	Risky	0.5	
A	$60K	Safe	Risky	1	
A	$98K	Safe	Risky	0.5	

Updating weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} = \alpha_i e^{-0.69} = \alpha_i / 2 & \text{, if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t} = \alpha_i e^{0.69} = 2\,\alpha_i & \text{, if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	0.5/2 = 0.25
B	$80K	Risky	Risky	1.5	0.75
C	$110K	Risky	Safe	1.5	2 * 1.5 = 3
A	$110K	Safe	Safe	2	1
A	$90K	Safe	Risky	1	2
B	$120K	Safe	Safe	2.5	1.25
C	$30K	Risky	Risky	3	1.5
C	$60K	Risky	Risky	2	1
B	$95K	Safe	Risky	0.5	1
A	$60K	Safe	Risky	1	2
A	$98K	Safe	Risky	0.5	1

CLASSIFICATION: AdaBoost Algorithm

- Given training data

$$(x_1, y_1), \ldots, (x_n, y_n)$$

$$y_i \in \{-1, +1\}$$

x_i is instance

y_i is classification

CLASSIFICATION: AdaBoost Algorithm

- For k = 1, …, K
  - Create distribution D_k on {1, …, n}
  - Select weak classifier with smallest error on D_k

$$h_k : X \rightarrow \{-1, +1\}$$

$$\varepsilon_k = \Pr_{D_k}\left[h_k(x_i) \neq y_i\right]$$

- Output single final classifier

CLASSIFICATION: AdaBoost Algorithm

- Constructing *Dk*
  - $$D_1(i) = \frac{1}{n}, \quad D_{k+1} = \frac{D_k(i)}{Z_k} c(x)$$

$$c(x) = \begin{cases} e^{-\alpha_k}, & y_i = h_k(x_i) \\ e^{\alpha_k}, & y_i \neq h_k(x_i) \end{cases}$$

$$\alpha_k = \frac{1}{2} \ln \frac{1-\varepsilon_k}{\varepsilon_k} > 0$$

$$Z_k = \text{ normalization constant}$$

$$H_{final}(x) = \text{sign}\left(\sum_k \alpha_k h_k(x) \right)$$

CLASSIFICATION: AdaBoost Example

- Training data:

Index	0	1	2	3	4	5	6	7	8	9
X value	0	1	2	3	4	5	6	7	8	9
Y value	1	1	1	-1	-1	-1	1	1	1	-1

CLASSIFICATION: AdaBoost Example

- Weak learner:
  - Hypotheses of form $x < T$ or $x > T$.
  - Threshold T to be determined so as to minimize the probability of error over the entire data.
  - Start with following probabilities for $k = 1$:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X value	0	1	2	3	4	5	6	7	8	9
Y value	1	1	1	-1	-1	-1	1	1	1	-1
p_0	**p_1**	**p_2**	**p_3**	**p_4**	**p_5**	**p_6**	**p_7**	**p_8**	**p_9**	
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	

CLASSIFICATION: AdaBoost Example

- Best threshold is between 2 and 3

$$h_1(x) = I(X < 2.5)$$

$$\varepsilon_1 = 0.3$$

$$\alpha_1 = 0.423649$$

$$c(x) = \begin{cases} e^{-0.423649} = 0.654654, & \text{for correct} \\ e^{0.423649} = 1.52753, & \text{for wrong} \end{cases}$$

CLASSIFICATION: AdaBoost Example

- Update the probabilities

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X value	0	1	2	3	4	5	6	7	8	9
Y value	1	1	1	-1	-1	-1	1	1	1	-1

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Correct	Y	Y	Y	Y	Y	Y	N	N	N	Y
Old p_i	.1	.1	.1	.1	.1	.1	.1	.1	.1	.1
Pre-norm p_i	.066	.066	.066	.066	.066	.066	.15	.15	.15	.066
Z_1 =0.916515										
New p_i	.071	.071	.071	.071	.071	.071	.17	.17	.17	.071

CLASSIFICATION: AdaBoost Example

$$f_1(x) = 0.423649 I(x < 2.5), \quad 3 \text{ mistakes}$$

- $k = 2$.
  - Now a threshold between 2 and 3 results in error of 0.5, but between 5 and 6 gives error of 0.28 and between 8 and 9 gives 0.214

$$h_2(x) = I(x < 8.5)$$
$$\varepsilon_2 = 0.214$$
$$\alpha_2 = 0.6496$$

CLASSIFICATION: AdaBoost Example

- Update the probabilities

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Correct	Y	Y	Y	N	N	N	Y	Y	Y	Y
Pre-norm p_i	.037	.037	.037	.137	.137	.137	.087	.087	.087	.037
Z_1=.82										
New p_i	.045	.045	.045	.167	.167	.167	.106	.106	.106	.045

CLASSIFICATION: AdaBoost Example

- 3 mistakes
- $k = 3$. $f_2(x) = 0.423649 I(x < 2.5) + 0.6496 I(x < 8.5)$
  - Best threshold between 5 and 6

$$h_3(x) = I(x > 5.5)$$
$$\varepsilon_3 = 0.1818$$
$$\alpha_2 = 0.7520$$

CLASSIFICATION: AdaBoost Example

- Update the probabilities

Index	0	1	2	3	4	5	6	7	8	9
Correct	N	N	N	Y	Y	Y	Y	Y	Y	Y
Pre-norm p_i	.096	.096	.096	.078	.078	.078	.05	.05	.05	.096
Z_1=.77										
New p_i	.125	.125	.125	.102	.102	.102	.064	.064	.064	.125

CLASSIFICATION: AdaBoost Example

$$f_3(x) = 0.423649 I(x < 2.5) + 0.6496 I(x < 8.5) + 0.752 I(x > 5.5)$$

- 0 mistakes

Python Code

```
from sklearn.ensemble import AdaBoostClassifier #For Classification
from sklearn.ensemble import AdaBoostRegressor #For Regression
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()
clf = AdaBoostClassifier(n_estimators=100, base_estimator=dt,learning_rate=1)
#Above I have used decision tree as a base estimator, you can use any ML learner as base esti
mator if it ac# cepts sample weight
clf.fit(x_train,y_train)
```

You can tune the parameters to optimize the performance of algorithms. I've mentioned below the key parameters for tuning:

- **n_estimators:** It controls the number of weak learners.
- **`learning_rate:`**Controls the contribution of weak learners in the final combination. There is a trade-off between `learning_rate` and `n_estimators`.
- **base_estimators**: It helps to specify different ML algorithm.

You can also tune the parameters of base learners to optimize its performance.

Boosting Advantage

Boosting (machine learning)
Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones.

Boosting Disadvantage

Time and computation expensive.

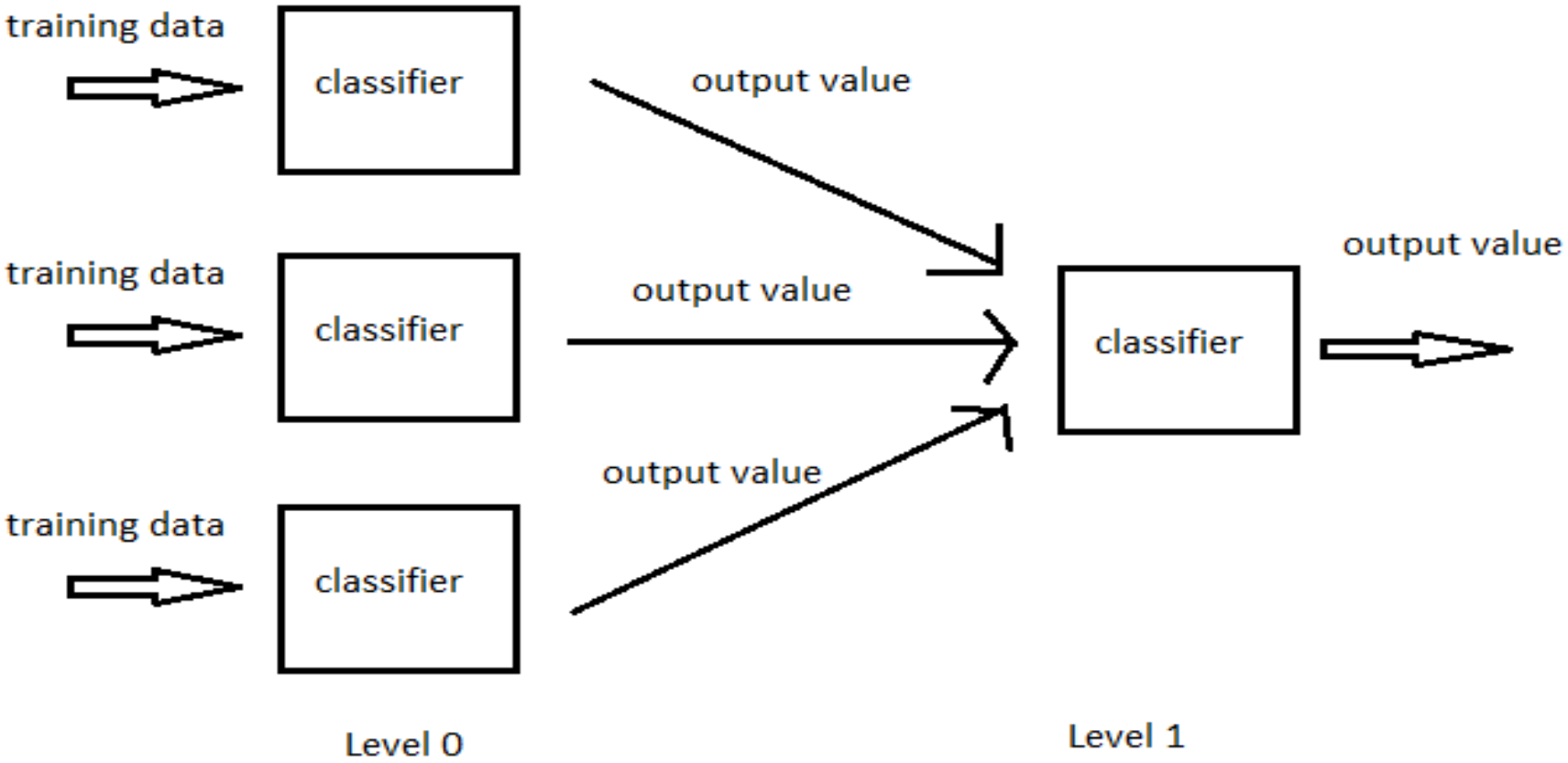
Hard to implement in real time platform.

Complexity of the classification increases.

Stacking

Competition winning algorithm

Concept Diagram of Stacking



```
# Caution! All models and parameter values are just
# demonstrational and shouldn't be considered as recommended.
# Initialize 1st level models.
models = [
    ExtraTreesClassifier(random_state = 0, n_jobs = -1,
        n_estimators = 100, max_depth = 3),

    RandomForestClassifier(random_state = 0, n_jobs = -1,
        n_estimators = 100, max_depth = 3),

    XGBClassifier(seed = 0, n_jobs = -1, learning_rate = 0.1,
        n_estimators = 100, max_depth = 3)]
```

```
# Compute stacking features
S_train, S_test = stacking(models, X_train, y_train, X_test,
    regression = False, metric = accuracy_score, n_folds = 4,
    stratified = True, shuffle = True, random_state = 0, verbose = 2)

# Initialize 2nd level model
model = XGBClassifier(seed = 0, n_jobs = -1, learning_rate = 0.1,
    n_estimators = 100, max_depth = 3)
```

```
# Fit 2nd level model
model = model.fit(S_train, y_train)

# Predict
y_pred = model.predict(S_test)

# Final prediction score
print('Final prediction score: [%.8f]' % accuracy_score(y_test, y_pred))
```

```
task:   [classification]
metric: [accuracy_score]

model 0: [ExtraTreesClassifier]
    fold 0: [0.93548387]
    fold 1: [0.96666667]
    fold 2: [1.00000000]
    fold 3: [0.89655172]
    ----
    MEAN:   [0.95000000]

model 1: [RandomForestClassifier]
    fold 0: [0.87096774]
    fold 1: [0.96666667]
    fold 2: [1.00000000]
    fold 3: [0.93103448]
    ----
    MEAN:   [0.94166667]

model 2: [XGBClassifier]
    fold 0: [0.83870968]
    fold 1: [0.93333333]
    fold 2: [1.00000000]
    fold 3: [0.93103448]
    ----
    MEAN:   [0.92500000]

Final prediction score: [0.96666667]
```

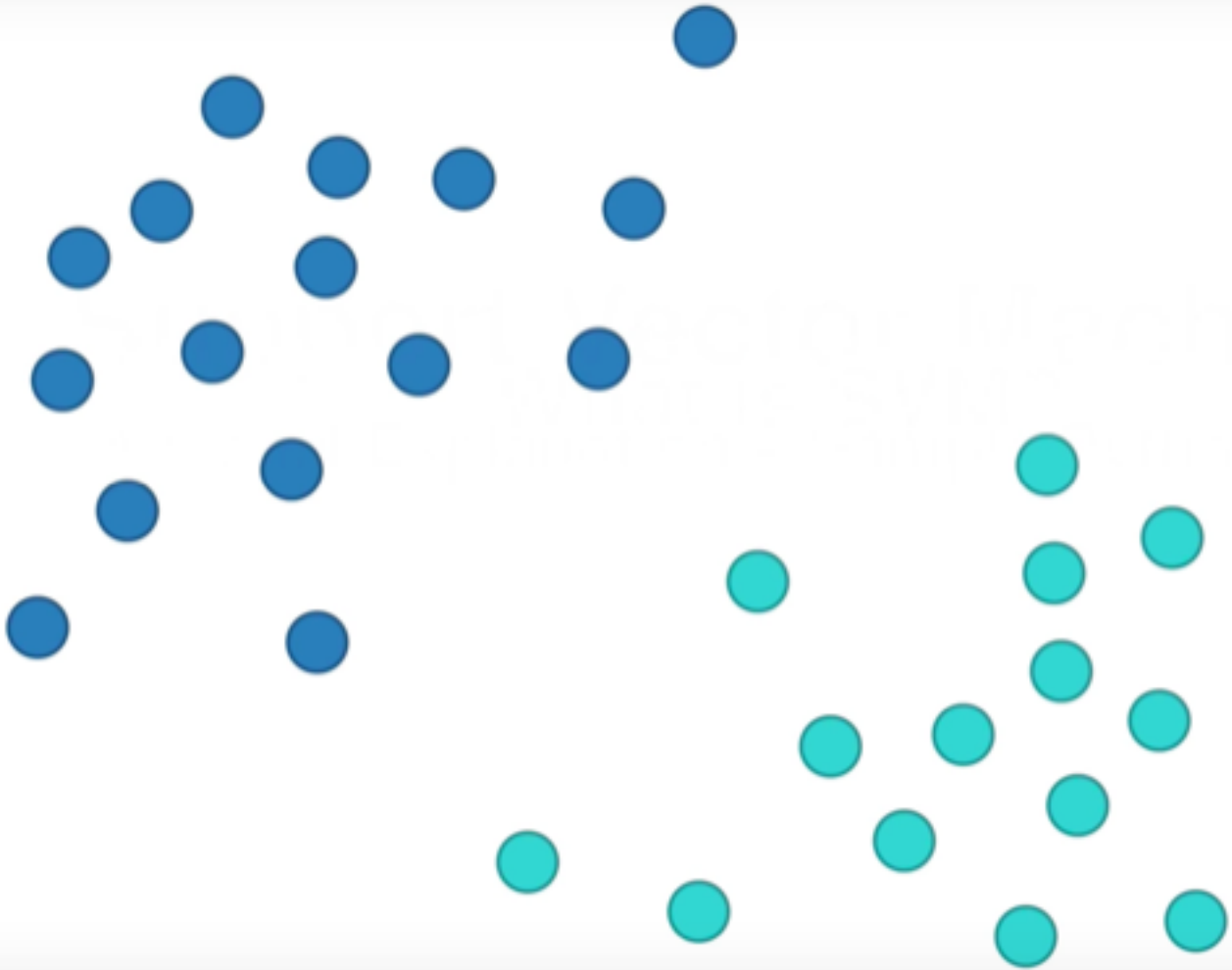
Support Vector Machine

Agenda

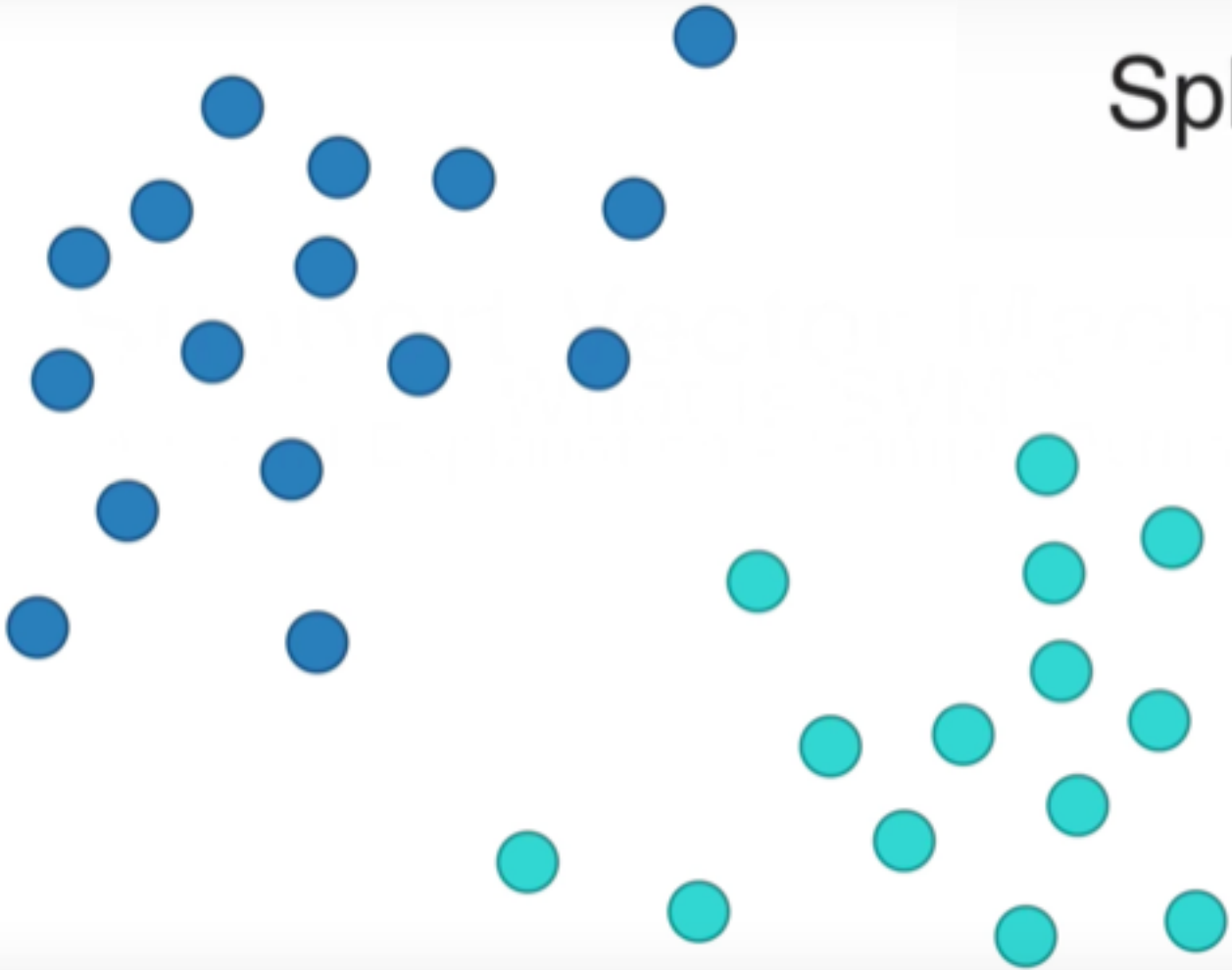
1. **SVM Basics**
   - Visual introduction
   - Example in Python
2. **Additional Complexities**
   - Higher dimensions
   - C parameter
   - Multiple classes
   - Kernel trick
3. **Closing Remarks**
   - Pros and cons
   - Other techniques

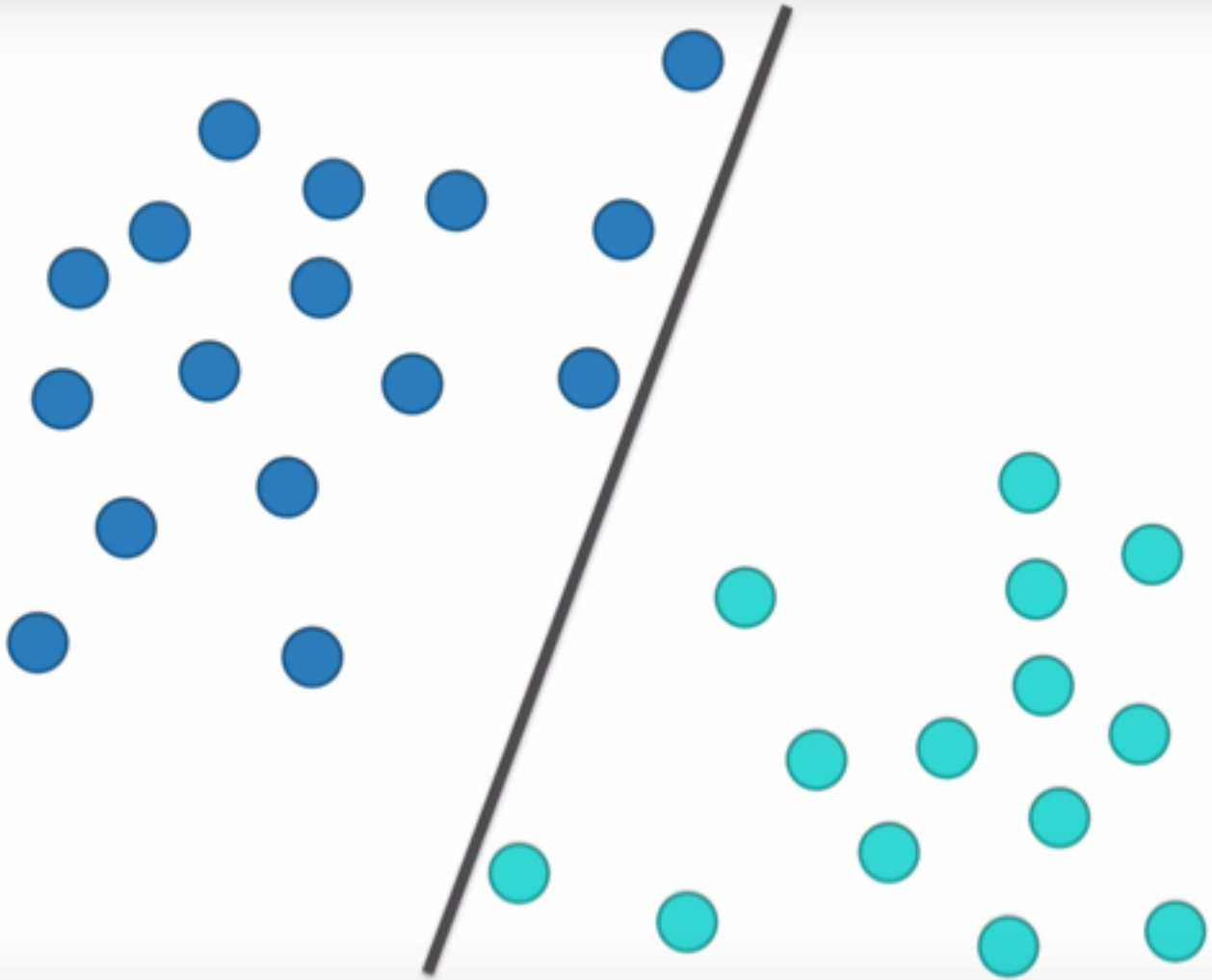
What is SVM?

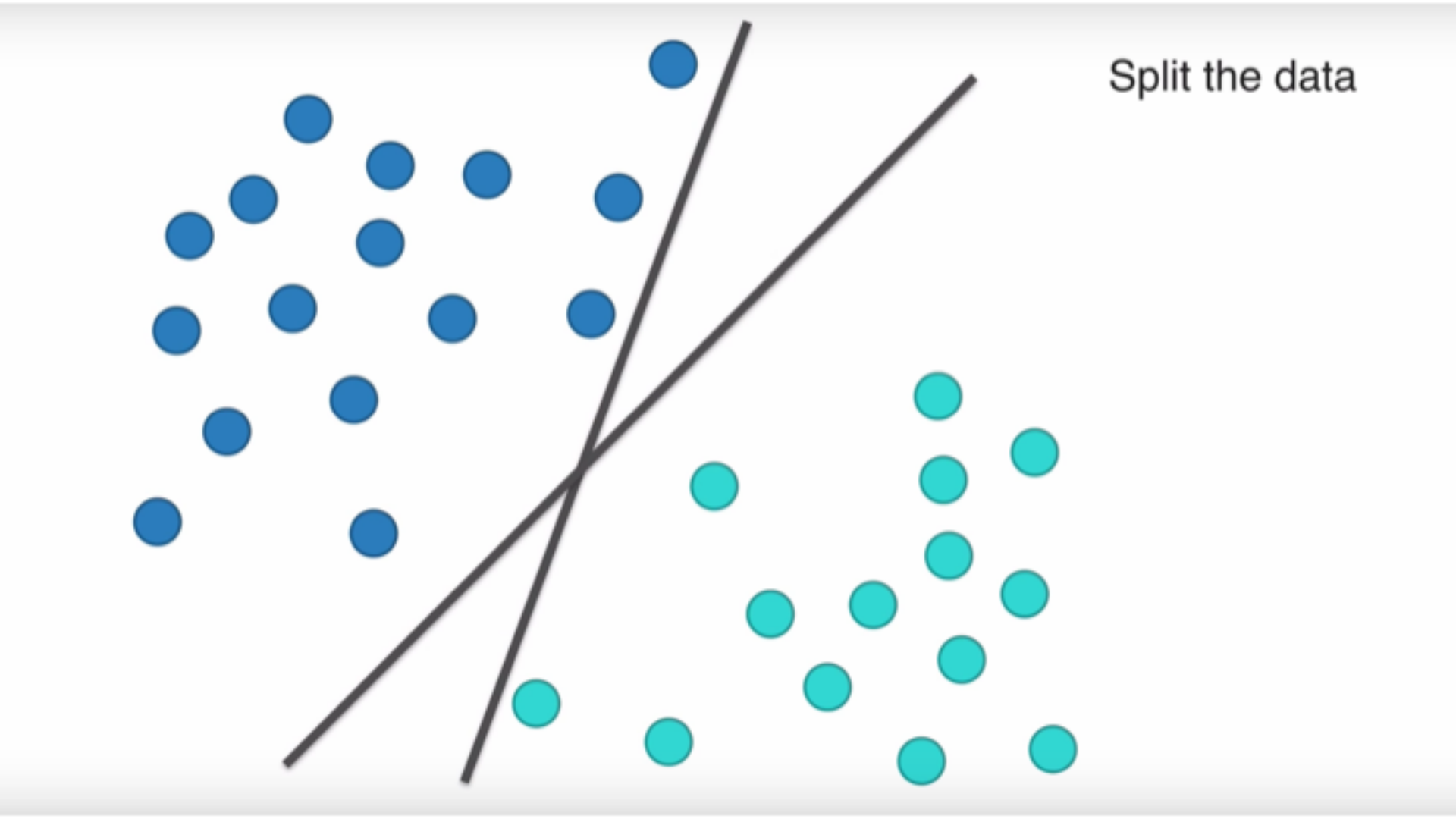
It’s a classification technique.

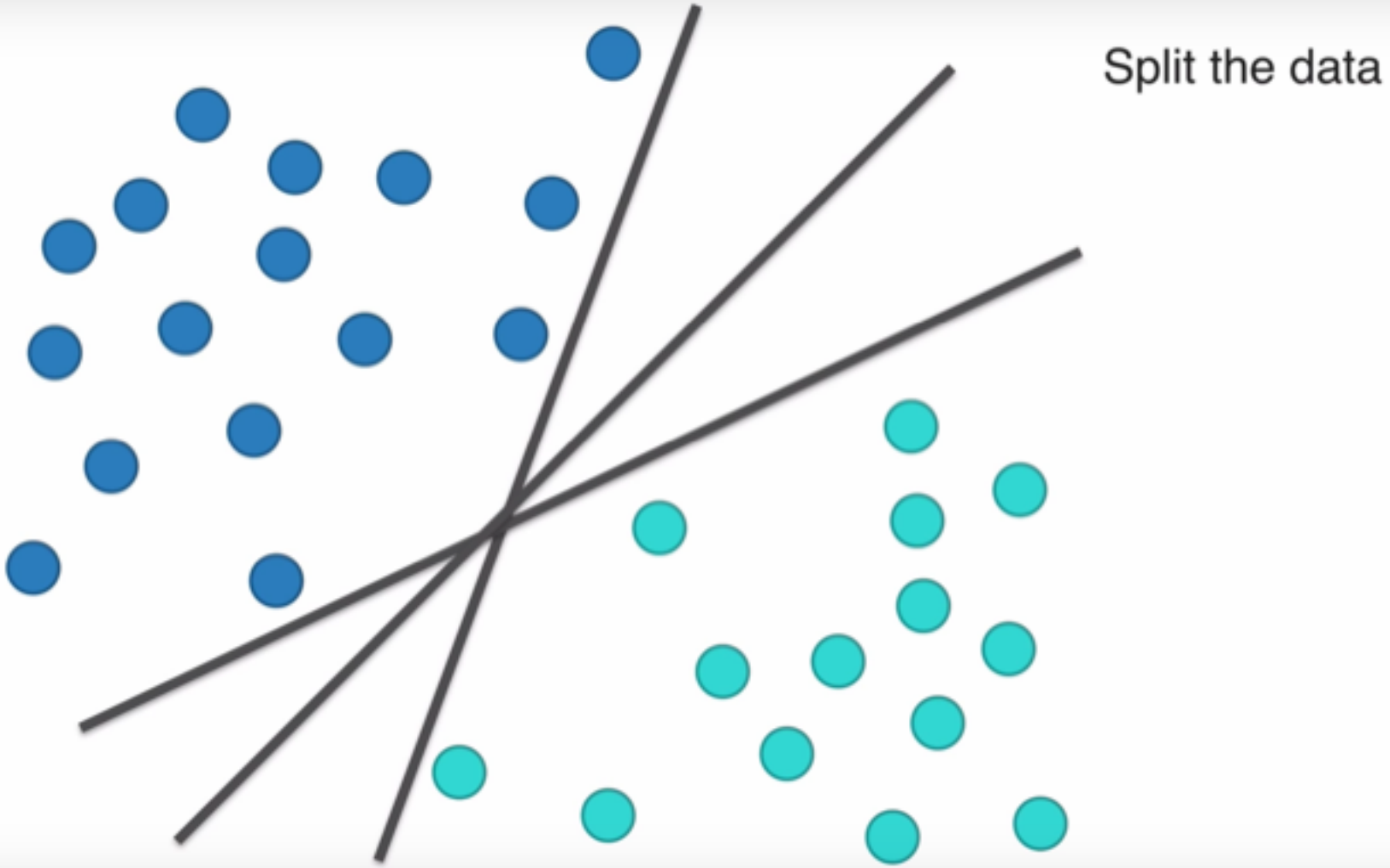


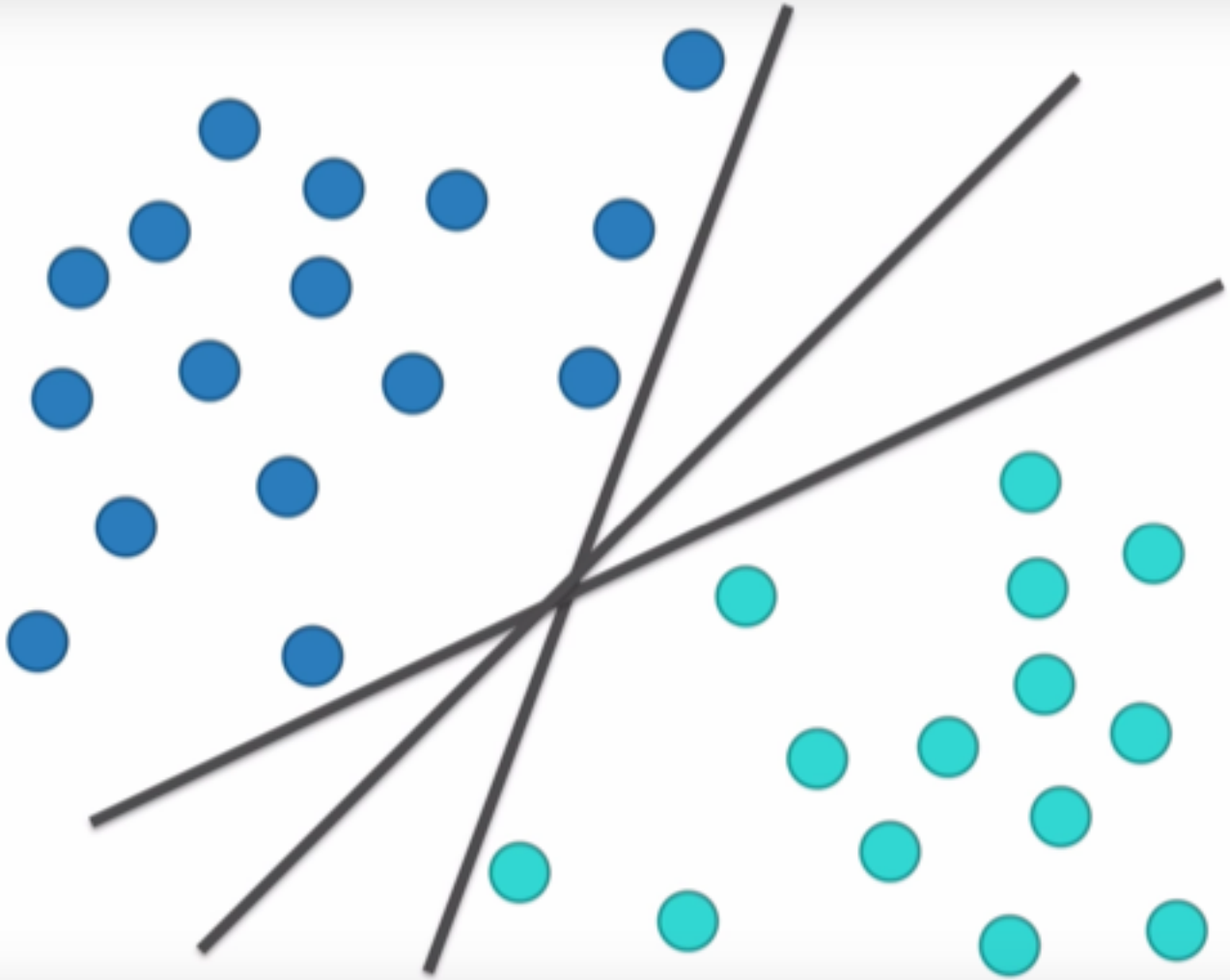
Split the data



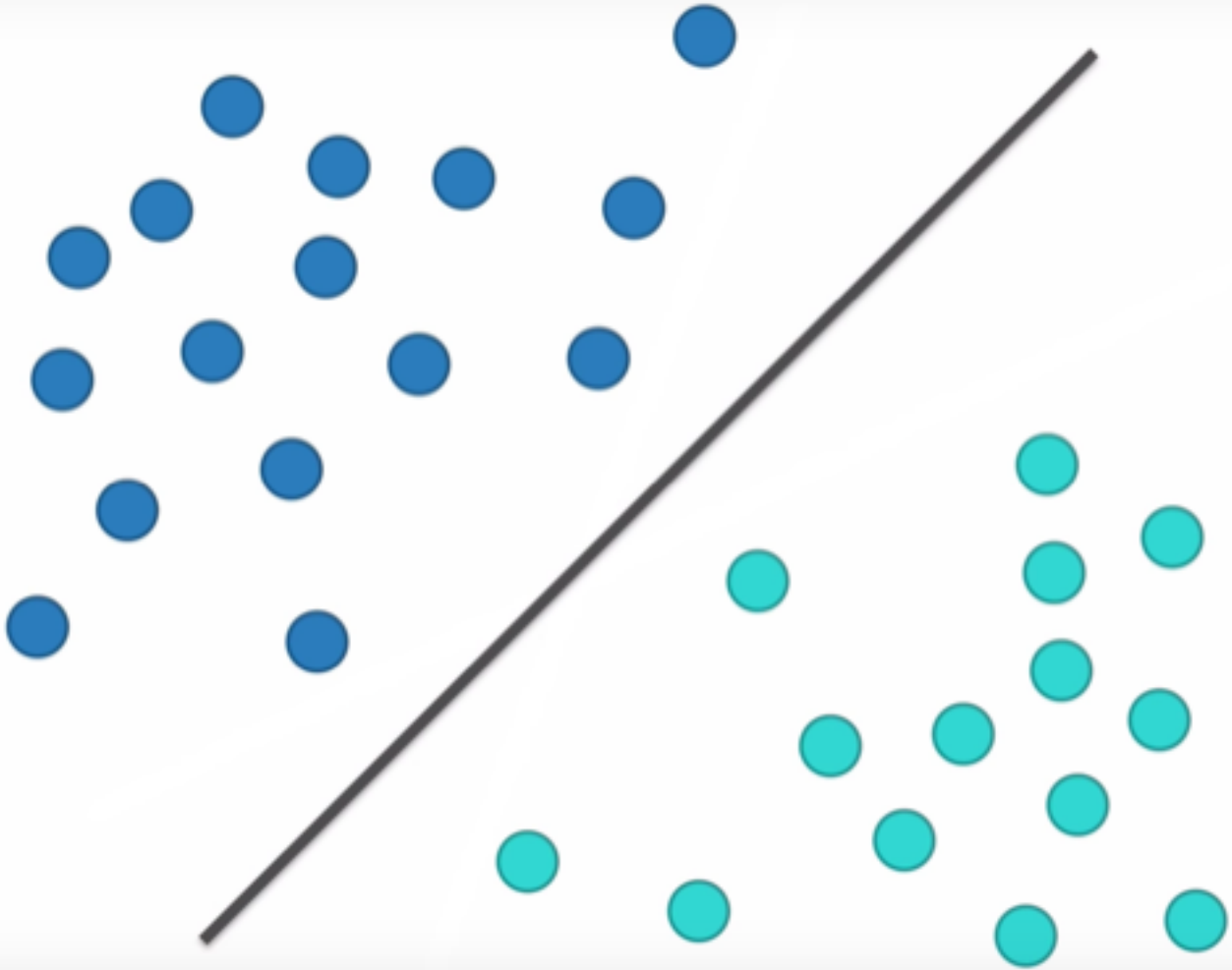

Split the data


Split the data

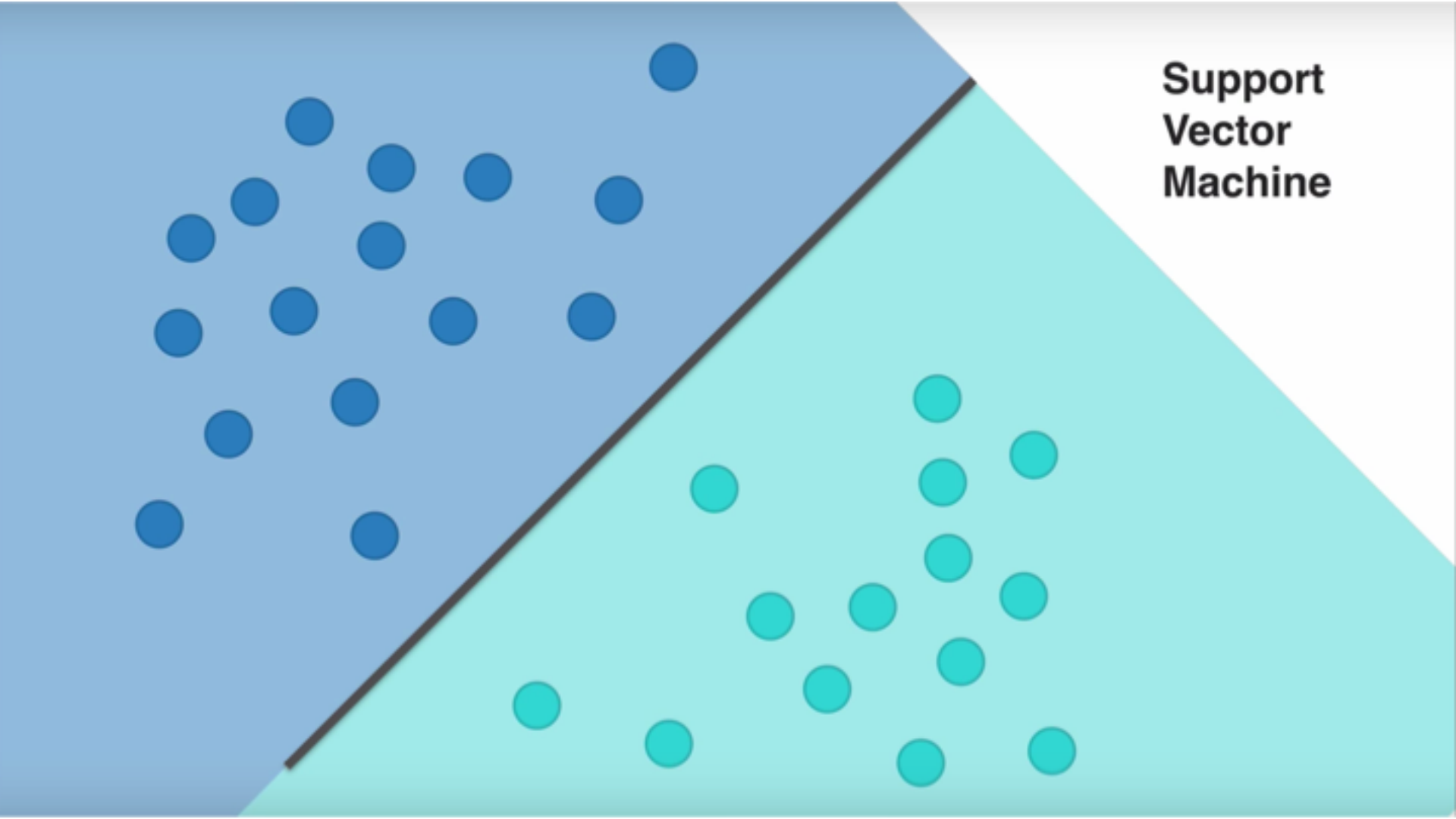

Split the data

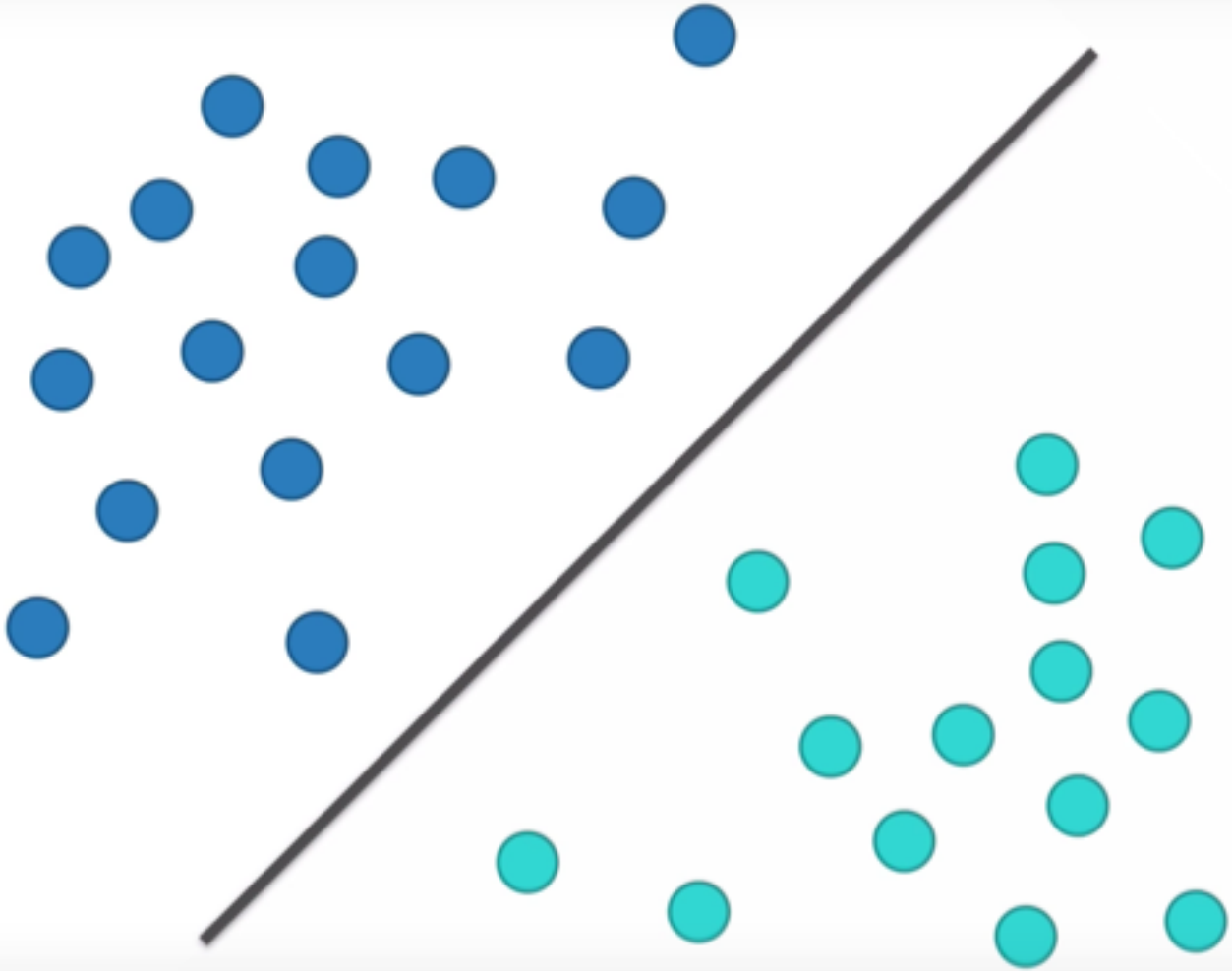


Split the data
in the best possible way

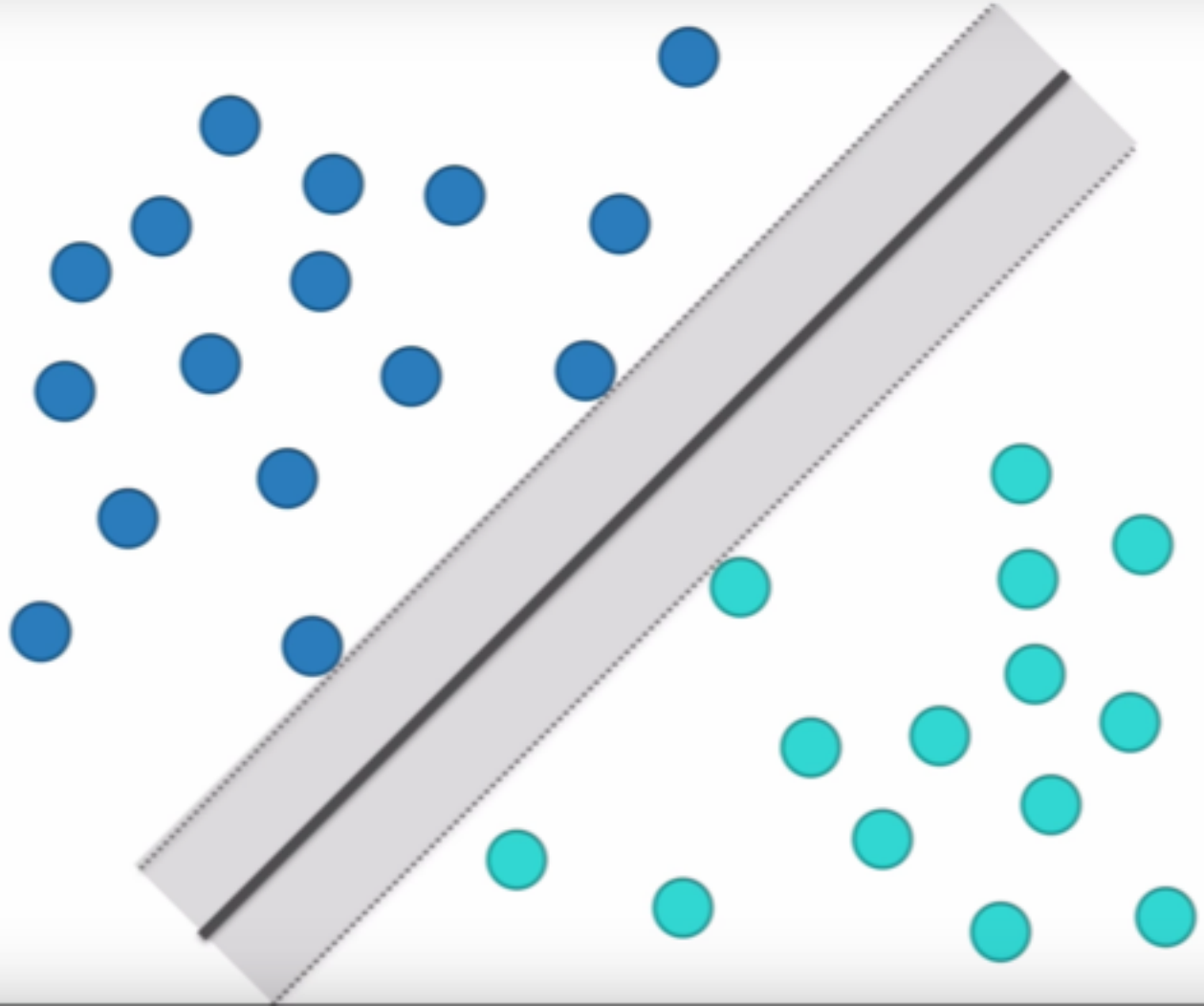


Split the data
in the best possible way

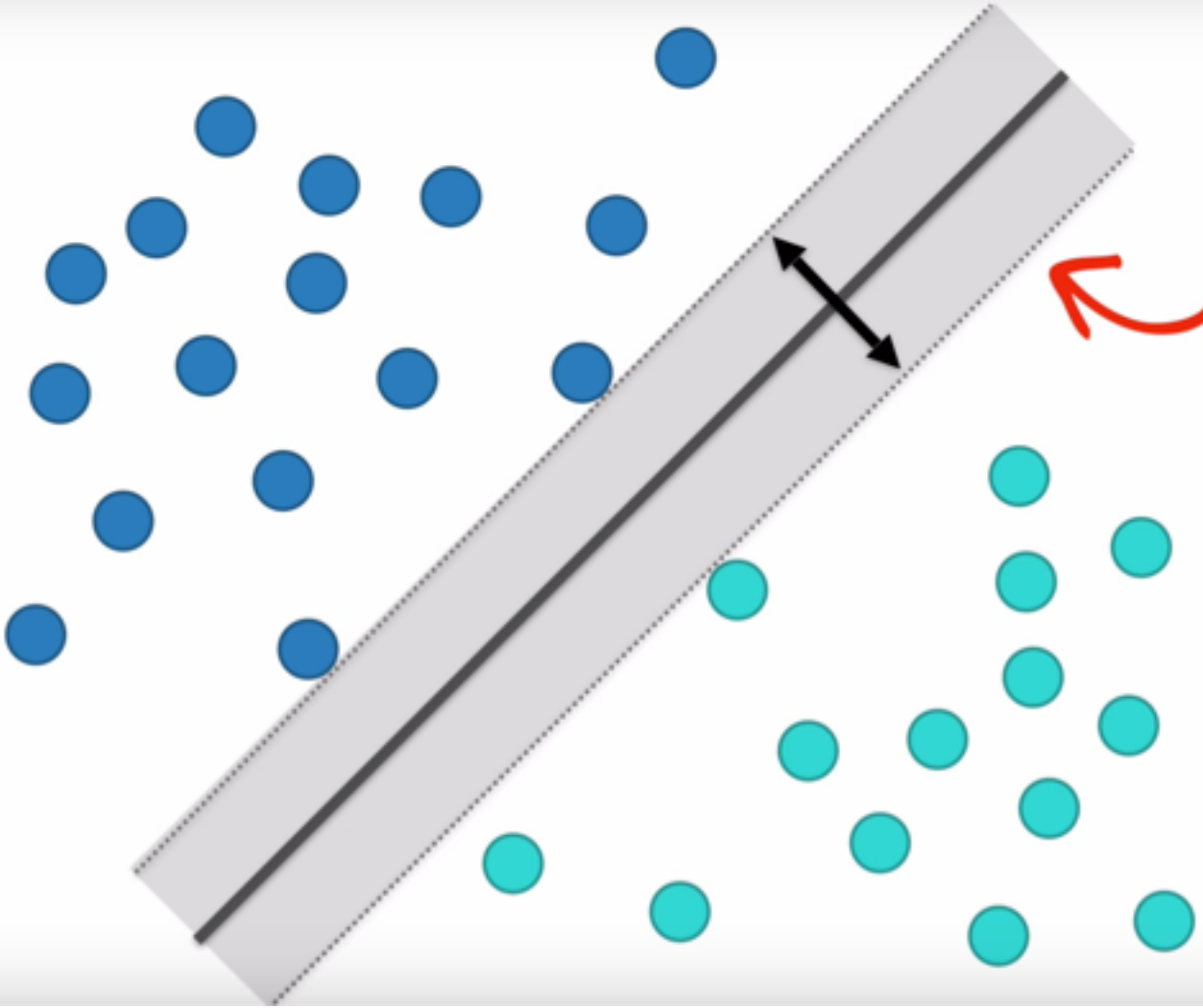

Support
Vector
Machine



Why is this the best split?

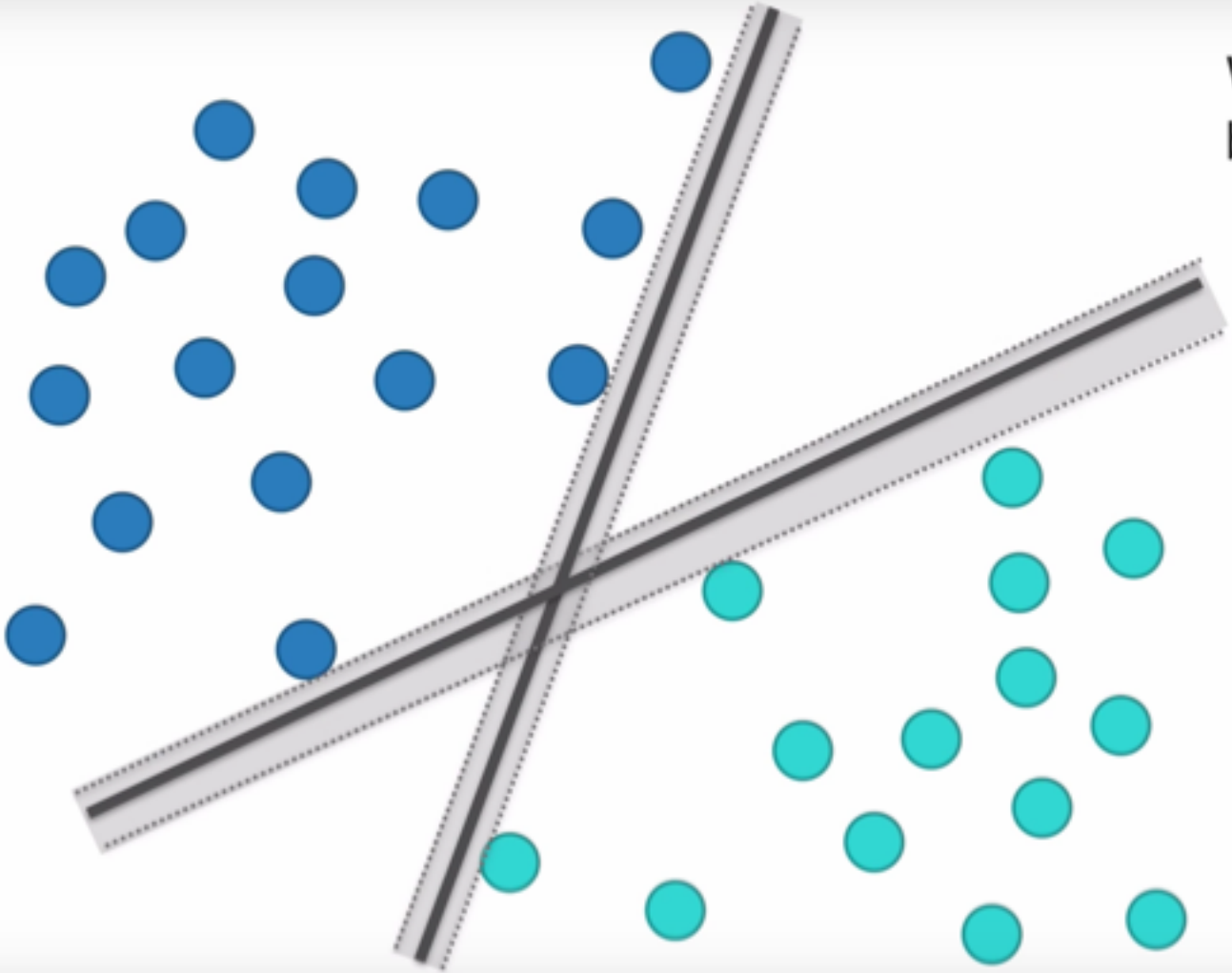


Why is this the best split?

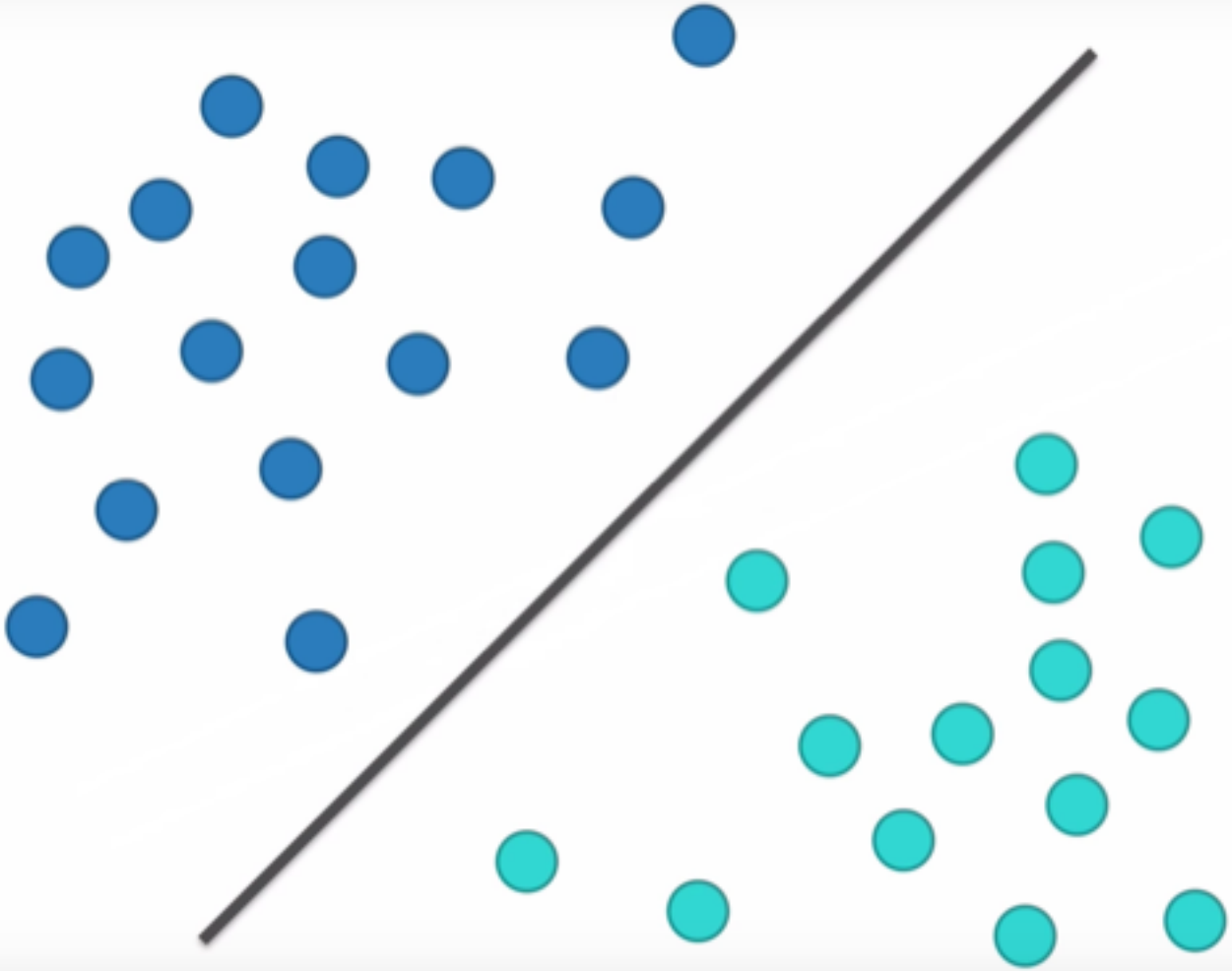


Why is this the best split?

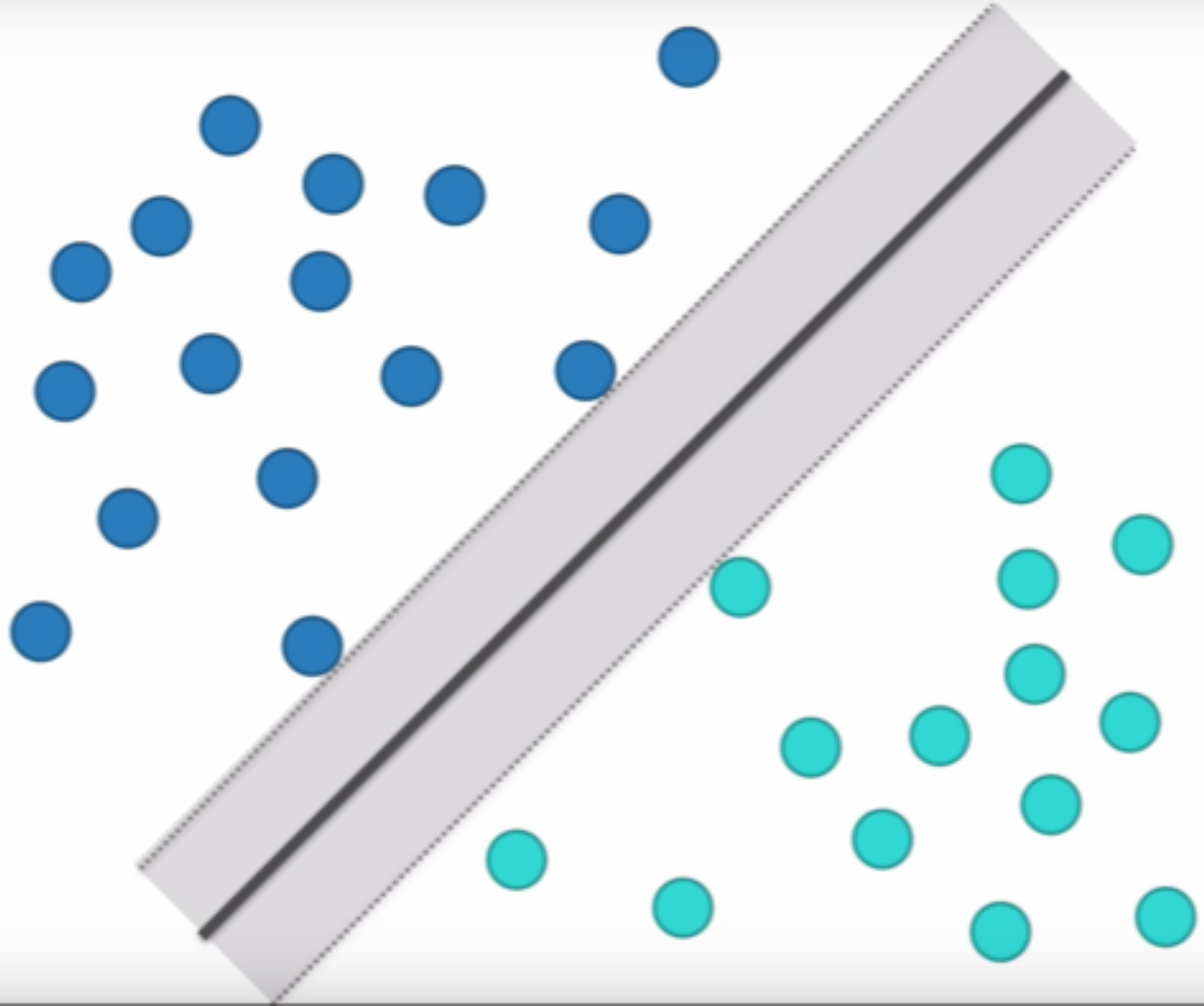
This is the widest road that separates the two groups



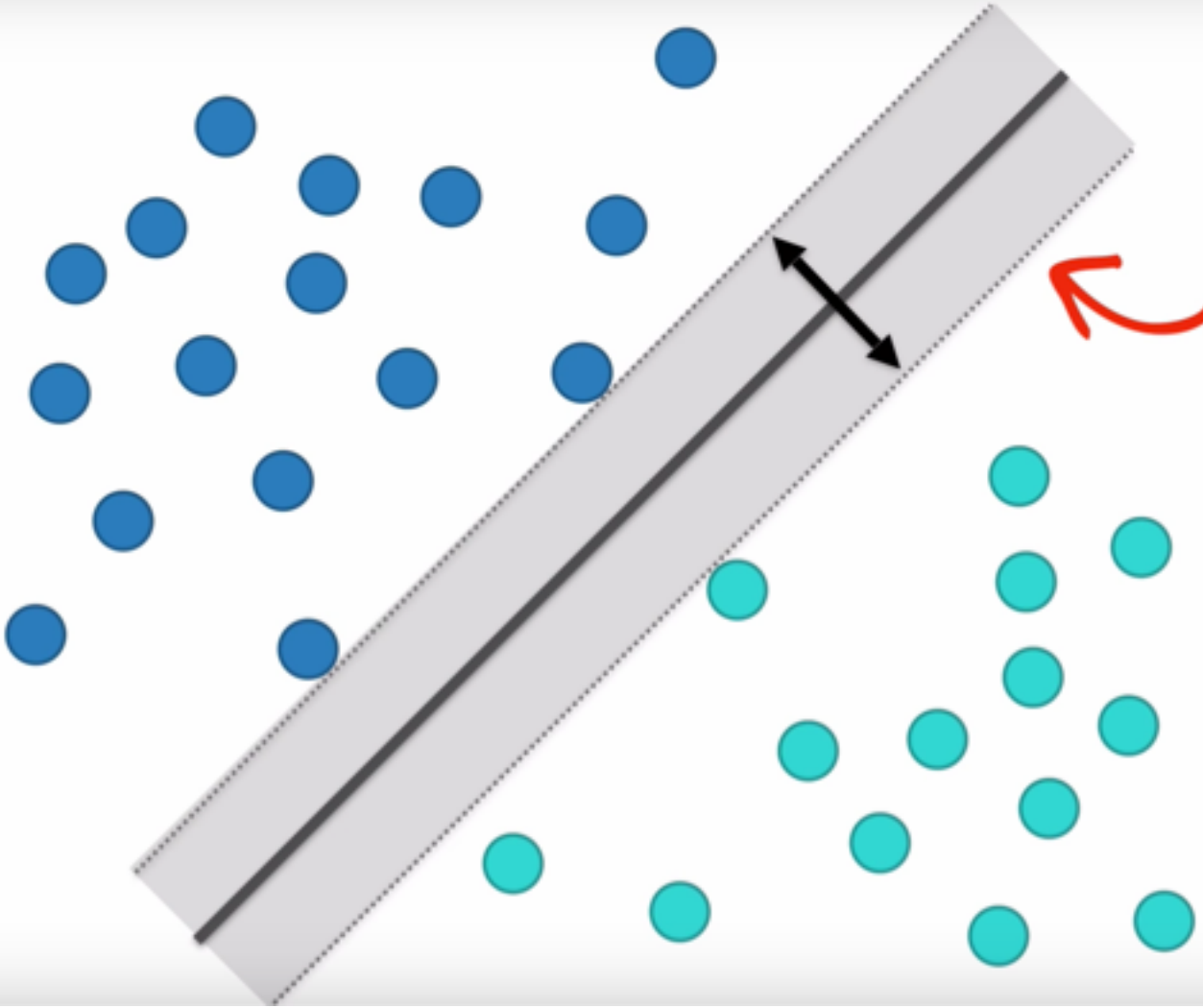
Why is this the best split?



Why is this the best split?

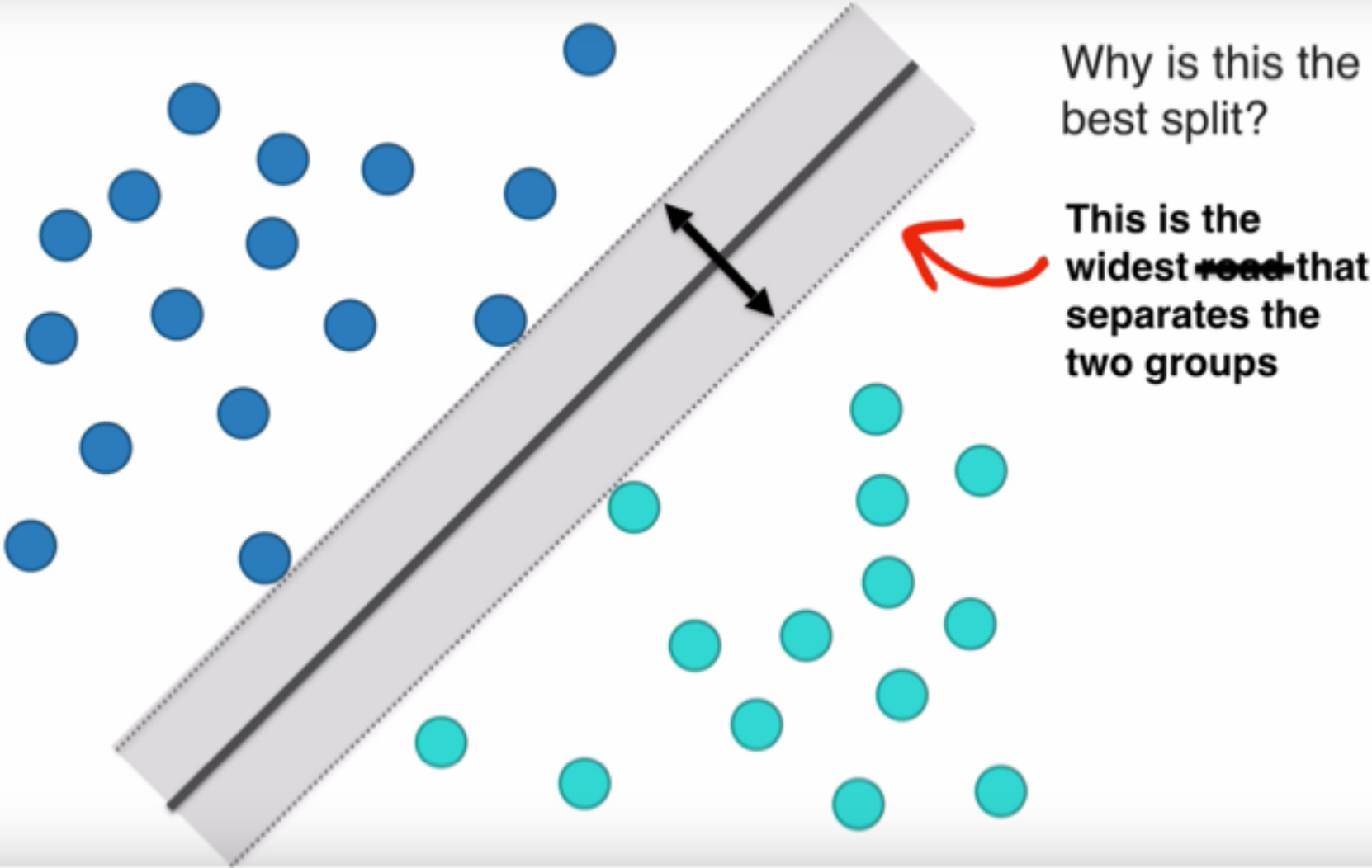


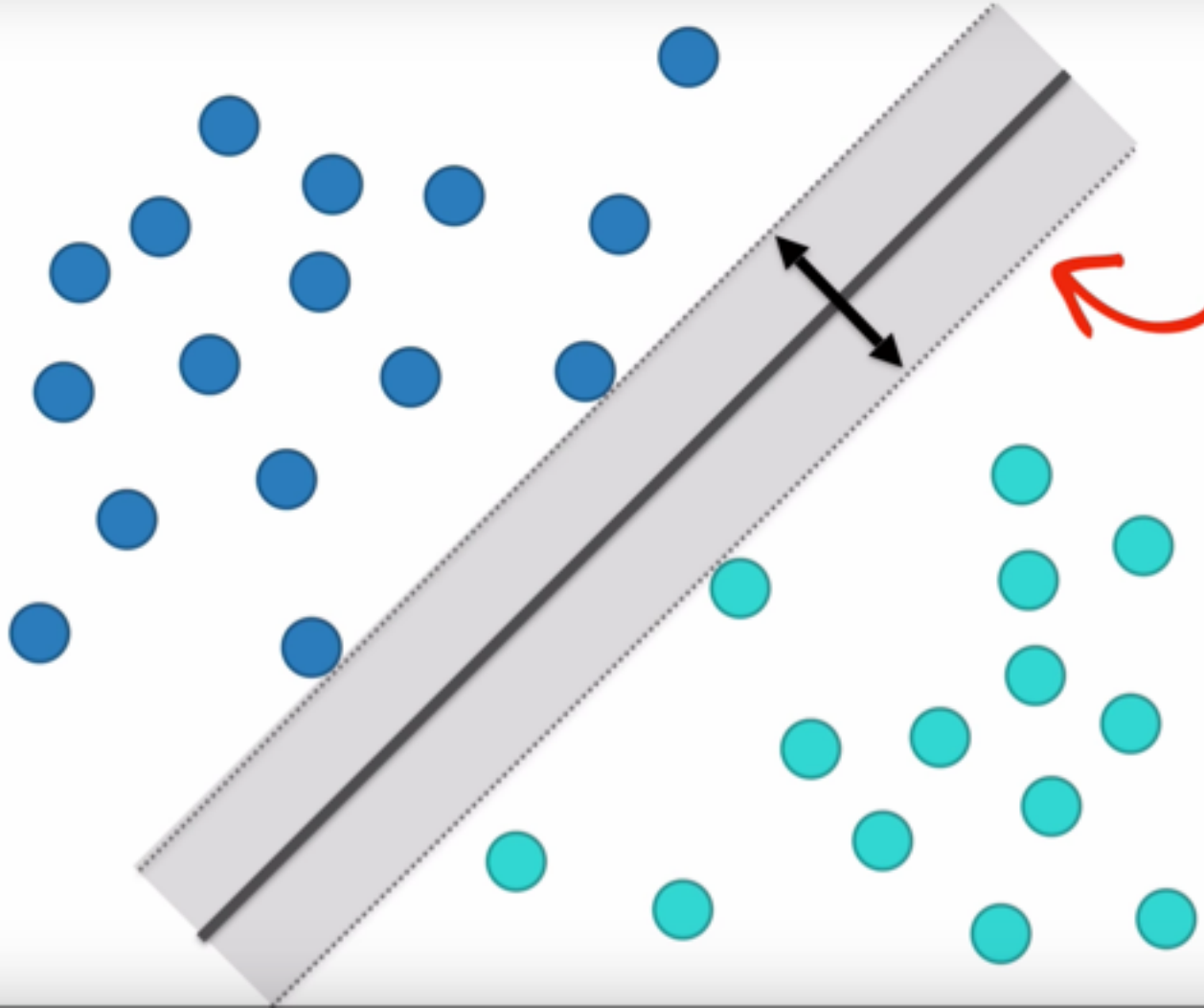
Why is this the best split?



Why is this the best split?

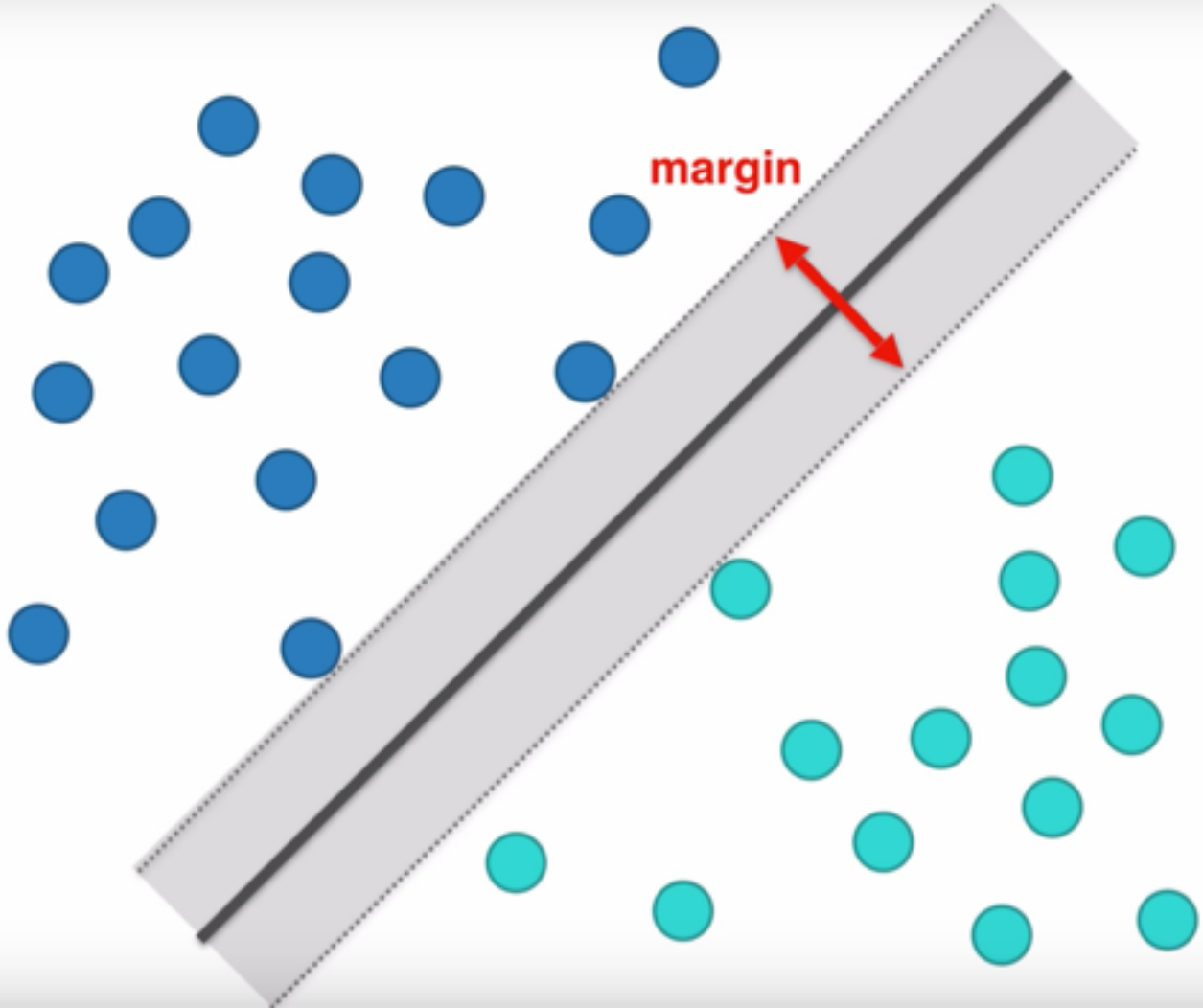
This is the widest road that separates the two groups


Why is this the best split?
This is the widest ~~road~~ that separates the two groups

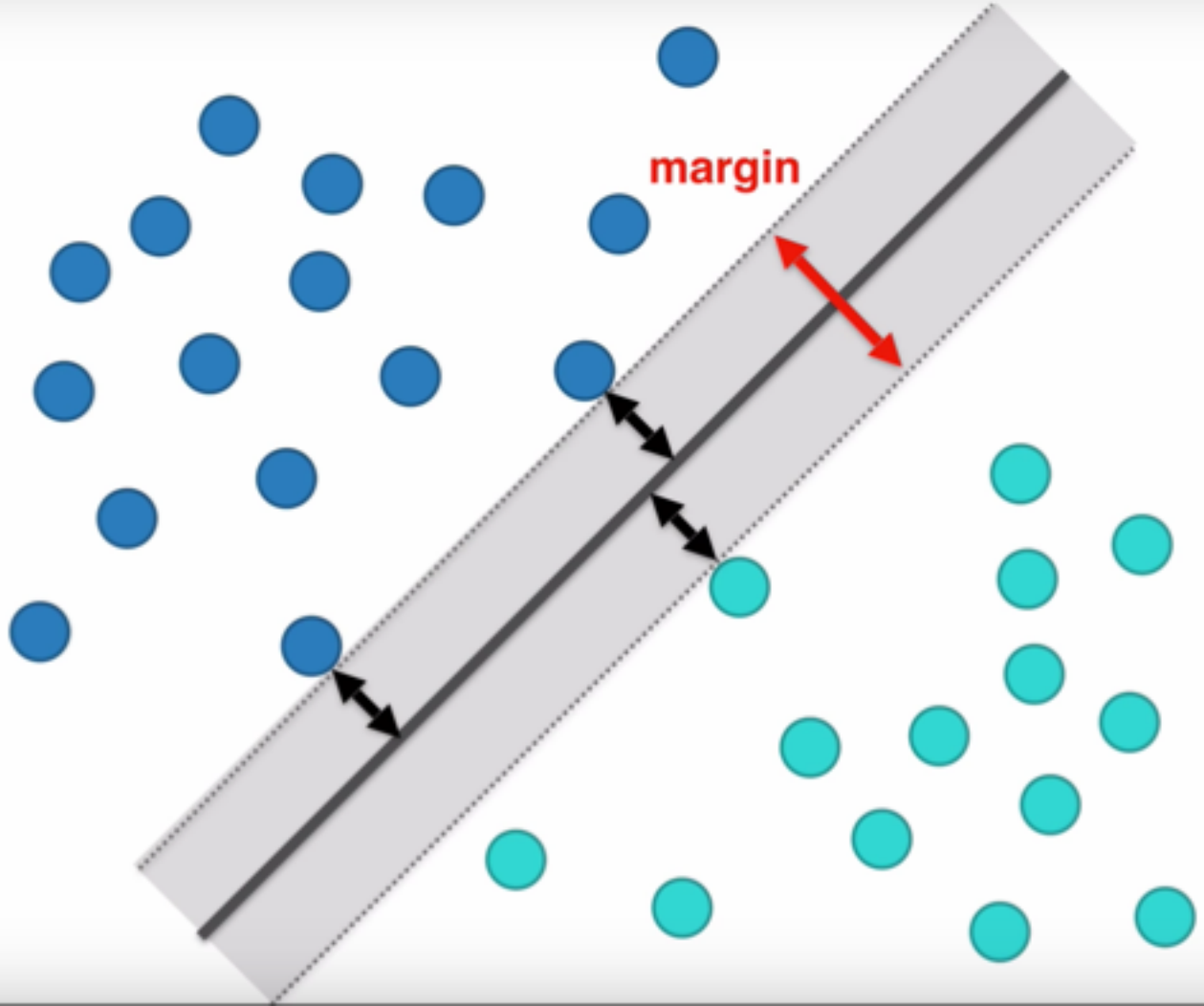


Why is this the best split?

This is the widest margin that separates the two groups

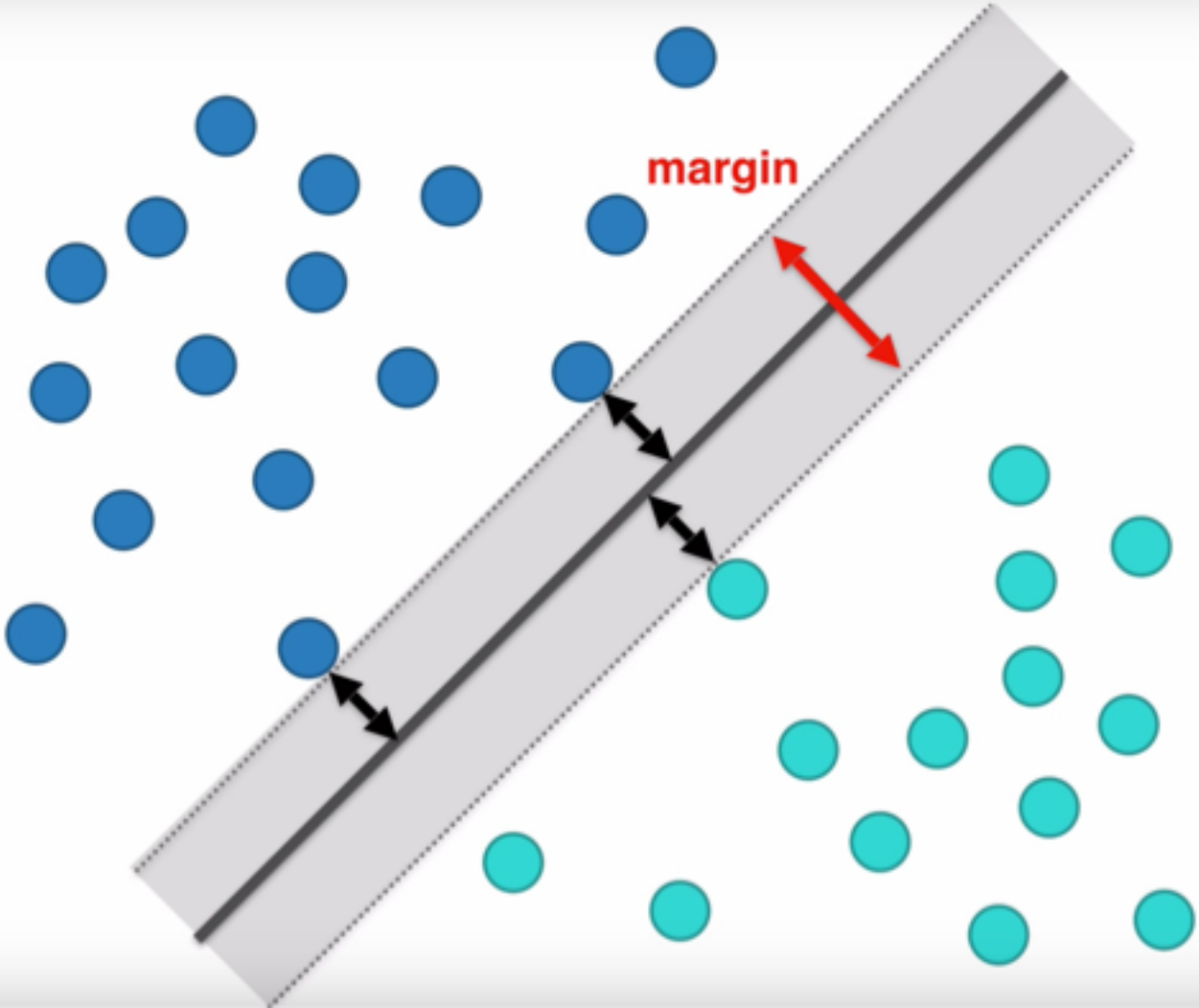


Why is this the best split?



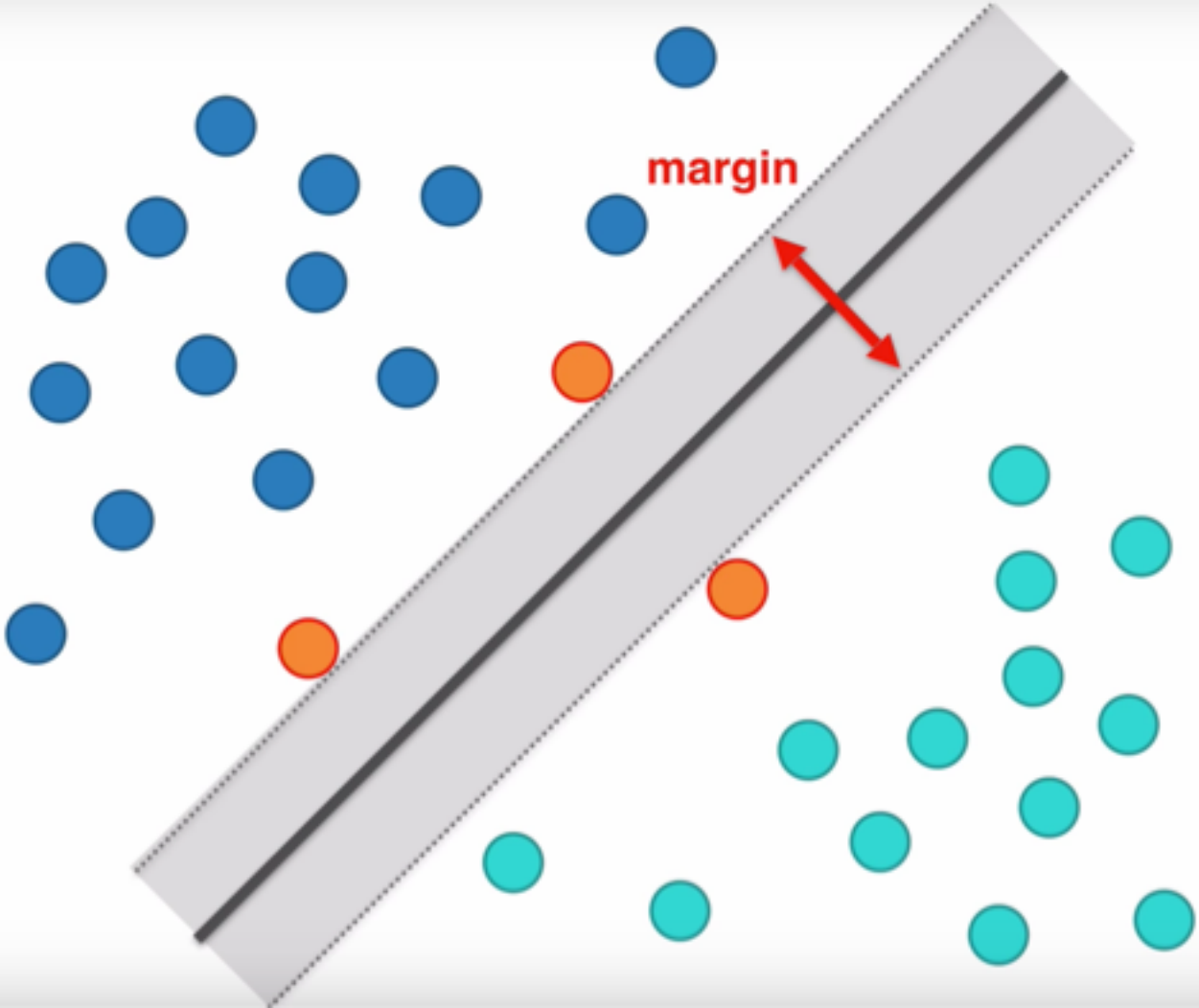
Why is this the best split?

The distance between the points and the line are as far as possible



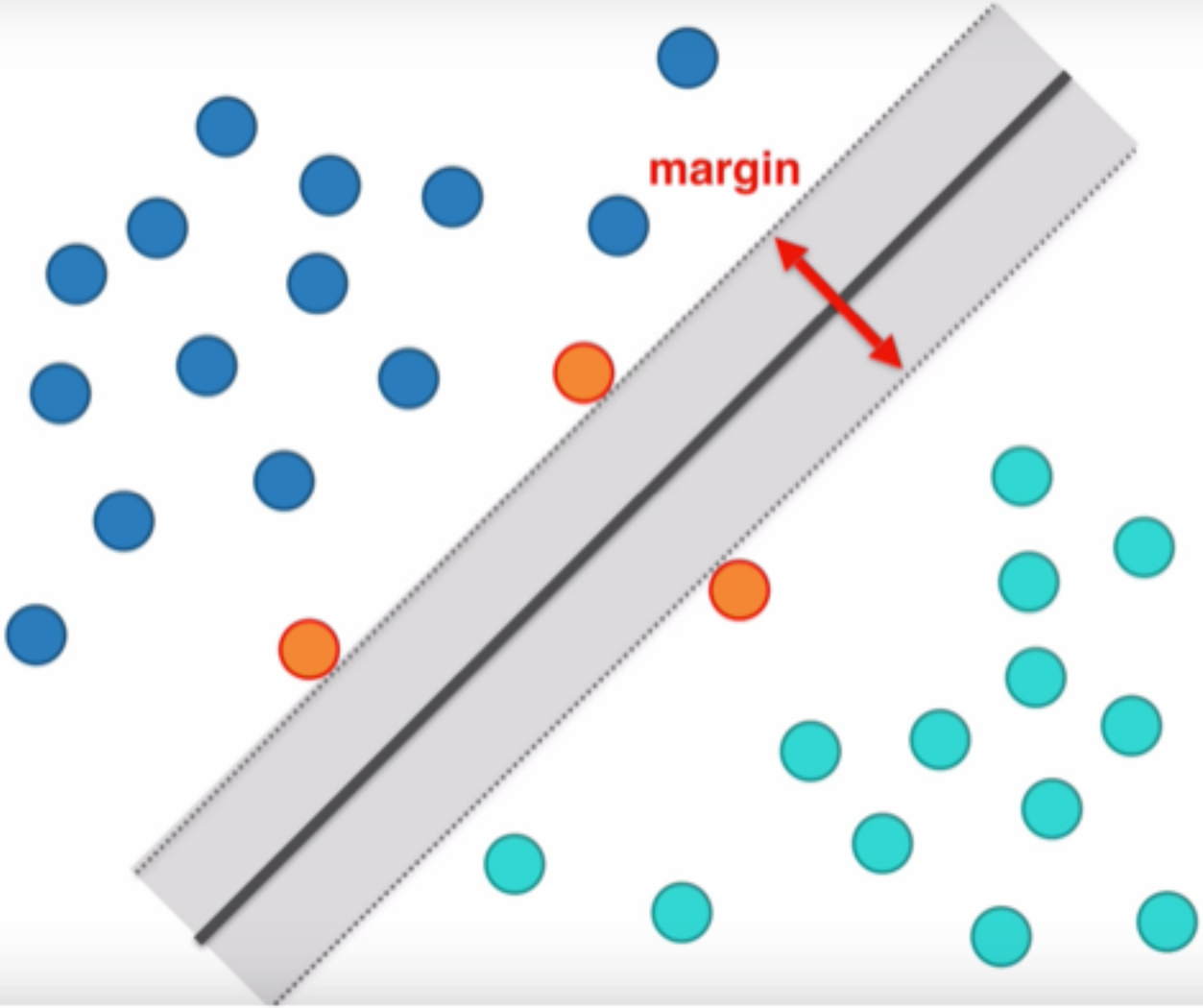
Why is this the best split?

The distance between the ~~points~~ and the line are as far as possible



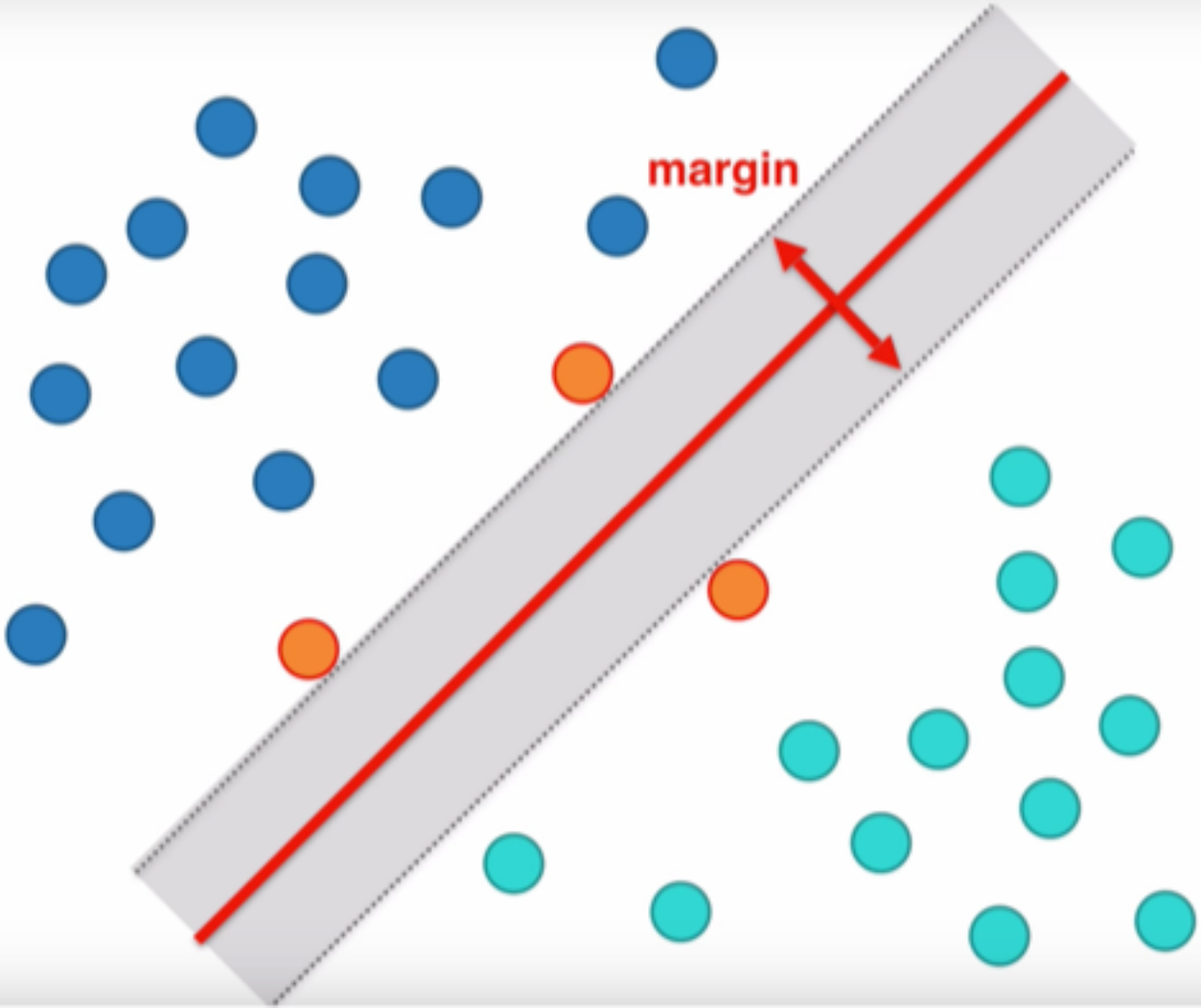
Why is this the best split?

The distance between the support vectors and the line are as far as possible



Why is this the best split?

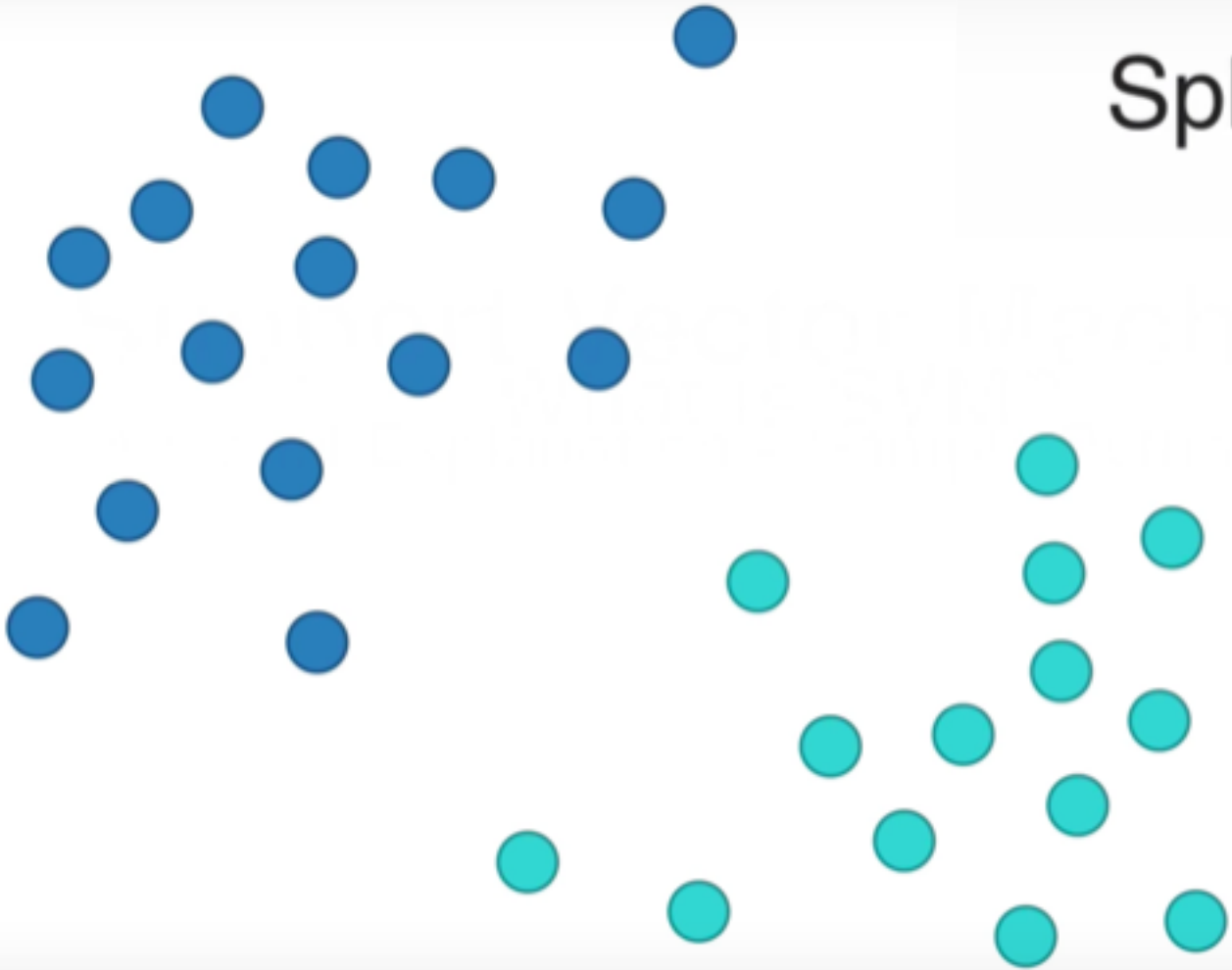
The distance between the support vectors and the ~~line~~ are as far as possible

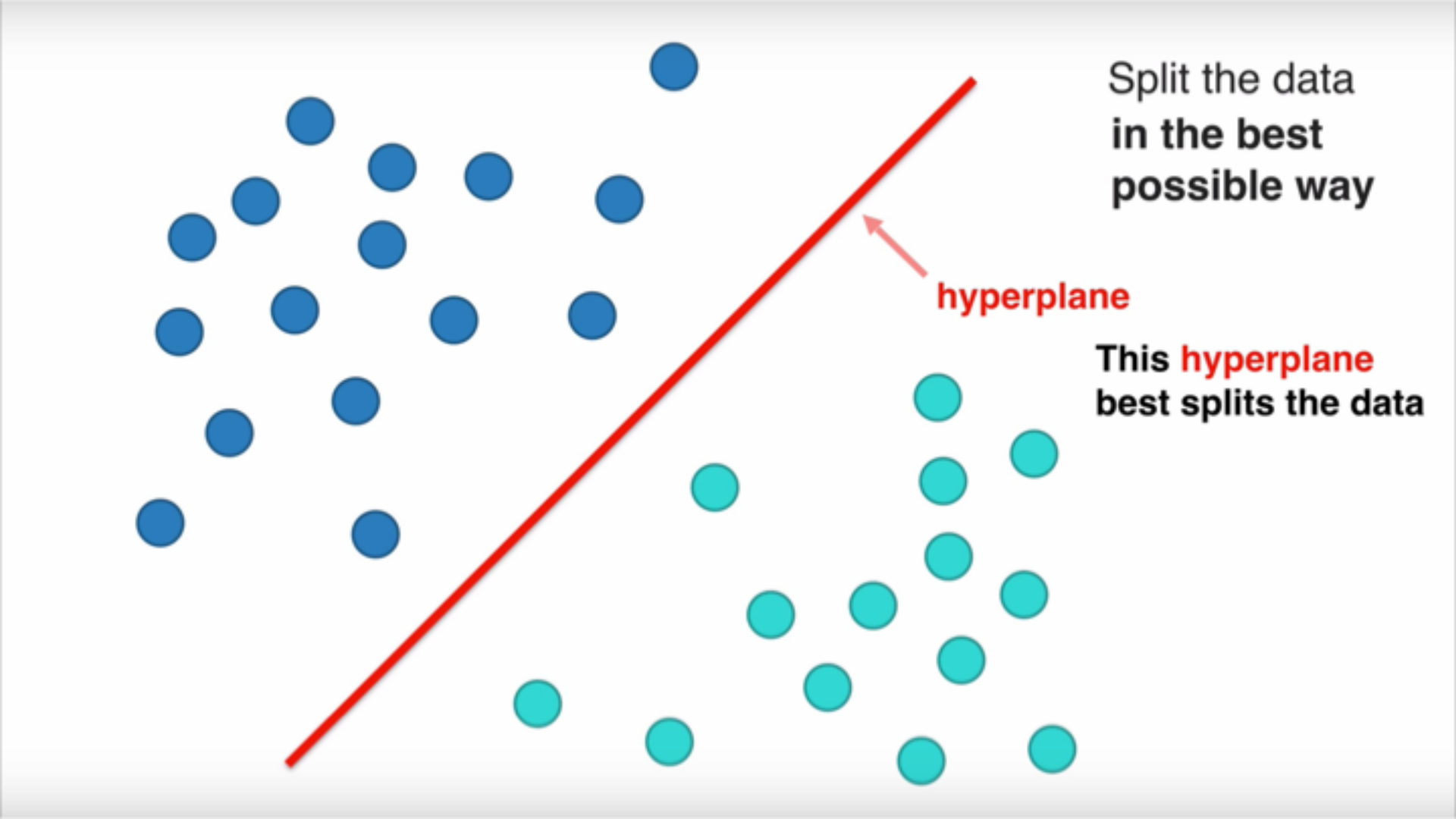


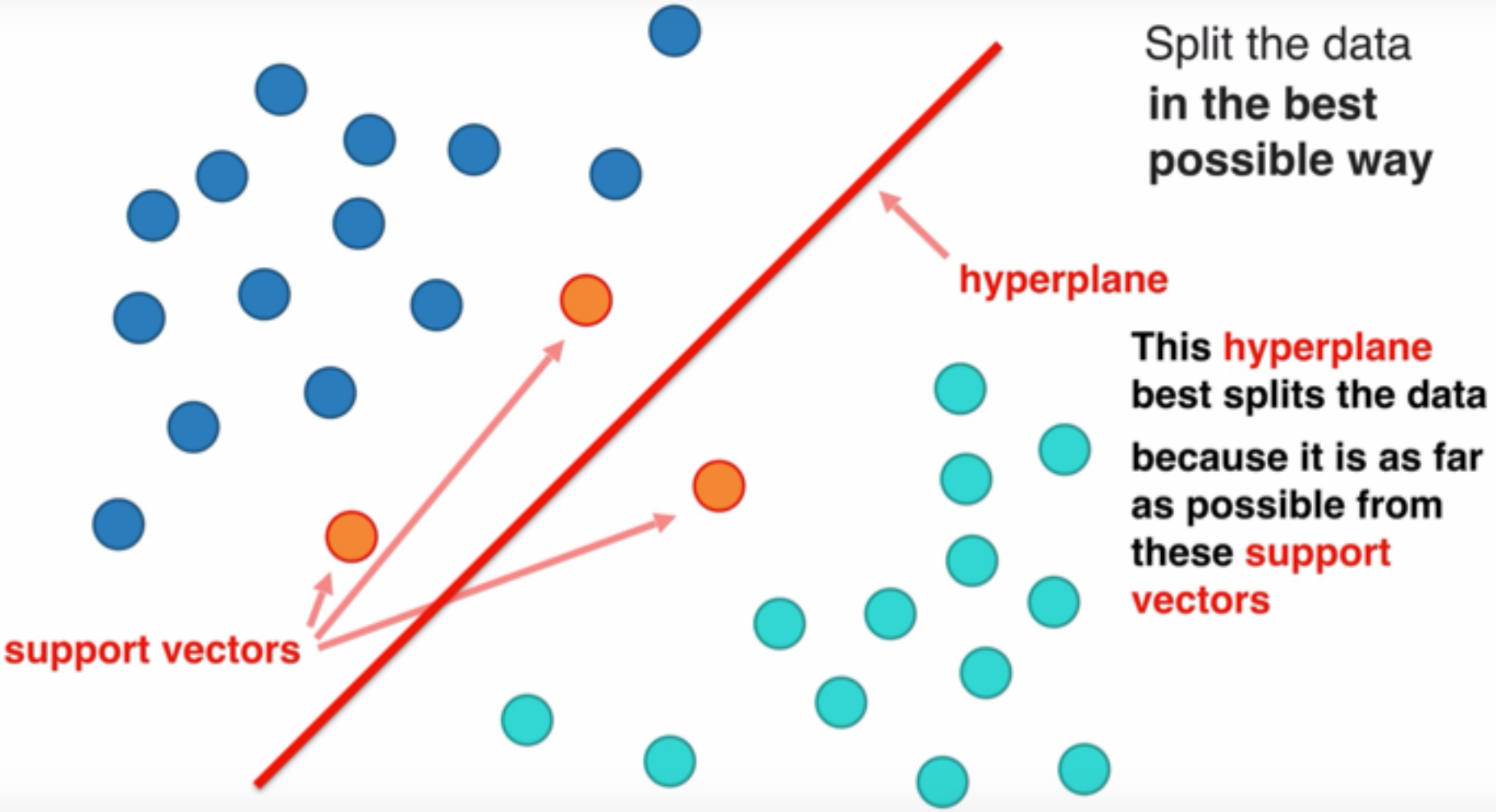
Why is this the best split?

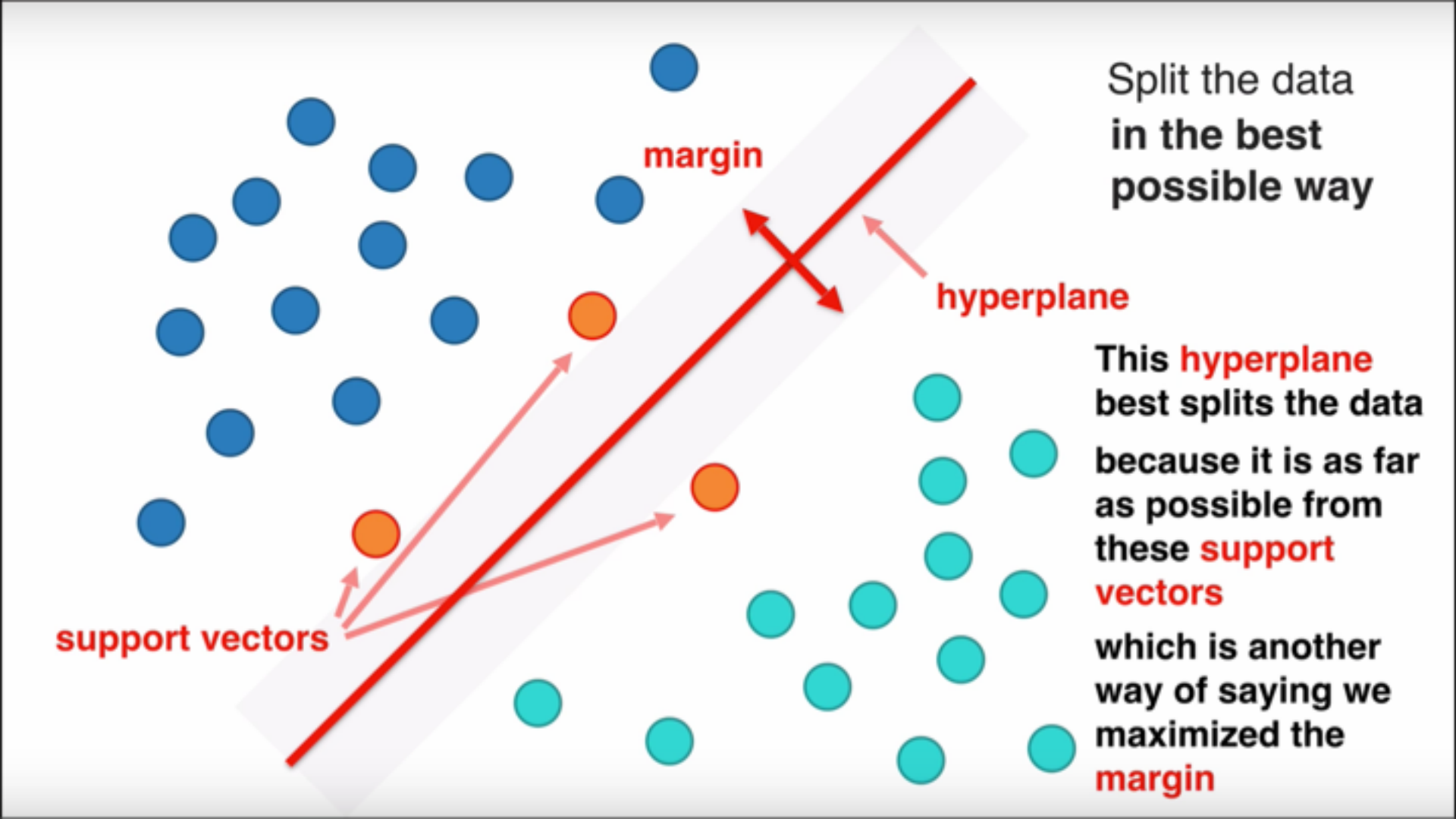
The distance between the support vectors and the hyperplane are as far as possible

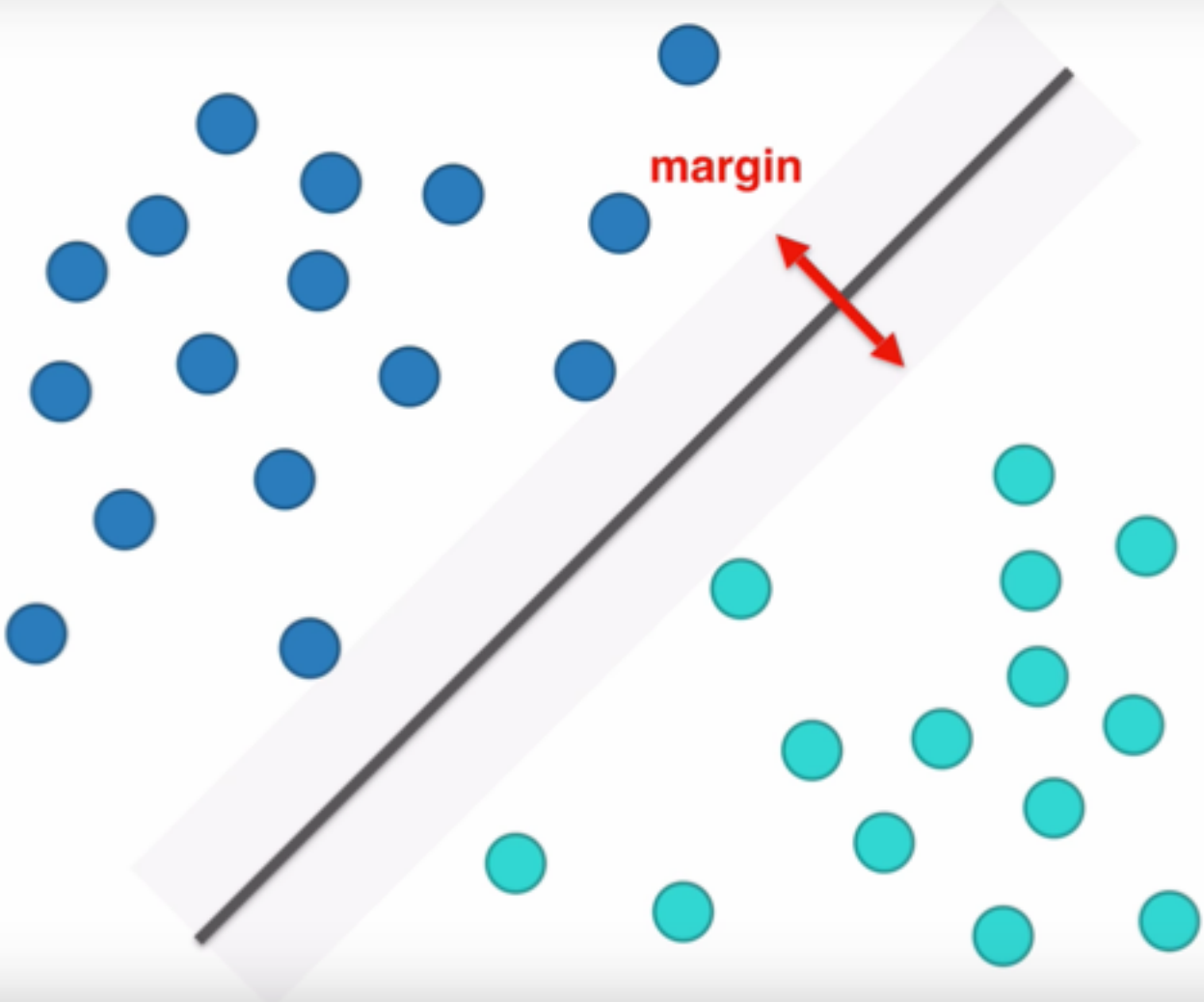
Split the data



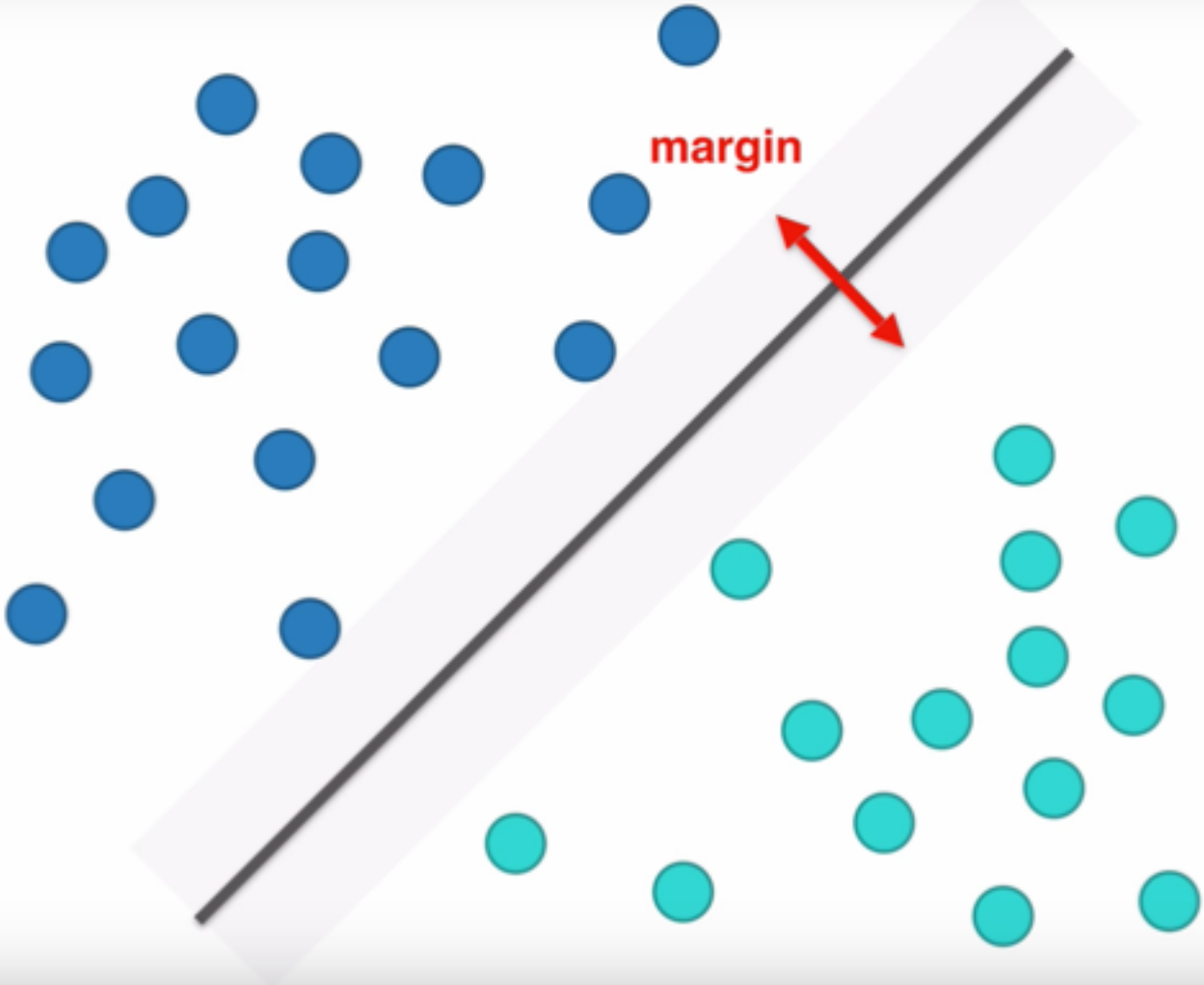

Split the data
in the best
possible way
hyperplane
This hyperplane
best splits the data


Split the data
in the best
possible way
hyperplane
This hyperplane
best splits the data
because it is as far
as possible from
these support
vectors
support vectors


Split the data
in the best
possible way
margin
hyperplane
This hyperplane
best splits the data
because it is as far
as possible from
these support
vectors
which is another
way of saying we
maximized the
margin
support vectors

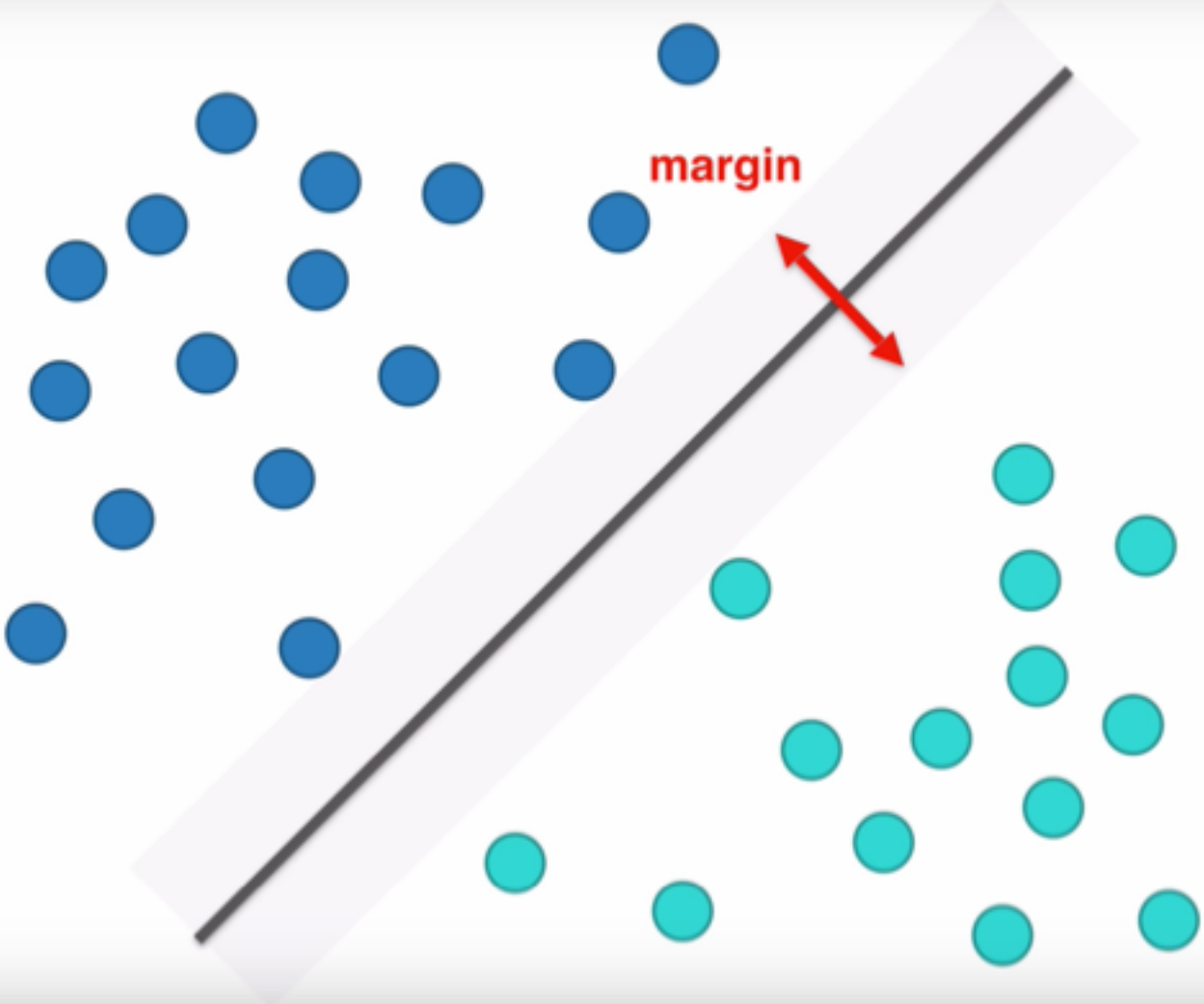


How do you maximize the margin?



How do you maximize the margin?

This is a **constrained optimization** problem



How do you maximize the margin?

This is a **constrained optimization** problem

which can be solved using the **Lagrange Multipliers** technique

So now what?

Let’s apply this to a real world problem.

Cupcakes

versus

Muffins

"a cupcake is just a muffin with frosting"

"a muffin is just a cupcake with random bits of stuff in it"

Cupcakes vs Muffins

The Challenge

Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

The Steps

1. Find the data
2. Apply a data science model
3. Review the results

Cupcakes vs Muffins

The Challenge

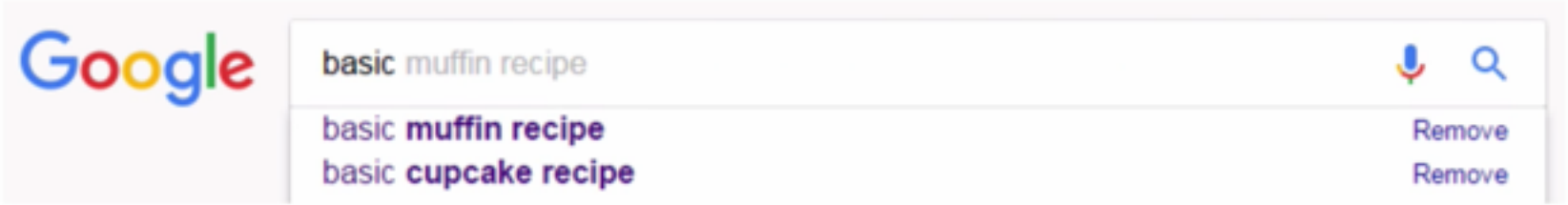
Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

The Steps

1. Find the data
2. Apply a data science model
3. Review the results

1. Find the data

1. Find the data



Recorded the top 10 muffin and top 10 cupcake recipes

Recipe	Flour	Sugar	Butter	Milk	Egg	Baking Powder	Salt	Vanilla
Muffin1	2 cups	1/2 cup	1/4 cup	1 cup	2 eggs	1 tbsp	1/2 tsp	1 tsp
Cupcake1	2 cups	3/4 cup	1/2 cup	1 cup	2 eggs	2 tsp	1/2 tsp	1 tsp
...	...	...	...	...	...	...	...	...

1. Find the data

Problem

Each recipe yields different amounts of batter

1. Find the data

Problem

Each recipe yields different amounts of batter

Solution

Amount-based

Recipe	Flour	Sugar	Other
Muffin1	2 cups	1/2 cup	…
Cupcake1	2 cups	3/4 cup	…

Percent-based

Recipe	Flour	Sugar	Other	Total Volume
Muffin1	47%	24%	…	100%
Cupcake1	42%	21%	…	100%

1. Find the data

Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
Muffin	55	28	3	7	5	2	0	0
Muffin	47	24	12	6	9	1	0	0
Muffin	47	23	18	6	4	1	0	0
Muffin	50	25	12	6	5	2	1	0
Muffin	55	27	3	7	5	2	1	0
Muffin	54	27	7	5	5	2	0	0
Muffin	47	26	10	10	4	1	0	0
Muffin	50	17	17	8	6	1	0	0
Muffin	50	17	17	11	4	1	0	0
Cupcake	39	0	26	19	14	1	1	0
Cupcake	34	17	20	20	5	2	1	0
Cupcake	39	13	17	19	10	1	1	0
Cupcake	38	15	23	15	8	0	1	0
Cupcake	42	18	25	9	5	1	0	0
Cupcake	36	14	21	14	11	2	1	0
Cupcake	38	15	31	8	6	1	1	0
Cupcake	36	16	24	12	9	1	1	0
Cupcake	34	17	23	11	13	0	1	0

Cupcakes vs Muffins

The Challenge

Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

The Steps

1. Find the data ✓
2. Apply a data science model
3. Review the results

Cupcakes vs Muffins

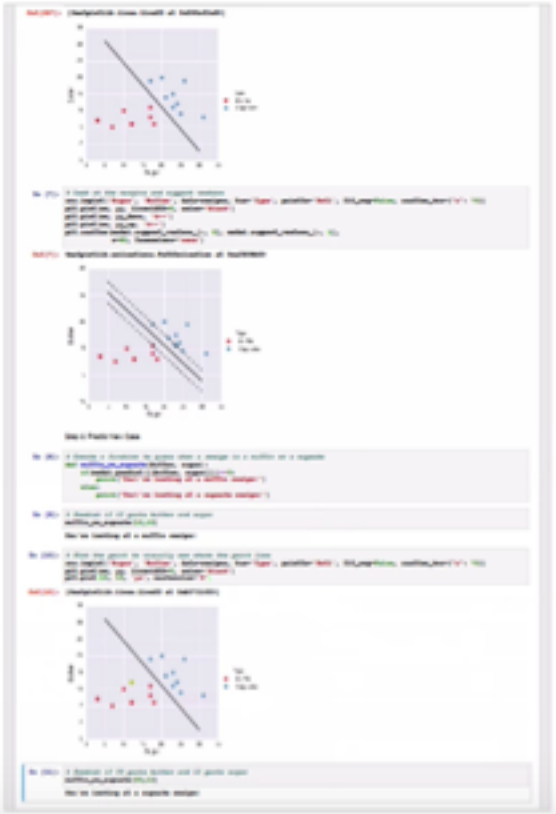
The Challenge

Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

The Steps

1. Find the data ✓
2. Apply a data science model
3. Review the results

Python Script



a. Import Libraries

b. Import Data

c. Prepare the Data

d. Fit the Model

e. Visualize Results

f. Predict New Case

a. Import Libraries

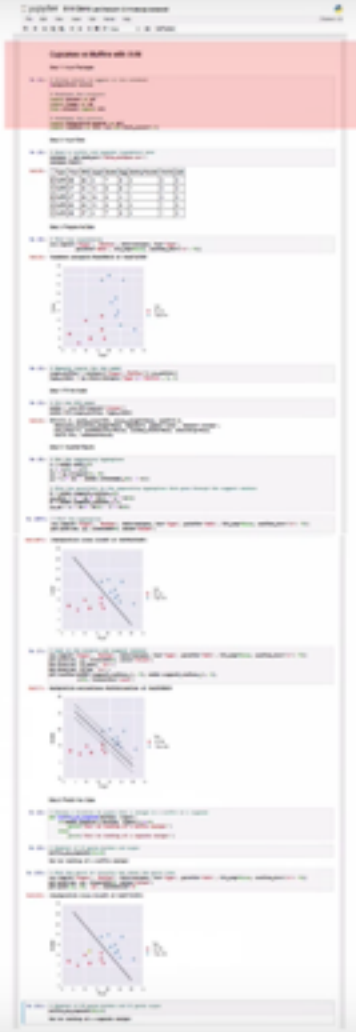
Cupcakes vs Muffins with SVM

Step 1: Import Libraries

In [13]:
```
# Allows charts to appear in the notebook
%matplotlib inline

# Libraries for analysis
import pandas as pd
import numpy as np
from sklearn import svm

# Libraries for visuals
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(font_scale=1.2)
```



b. Import Data

Step 2: Import Data

In [2]:
```
# Read in muffin and cupcake ingredient data
recipes = pd.read_csv('data_recipes.csv')
recipes.head()
```

Out[2]:

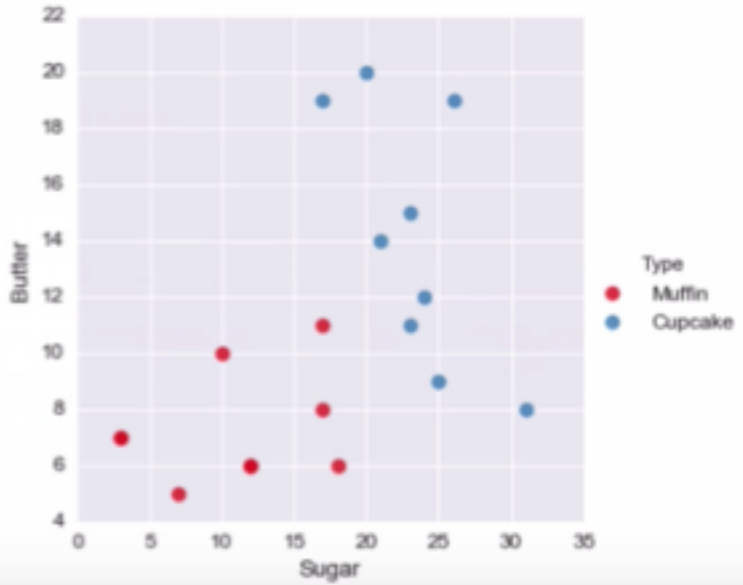
	Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
0	Muffin	55	28	3	7	5	2	0	0
1	Muffin	47	24	12	6	9	1	0	0
2	Muffin	47	23	18	6	4	1	0	0
3	Muffin	50	25	12	6	5	2	1	0
4	Muffin	55	27	3	7	5	2	1	0

c. Prepare the Data

Step 3: Prepare the Data

In [3]:
```
# Plot two ingredients
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',
           palette='Set1', fit_reg=False, scatter_kws={"s": 70})
```

Out[3]: <seaborn.axisgrid.FacetGrid at 0xad7af28>



d. Fit the Model

Step 4: Fit the Model

In [4]:
```
# Specify inputs for the model
sugar_butter = recipes[['Sugar','Butter']].as_matrix()
type_label = np.where(recipes['Type']=='Muffin', 0, 1)
```

In [5]:
```
# Fit the SVM model
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

Out[5]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

d. Fit the Model

Step 4: Fit the Model

In [4]:
```
# Specify inputs for the model
sugar_butter = recipes[['Sugar','Butter']].as_matrix()
type_label = np.where(recipes['Type']=='Muffin', 0, 1)
```

In [5]:
```
# Fit the SVM model
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

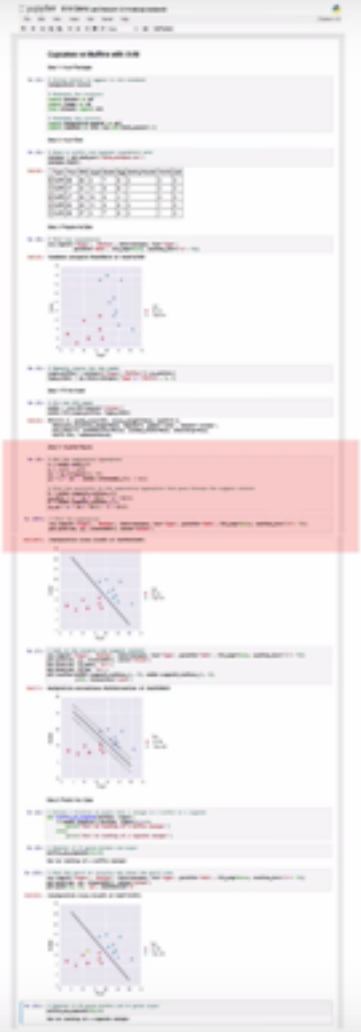
Out[5]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

e. Visualize Results

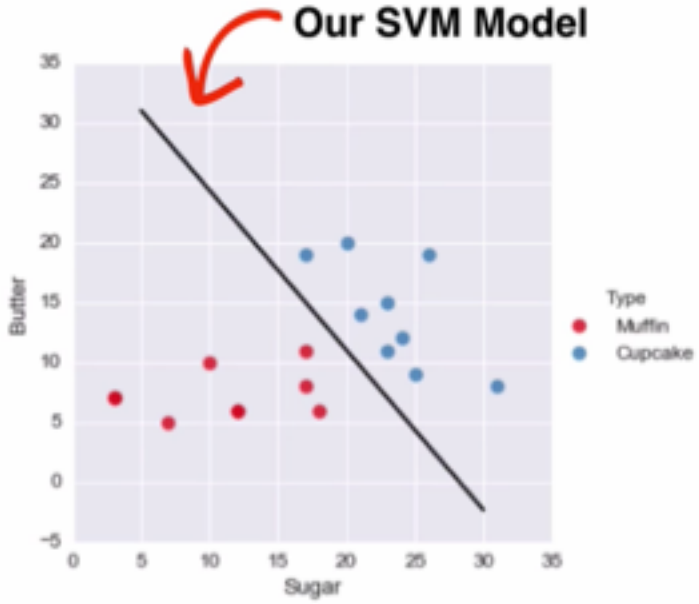
Step 5: Visualize Results

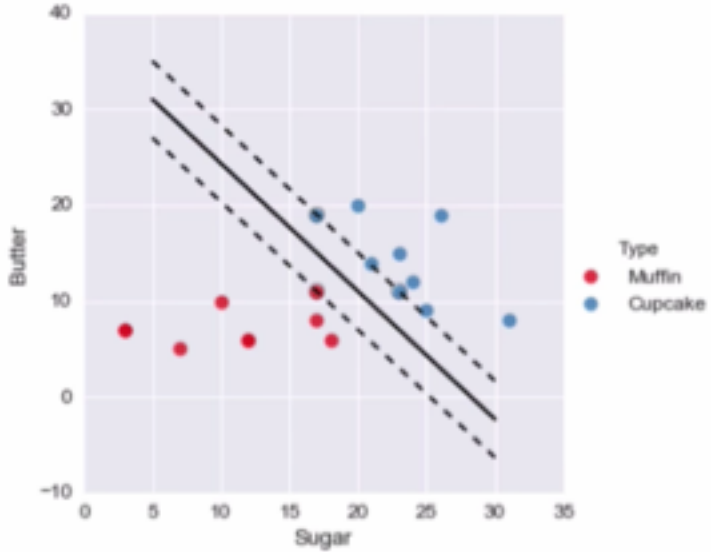
In [6]:
```
# Get the separating hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(5, 30)
yy = a * xx - (model.intercept_[0]) / w[1]

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



e. Visualize Results





Cupcakes vs Muffins

The Challenge

Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

The Steps

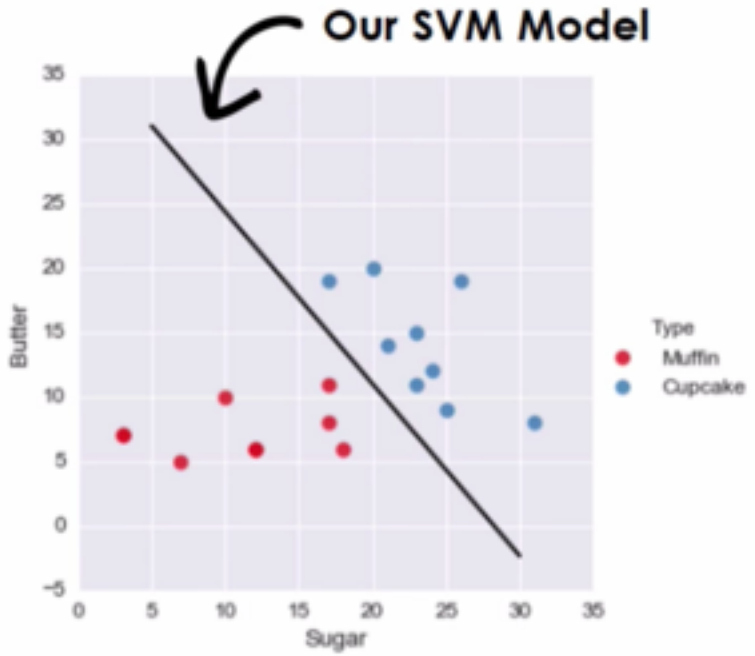
1. Find the data ✓
2. Apply a data science model ✓
3. Review the results

3. Review the Results

The Challenge

Classify recipes as cupcakes or muffins. ✓

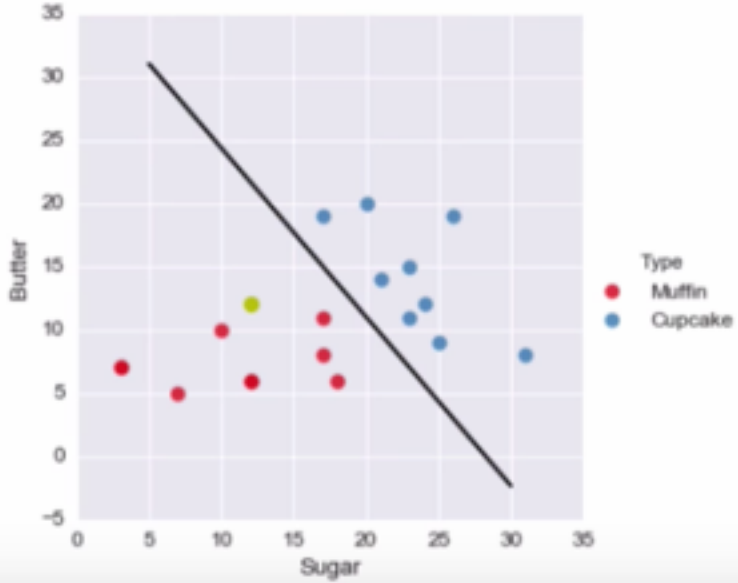
When given a new recipe, determine if it's a cupcake or a muffin.

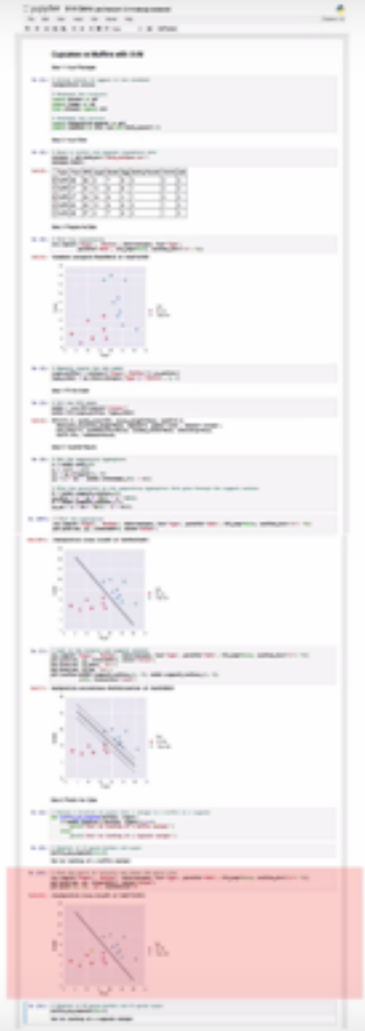


f. Predict New Case

In [10]:
```
# Plot the point to visually see where the point lies
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',
           palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(12, 12, 'yo', markersize='9')
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0xb075160>]
```





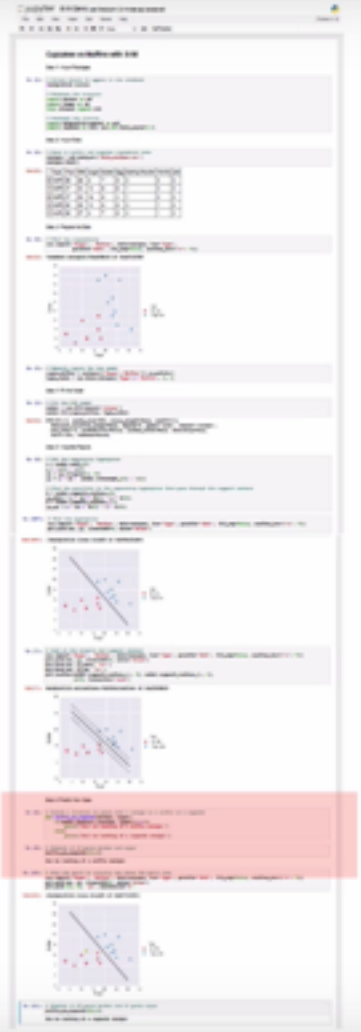
f. Predict New Case

Step 6: Predict New Case

In [8]:
```
# Create a function to guess when a recipe is a muffin
# or a cupcake using the SVM model we created
def muffin_or_cupcake(butter, sugar):
    if(model.predict([[butter, sugar]]))==0:
        print('You\'re looking at a muffin recipe!')
    else:
        print('You\'re looking at a cupcake recipe!')
```

In [9]:
```
# Predict if 12 parts butter and sugar
muffin_or_cupcake(12,12)
```

```
You're looking at a muffin recipe!
```



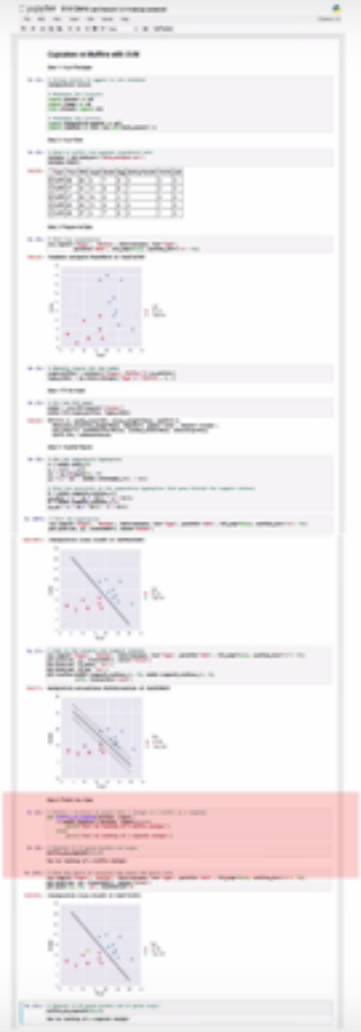
f. Predict New Case

Step 6: Predict New Case

In [8]:
```
# Create a function to guess when a recipe is a muffin
# or a cupcake using the SVM model we created
def muffin_or_cupcake(butter, sugar):
    if(model.predict([[butter, sugar]]))==0:
        print('You\'re looking at a muffin recipe!')
    else:
        print('You\'re looking at a cupcake recipe!')
```

In [9]:
```
# Predict if 12 parts butter and sugar
muffin_or_cupcake(12,12)
```

```
You're looking at a muffin recipe!
```



Cupcakes

versus

Muffins

Cupcakes and muffins can be classified using Support Vector Machines!

Basic case ✓

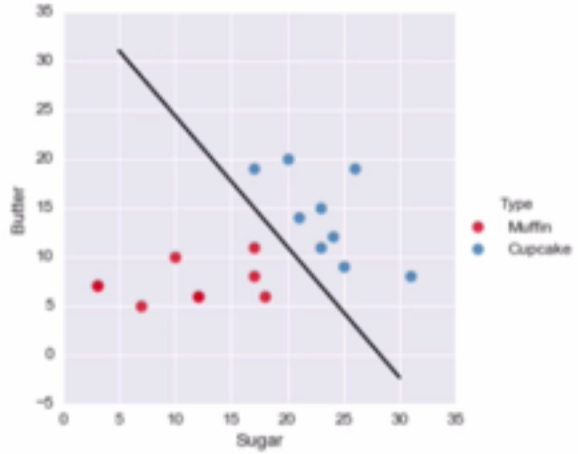
Up next: How to make SVM even more powerful

Agenda

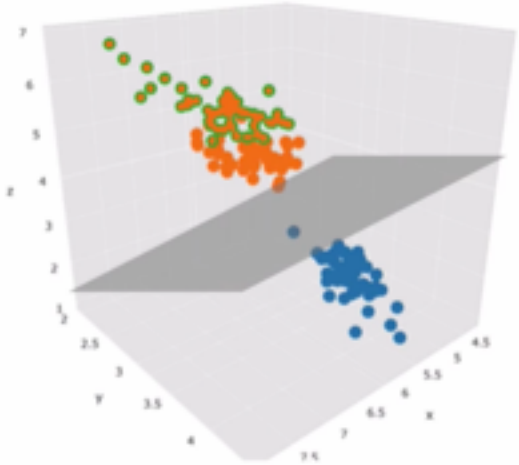
1. SVM Basics
   - Visual introduction
   - Example in Python
2. Additional Complexities
   - Higher dimensions
   - Multiple classes
   - C parameter
   - Kernel trick
3. Closing Remarks
   - Pros and cons
   - Other techniques

Higher Dimensions: Visual

2D: Separate with Line



3D: Separate with Plane



4D+: Separate with Hyperplane

Hard to Visualize

Higher Dimensions: Code

- Hard to visualize, but easy to code

```
# Fit SVM model for sugar & butter
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

```
[ 3,  7],
[12,  6],
[18,  6],
[12,  6],
[ 3,  7],...
```

Higher Dimensions: Code

- Hard to visualize, but easy to code

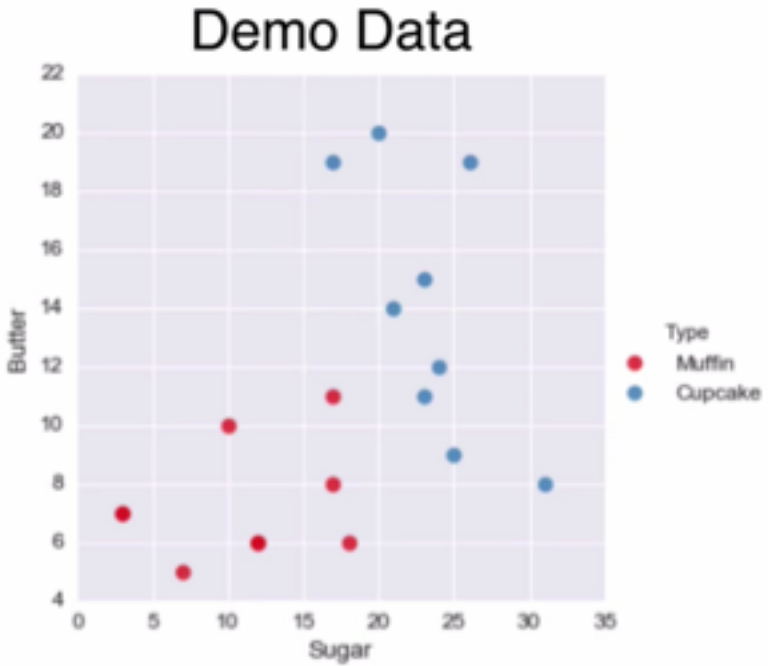
```
# Fit SVM model for sugar & butter
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

```
[ 3,  7],
[12,  6],
[18,  6],
[12,  6],
[ 3,  7], ...
```

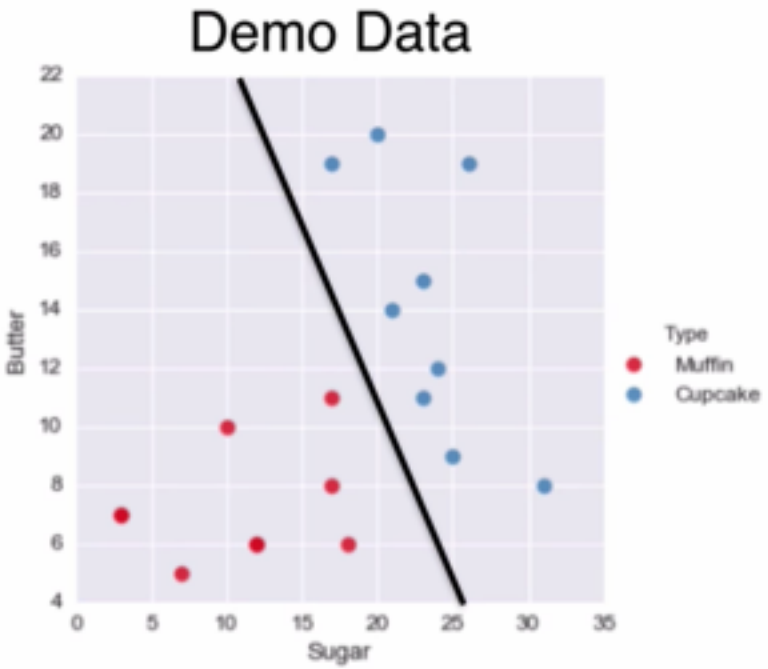
all_ingredients

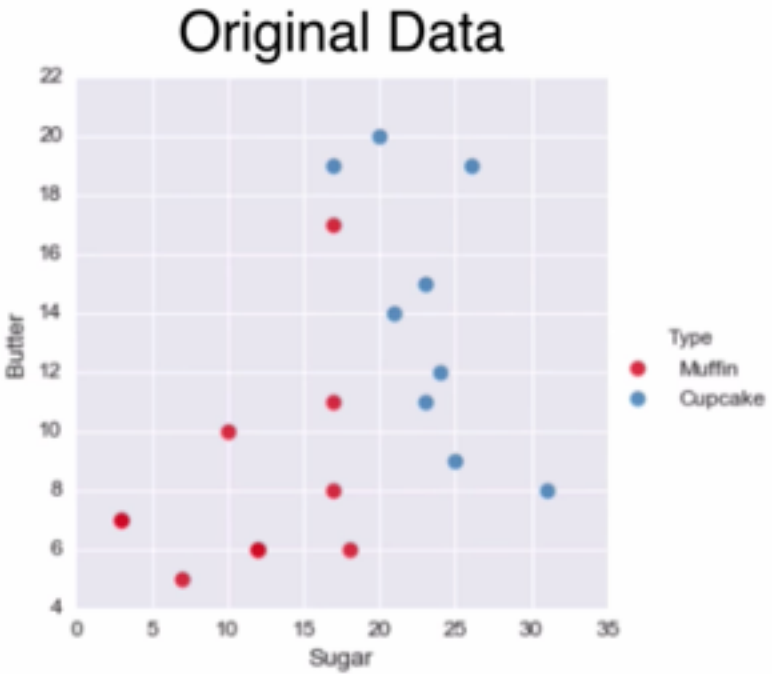
```
[55, 28,  3,  7,  5,  2,  0,  0],
[47, 24, 12,  6,  9,  1,  0,  0],
[47, 23, 18,  6,  4,  1,  0,  0],
[50, 25, 12,  6,  5,  2,  1,  0],
[55, 27,  3,  7,  5,  2,  1,  0], ...
```

C Parameter: Visual

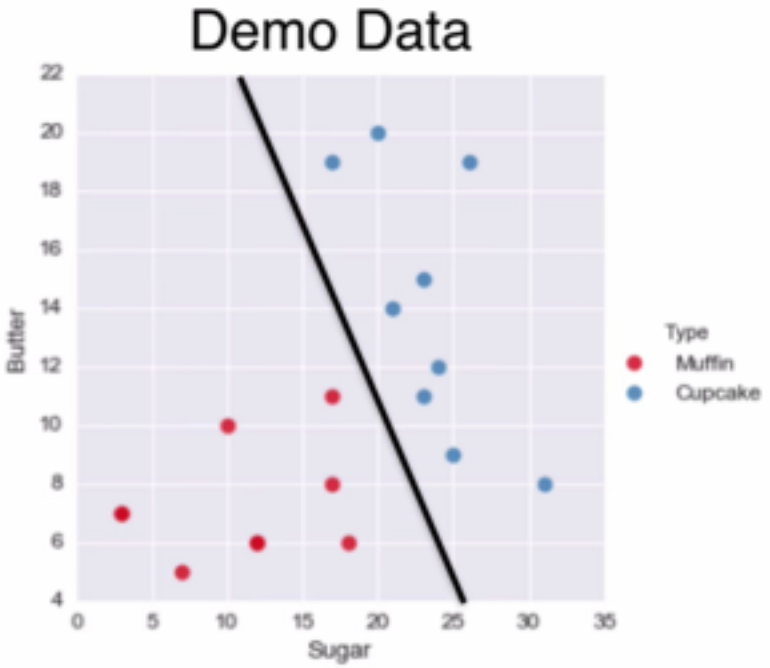


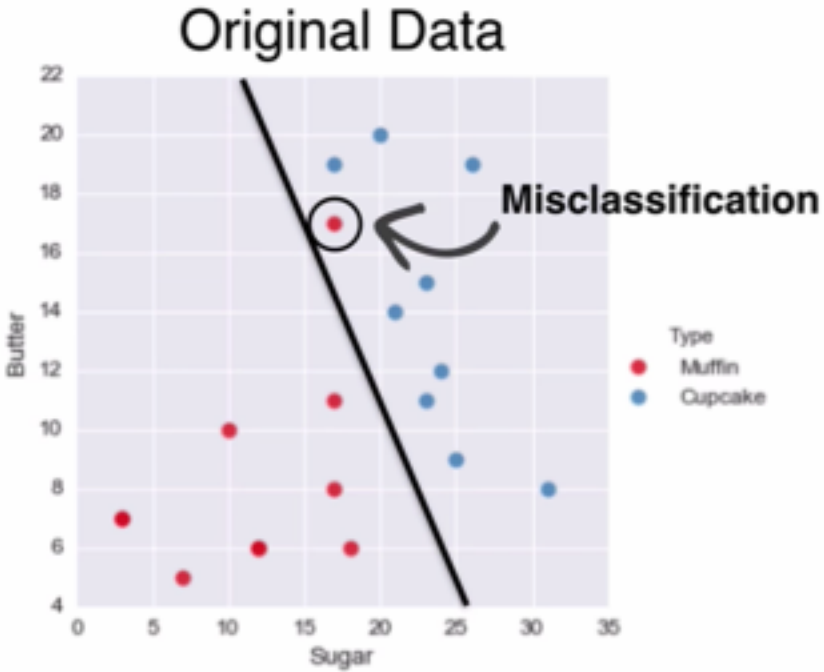
C Parameter: Visual





C Parameter: Visual





C Parameter: Code

The C parameter allows you to decide how much you want to penalize misclassified points

```
In [5]: # Fit the SVM model
        model = svm.SVC(kernel='linear')
        model.fit(sugar_butter, type_label)

Out[5]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

Default Value

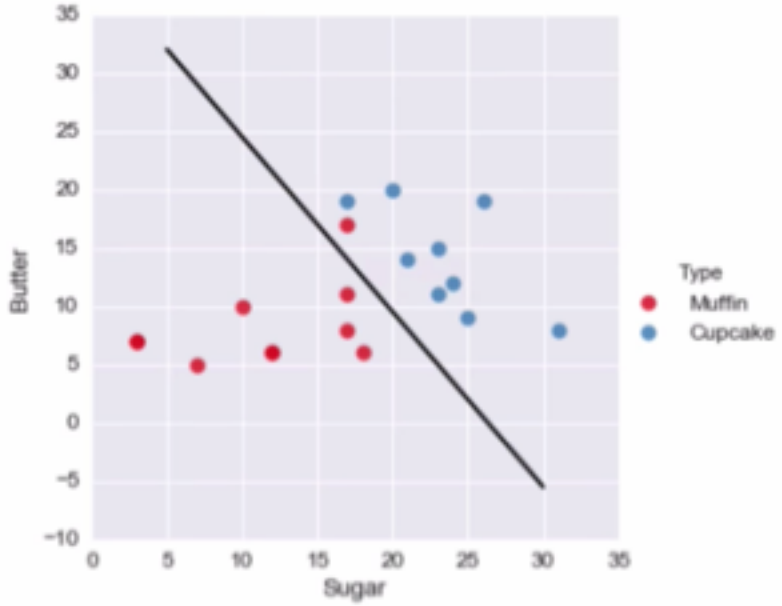
C Parameter: Comparison

```
# Fit the SVM model with a LOW C
model = svm.SVC(kernel='linear', C=2**-5)
model.fit(sugar_butter, type_label)
```

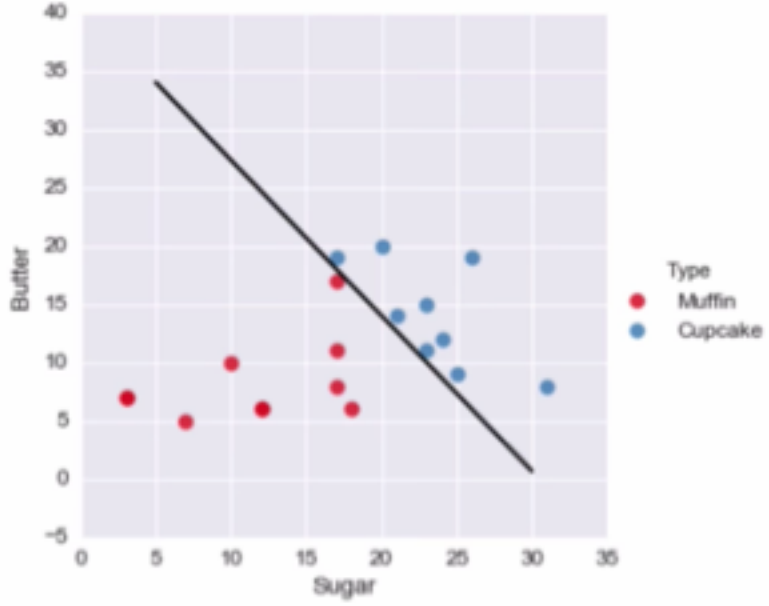
```
# Fit the SVM model with a HIGH C
model = svm.SVC(kernel='linear', C=2**5)
model.fit(sugar_butter, type_label)
```

C Parameter: Comparison

```
# Fit the SVM model with a LOW C
model = svm.SVC(kernel='linear', C=2**-5)
model.fit(sugar_butter, type_label)
```



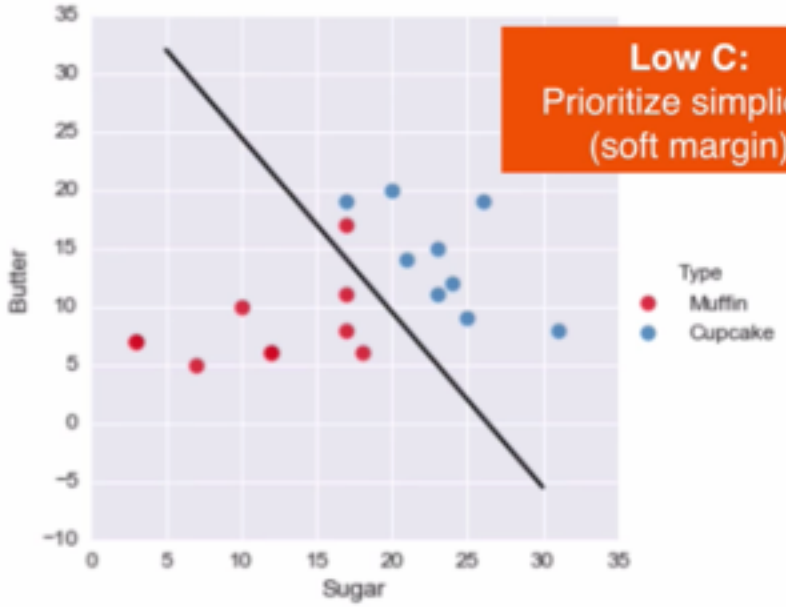

```
# Fit the SVM model with a HIGH C
model = svm.SVC(kernel='linear', C=2**5)
model.fit(sugar_butter, type_label)
```



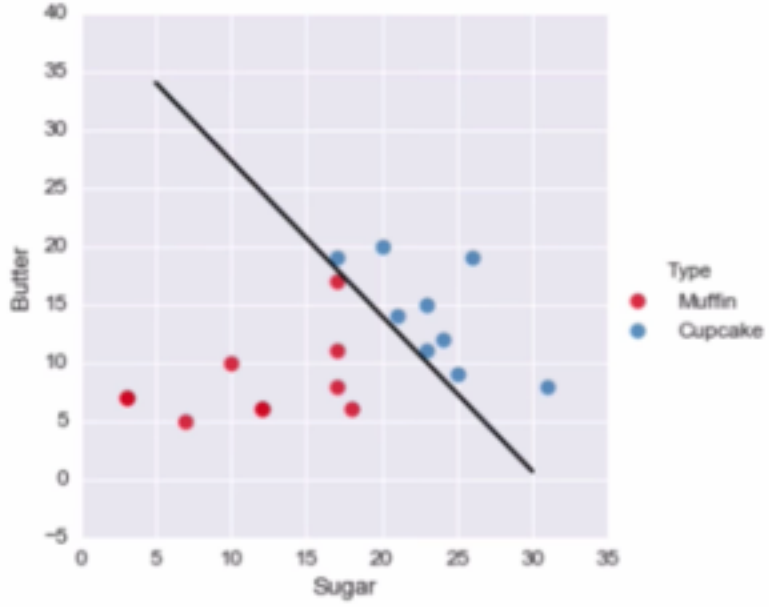
C Parameter: Comparison

```
# Fit the SVM model with a LOW C
model = svm.SVC(kernel='linear', C=2**-5)
model.fit(sugar_butter, type_label)
```

```
# Fit the SVM model with a HIGH C
model = svm.SVC(kernel='linear', C=2**5)
model.fit(sugar_butter, type_label)
```



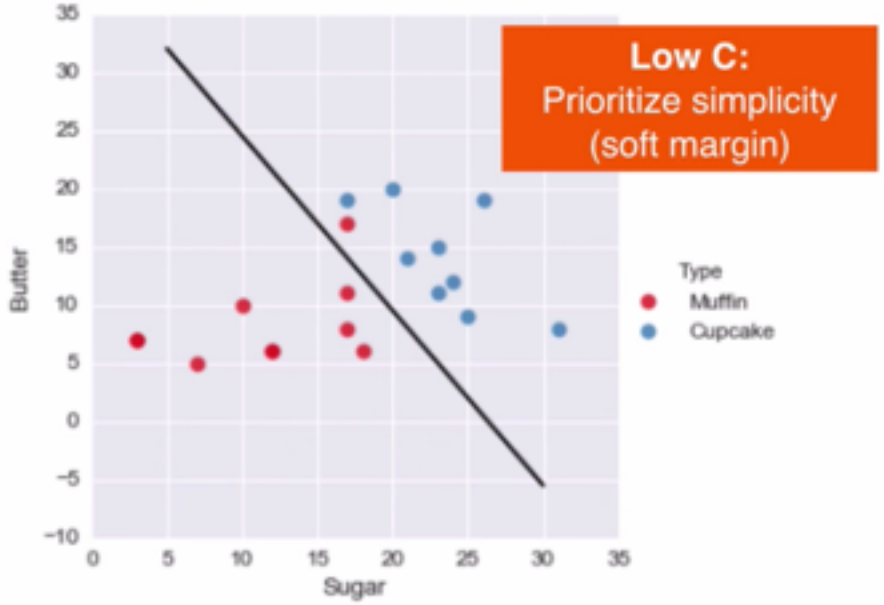
Low C:
Prioritize simplicity
(soft margin)

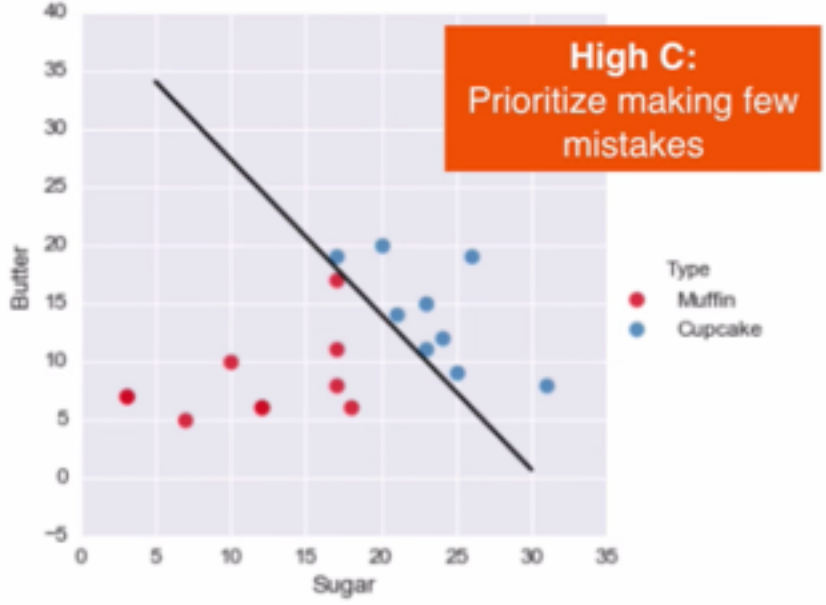


C Parameter: Comparison

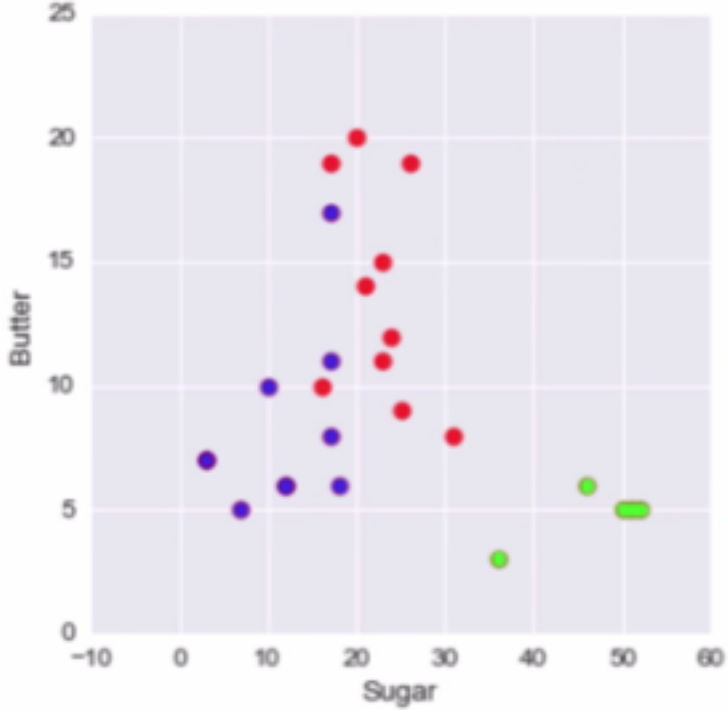
```
# Fit the SVM model with a LOW C
model = svm.SVC(kernel='linear', C=2**-5)
model.fit(sugar_butter, type_label)
```

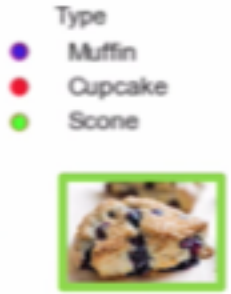
```
# Fit the SVM model with a HIGH C
model = svm.SVC(kernel='linear', C=2**5)
model.fit(sugar_butter, type_label)
```

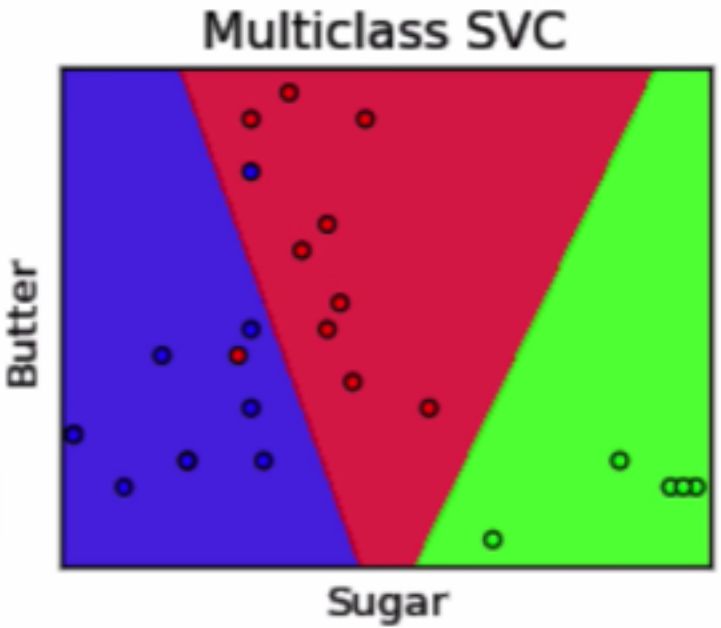




Multiple Classes: Visual







Multiple Classes: Code

Original Code
(2 classes)

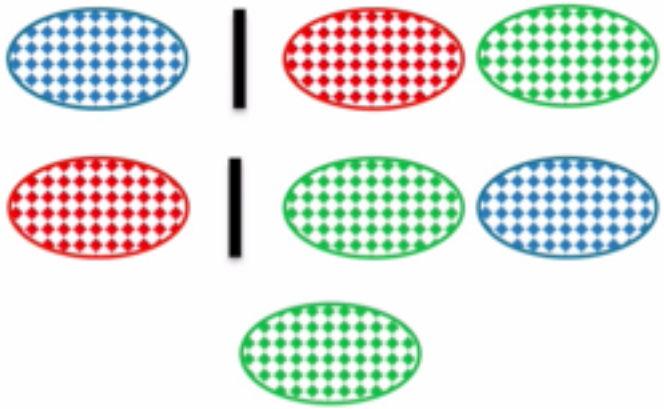
```
# Fit the SVM model
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

Updated Code
(3+ classes)

```
# Fit the SVM model for more than 2 classes
model = svm.SVC(kernel='linear', decision_function_shape='ovr')
model.fit(sugar_butter, type_label)
```

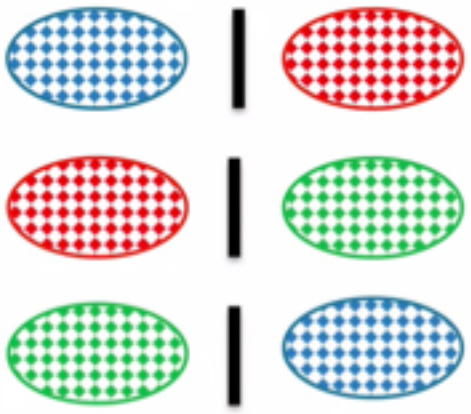
Multiple Classes: Comparison

OVR: One vs Rest



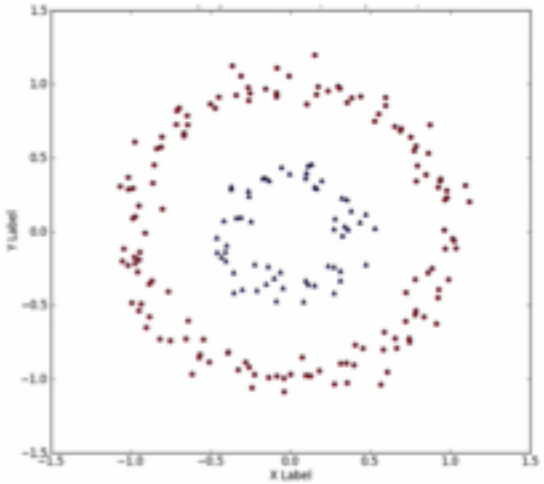
Pros: Fewer classifications
Cons: Classes may be imbalanced

OVO: One vs One

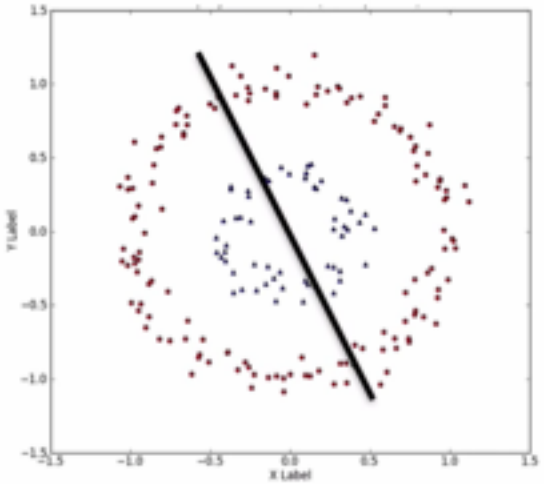


Pros: Less sensitive to imbalance
Cons: More classifications

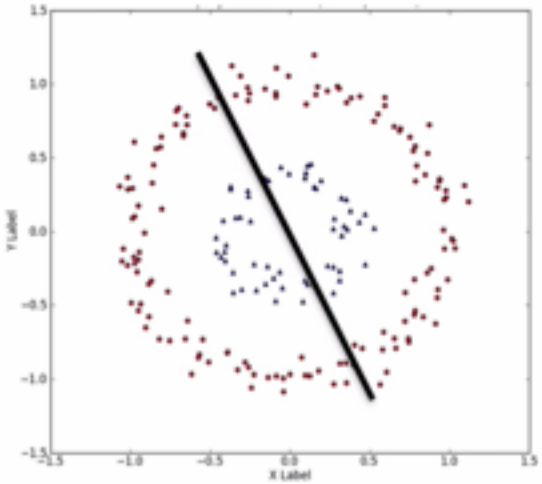
Kernel Trick: Visual



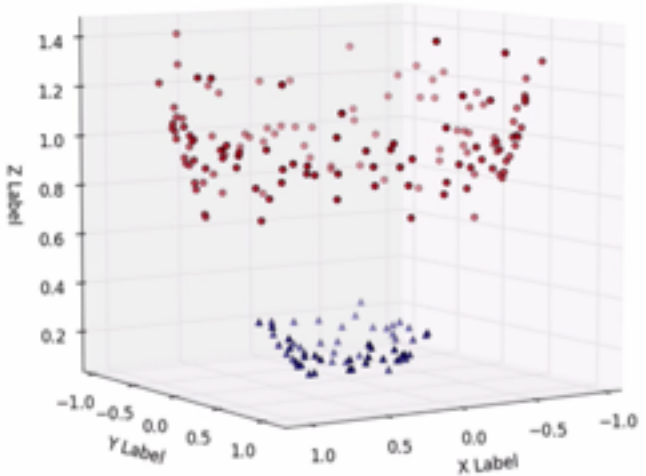
Kernel Trick: Visual



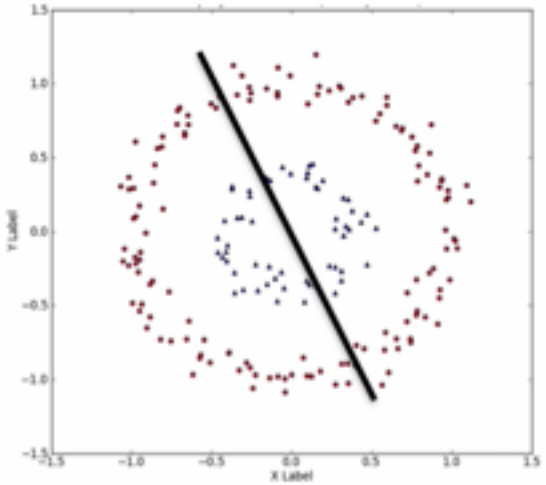
Kernel Trick: Visual



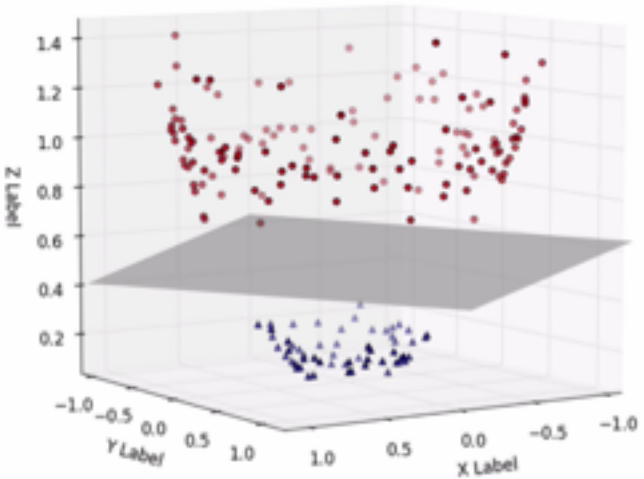
Kernel



Kernel Trick: Visual



Kernel



Kernel Options

- Linear
- Radial Basis Function
- Polynomial
- Sigmoid

Kernel Trick: Code

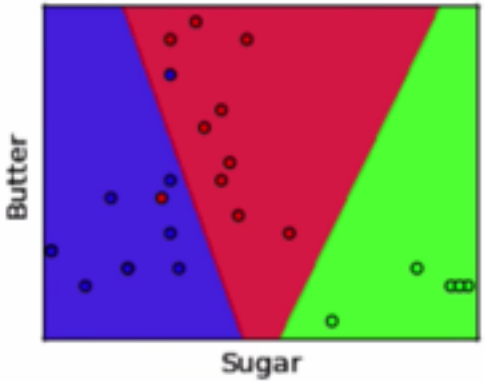
Original Code (linear)

```
# Fit basic SVC model (linear kernel)
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

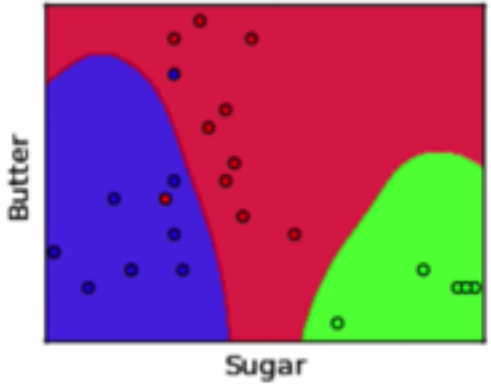
Updated Code (RBF)

```
# Fit the SVC model with radial kernel
model = svm.SVC(kernel='rbf', C=1, gamma=2**-5)
model.fit(sugar_butter, type_label)
```

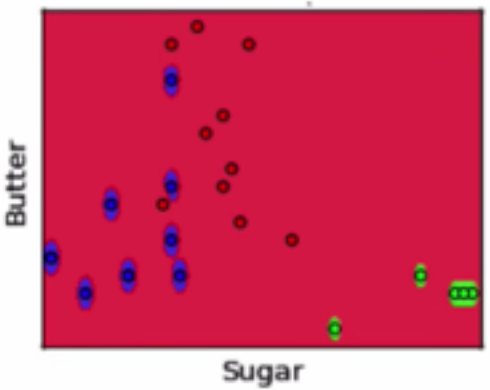
Kernel Trick: Comparison



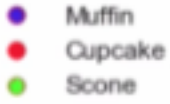
Kernel: Linear
C: 1



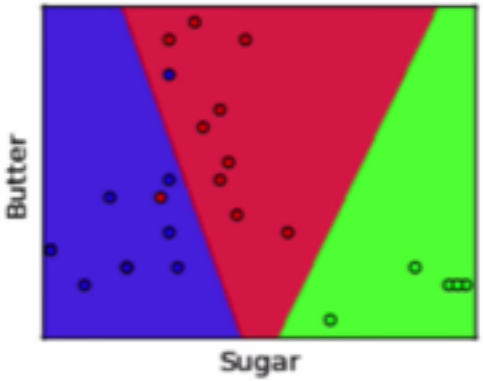
Kernel: RBF
C: 1
Gamma: 2^-5



Kernel: RBF
C: 1
Gamma: 2^1

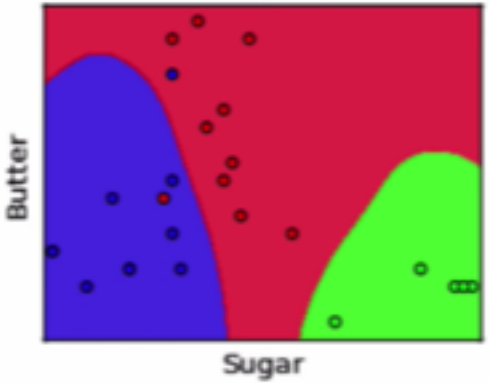


Kernel Trick: Comparison



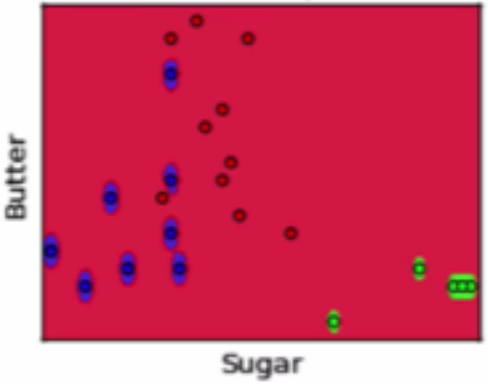
Kernel: Linear
C: 1

Muffin
Cupcake
Scone



Kernel: RBF
C: 1
Gamma: 2^-5

Small Gamma:
Less complexity

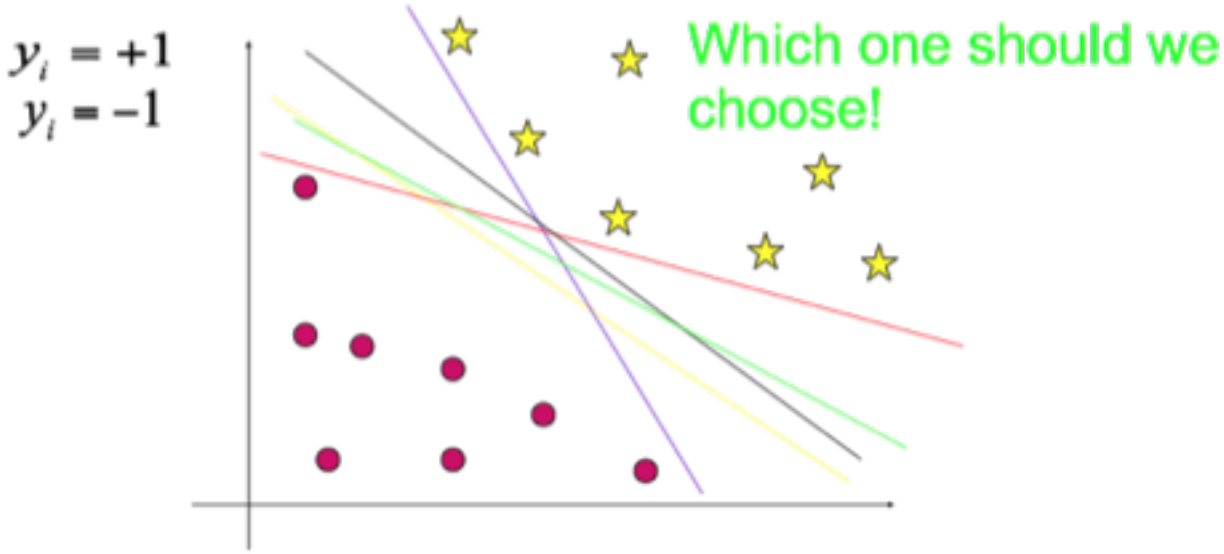


Kernel: RBF
C: 1
Gamma: 2^1

Large Gamma:
More complexity

Pros and Cons of SVM

- **Pros**
  - Good at dealing with high dimensional data
  - Works well on small data sets
- **Cons**
  - Picking the right kernel and parameters can be computationally intensive



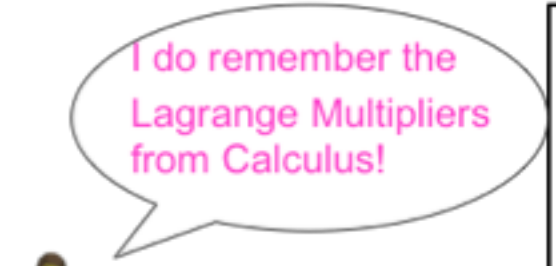
Yes, There are many possible separating hyperplanes

It could be this one or this or this or maybe….!

SVM : Linear separable case.

The optimization problem:

-Our optimization problem so far:



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

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

We will solve this problem by introducing Lagrange multipliers α_i associated with the constrains:

$$\text{minimize} \quad L_p(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i(y_i(x_i \cdot w + b) - 1)$$

$$s.t \quad \alpha_i \geq 0$$

SVM : Linear separable case.

The optimization problem cont':

So our primal optimization problem now:

$$\text{minimize } L_p(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n}\alpha_i(y_i(x_i \cdot w + b) - 1)$$

$$s.t \quad \alpha_i \geq 0$$

We star solving this problem:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

SVM : Linear separable case.

Inroducing The Legrangin Dual Problem.

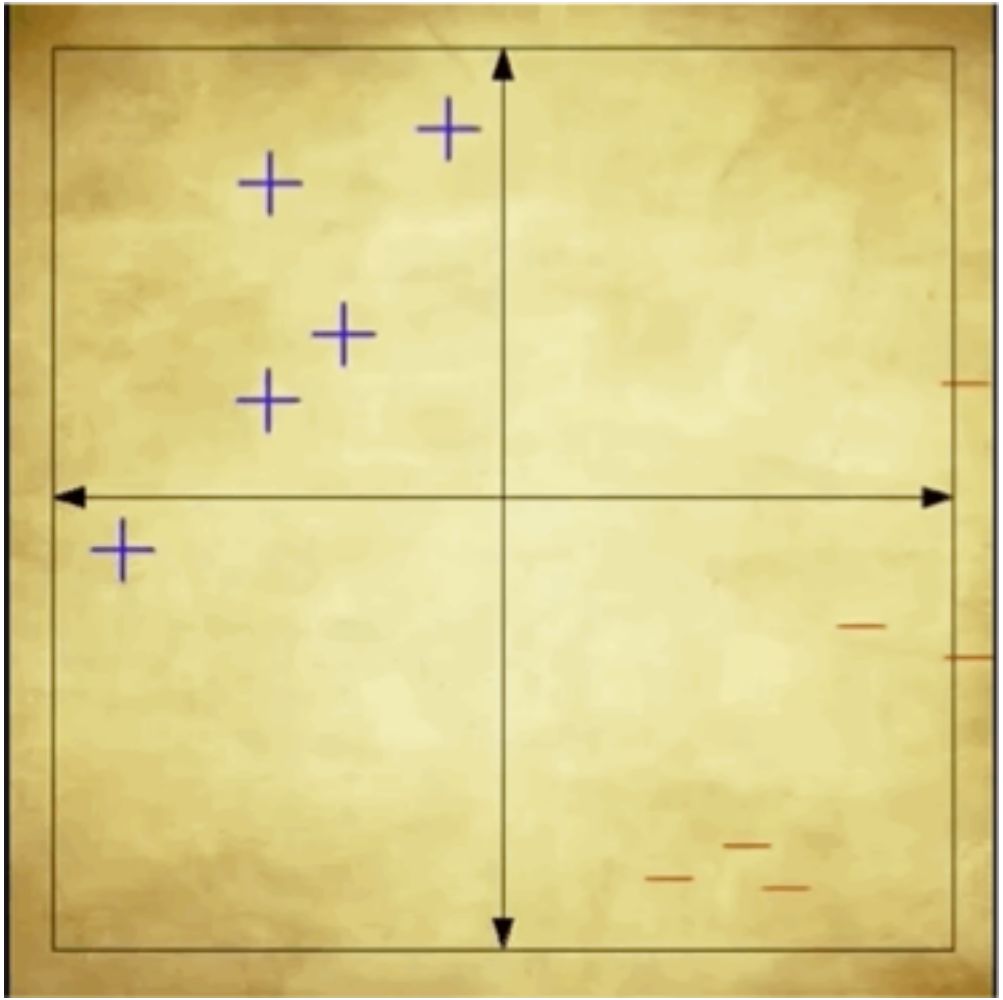
By substituting the above results in the primal problem and doing some math manipulation we get:

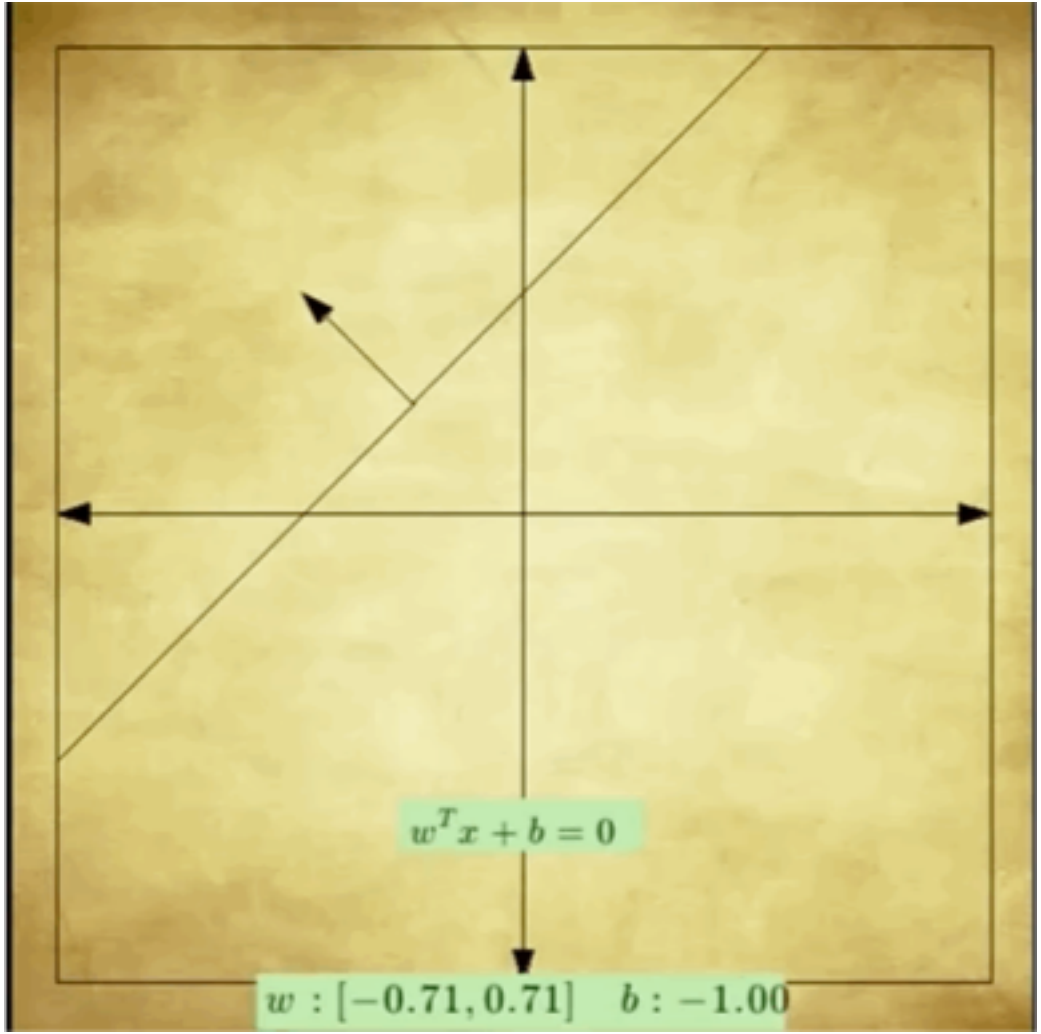
Lagrangian Dual Problem:

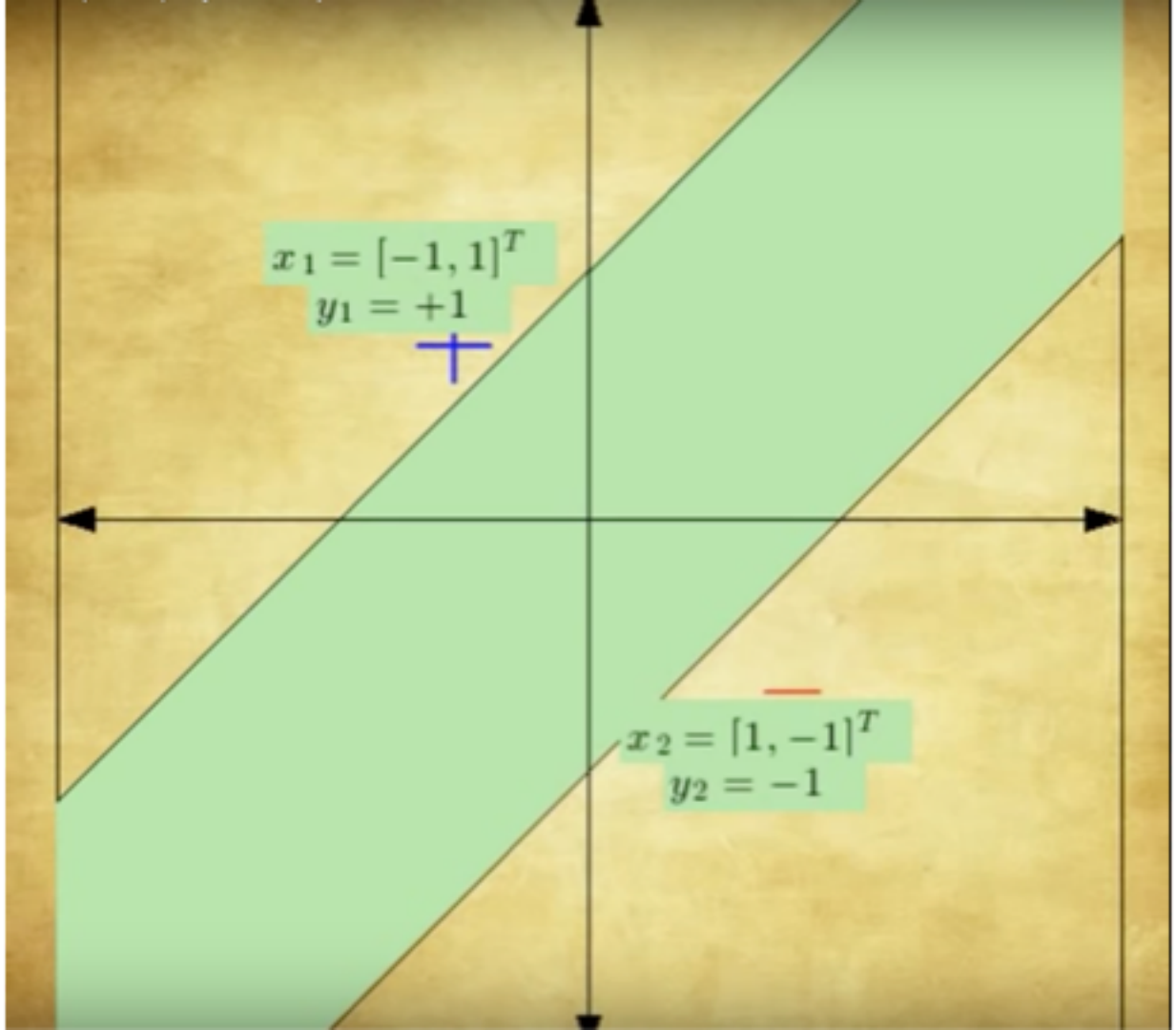
$$\text{maximaize } L_D(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=0}^{n} \sum_{j=0}^{n} \alpha_i \alpha_j y_i y_j x_i^t x_j$$

$$s.t \;\; \alpha_i \geq 0 \;\; and \;\; \sum_{i=1}^{n} \alpha_i y_i = 0$$

$\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ are now our variables, one for each sample point x_i.




$w^T x + b = 0$
$w : [-0.71, 0.71] \quad b : -1.00$


$x_1 = [-1, 1]^T$
$y_1 = +1$
$x_2 = [1, -1]^T$
$y_2 = -1$

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

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

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

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

Constraints:

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

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

Constraints:

plus class is above $\rightarrow w^T x_1 + b \geq 1$

minus class is below $\rightarrow w^T x_2 + b \leq -1$

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

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

Constraints:

plus class is above $\rightarrow w^T x_1 + b \geq 1$

minus class is below $\rightarrow w^T x_2 + b \leq -1$

Multiply each constraint with labels

$y_1 \left(w^T x_1 + b\right) \geq 1$ and $y_2 \left(w^T x_2 + b\right) \geq 1$

$$\max \frac{2}{||w||}$$

$$\max \frac{2}{||w||} \rightarrow \min \frac{||w||}{2}$$

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

Constraints:

plus class is above $\rightarrow w^T x_1 + b \geq 1$

minus class is below $\rightarrow w^T x_2 + b \leq -1$

Multiply each constraint with labels

$y_1 \left(w^T x_1 + b\right) \geq 1$ and $y_2 \left(w^T x_2 + b\right) \geq 1$

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

subject to $y_i \left(w^T x_i + b\right) - 1 \geq 0$

Inserting Lagrange multipliers: α_i:

$$L_{pd} = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(w^T x_i + b \right) - 1 \right]$$

Karush-Kuhn-Tucker conditions:

$$\frac{\partial L_{pd}}{\partial \mathbf{w}} = 0$$

$$\frac{dL_{pd}}{db} = 0$$

$$\alpha_i \geq 0$$

Karush-Kuhn-Tucker conditions:

$$\frac{\partial L_{pd}}{\partial \mathbf{w}} = 0$$

$$\frac{dL_{pd}}{db} = 0$$

$$\alpha_i \geq 0$$

$$\alpha_i \left[y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right] = 0$$

Karush-Kuhn-Tucker conditions:

$$\frac{\partial L_{pd}}{\partial \mathbf{w}} = 0$$

$$\frac{dL_{pd}}{db} = 0$$

$$\alpha_i \geq 0$$

$$\alpha_i \left[y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right] = 0$$

$$\frac{\partial L_{pd}}{\partial \mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Let's put the knowns in place:

$y_1 = 1, y_2 = -1$

$\mathbf{x}_1 = [-1, 1]^T, \mathbf{x}_2 = [1, -1]^T$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Let's put the knowns in place:

$y_1 = 1, y_2 = -1$

$\mathbf{x}_1 = [-1, 1]^T, \mathbf{x}_2 = [1, -1]^T$

$L_d = -\alpha_1^2 - \alpha_2^2 - 2\alpha_1\alpha_2 + \alpha_1 + \alpha_2$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Let's put the knowns in place:

$y_1 = 1, y_2 = -1$

$\mathbf{x}_1 = [-1, 1]^T, \mathbf{x}_2 = [1, -1]^T$

$L_d = -\alpha_1^2 - \alpha_2^2 - 2\alpha_1\alpha_2 + \alpha_1 + \alpha_2$

$\frac{\partial L_d}{\partial \alpha} = \mathbf{0} \rightarrow \alpha_1 + \alpha_2 = 1/2$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Let's put the knowns in place:

$y_1 = 1, y_2 = -1$

$\mathbf{x}_1 = [-1, 1]^T, \mathbf{x}_2 = [1, -1]^T$

$L_d = -\alpha_1^2 - \alpha_2^2 - 2\alpha_1\alpha_2 + \alpha_1 + \alpha_2$

$$\frac{\partial L_d}{\partial \alpha} = \mathbf{0} \rightarrow \alpha_1 + \alpha_2 = 1/2$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \rightarrow \alpha_1 = \alpha_2$$

Applying KKT to primal-dual, I get

$$L_d = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Let's put the knowns in place:

$y_1 = 1, y_2 = -1$

$\mathbf{x}_1 = [-1, 1]^T, \mathbf{x}_2 = [1, -1]^T$

$L_d = -\alpha_1^2 - \alpha_2^2 - 2\alpha_1\alpha_2 + \alpha_1 + \alpha_2$

$$\frac{\partial L_d}{\partial \alpha} = \mathbf{0} \rightarrow \alpha_1 + \alpha_2 = 1/2$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \rightarrow \alpha_1 = \alpha_2$$

$\alpha_1 + \alpha_2 = 1/2 \rightarrow \alpha_1 = \alpha_2 = 1/4$

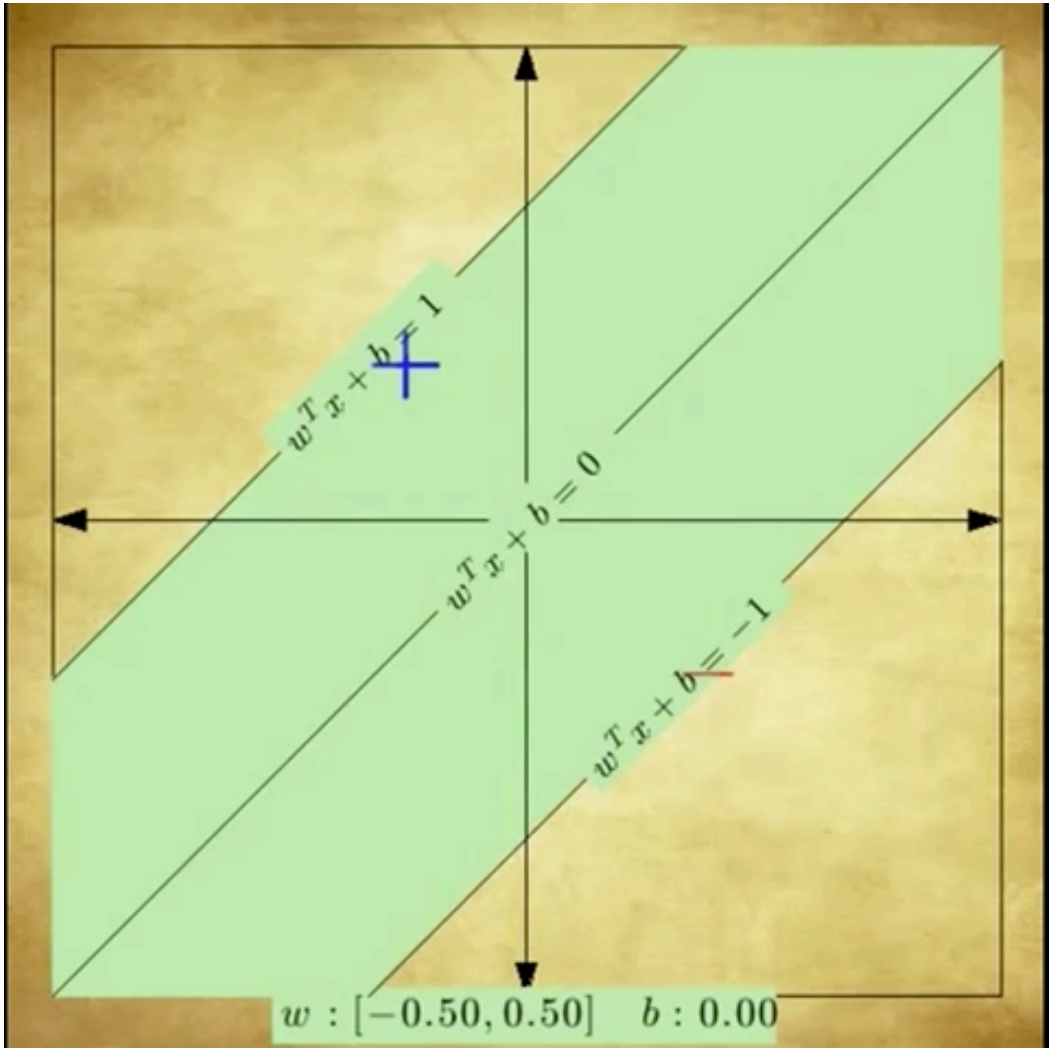
$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = [-1/2, 1/2]^T$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = [-1/2, 1/2]^T$$

$$\alpha_i \left[y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right] = 0$$

$$b = 1/y_i - \mathbf{w}^T \mathbf{x}_i \quad \forall i \quad \text{s.t.} \quad \alpha_i \neq 0$$

$$b = 0 \quad \forall i$$


$w^T x + b = 1$
$w^T x + b = 0$
$w^T x + b = -1$
$w : [-0.50, 0.50] \quad b : 0.00$

THANK YOU