

Support Vector Machines

A Visual Explanation + Sample Python Code

https://github.com/MrinalGitHub/GyanVriksh_Course

Agenda

1. SVM Basics

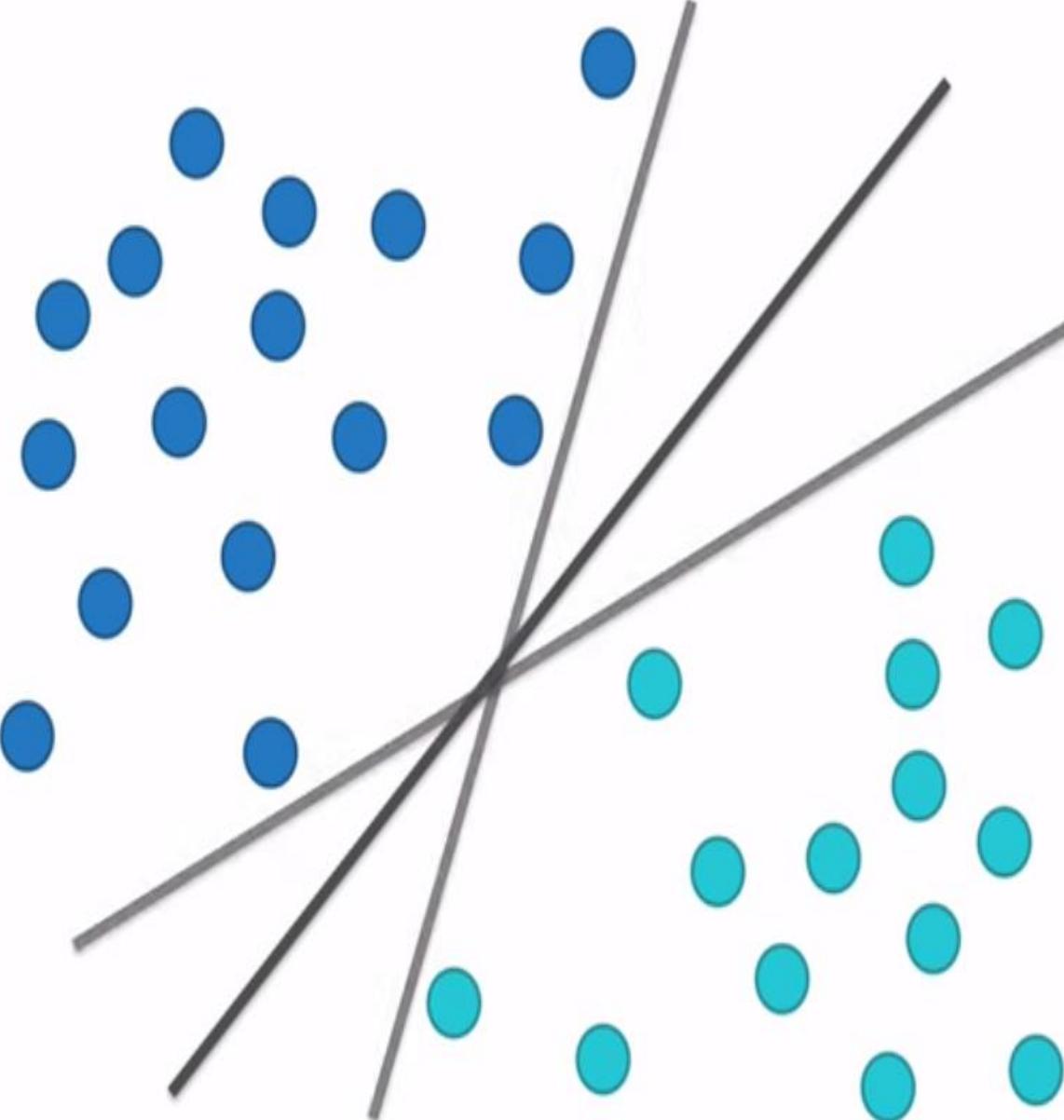
- Visual introduction
- Example in Python

2. Additional Complexities

- Higher dimensions
- Multiple classes
- C parameter
- Kernel trick

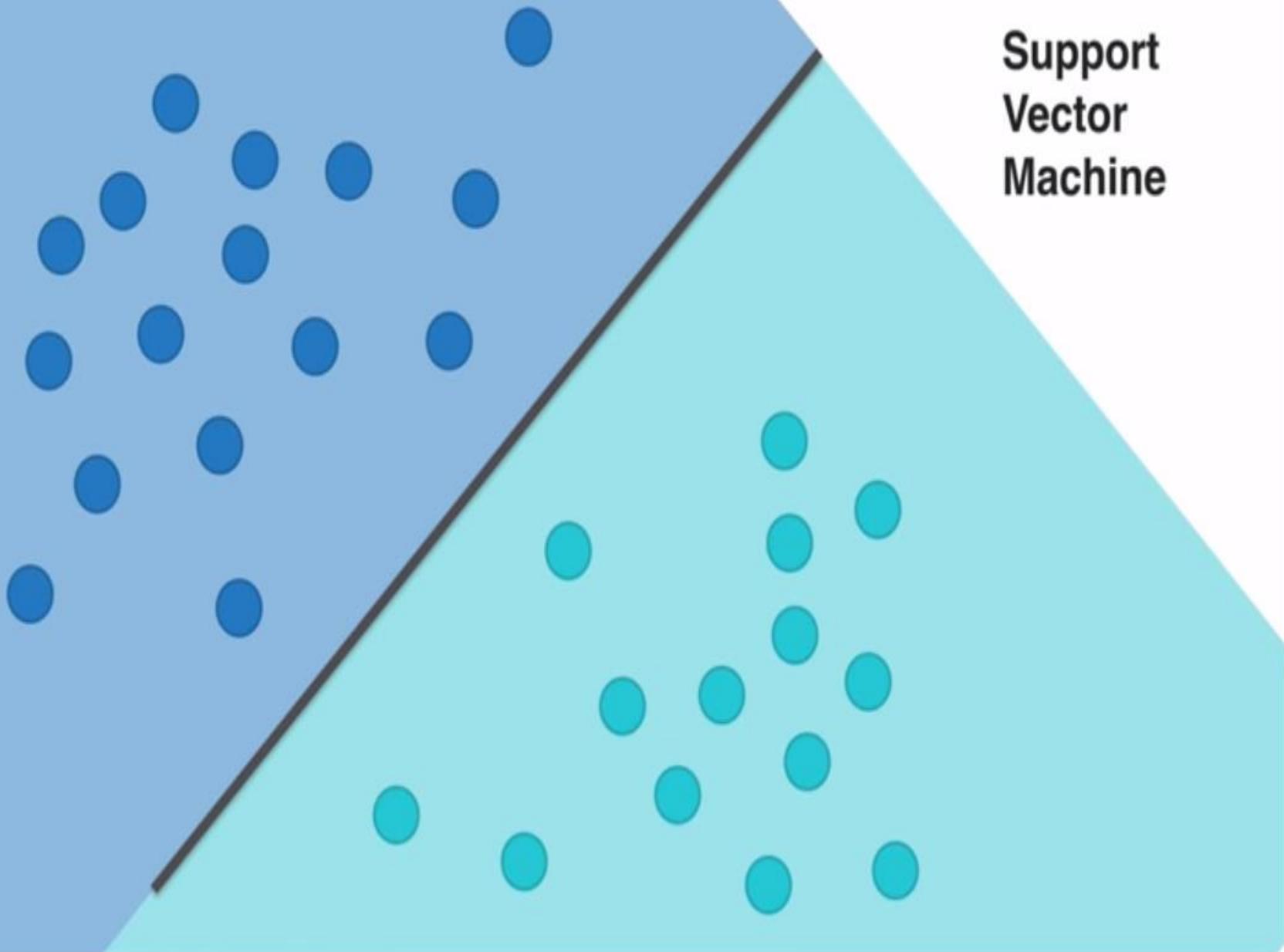
3. Closing Remarks

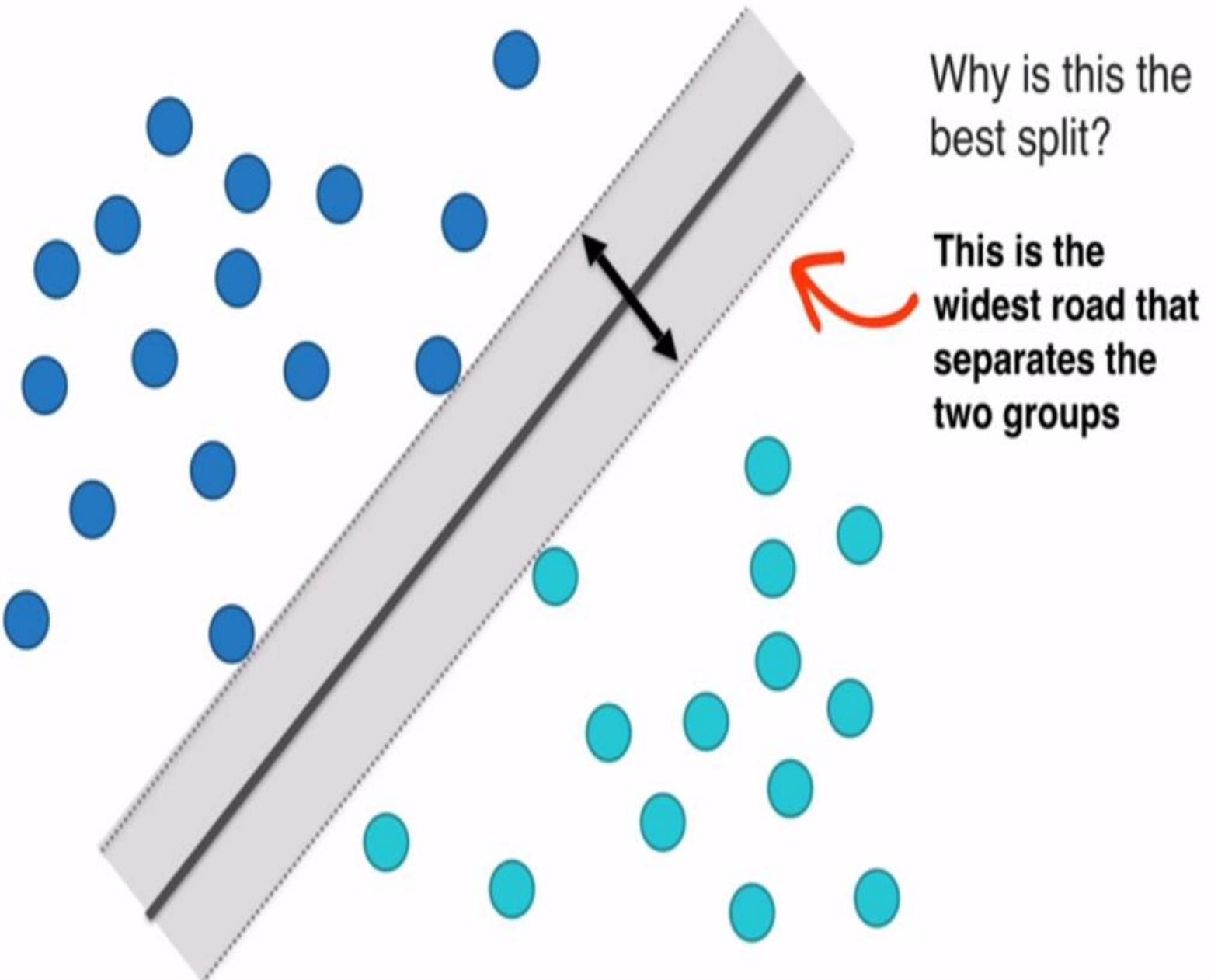
- Pros and cons
- Other techniques

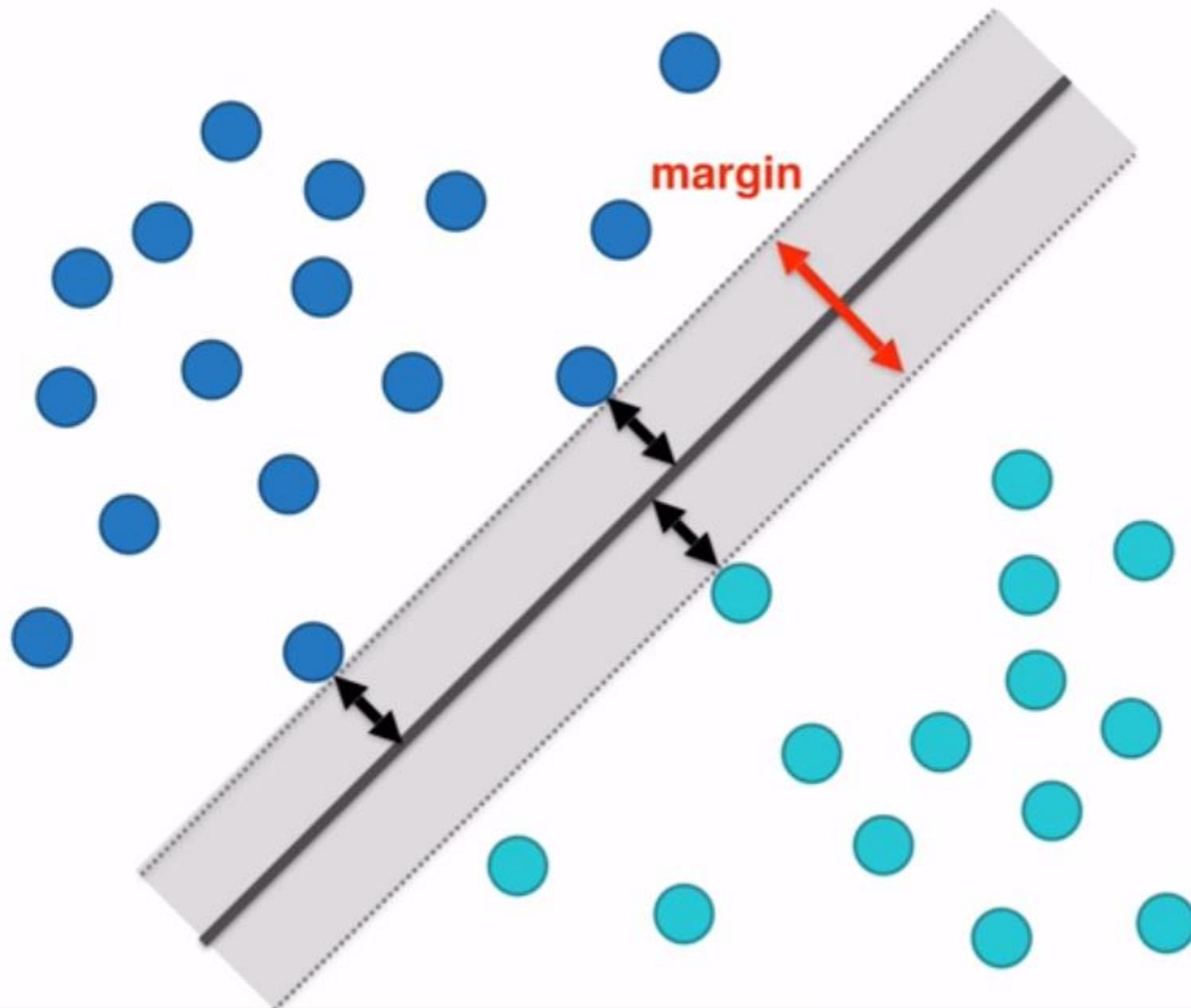


Split the data
in the best
possible way

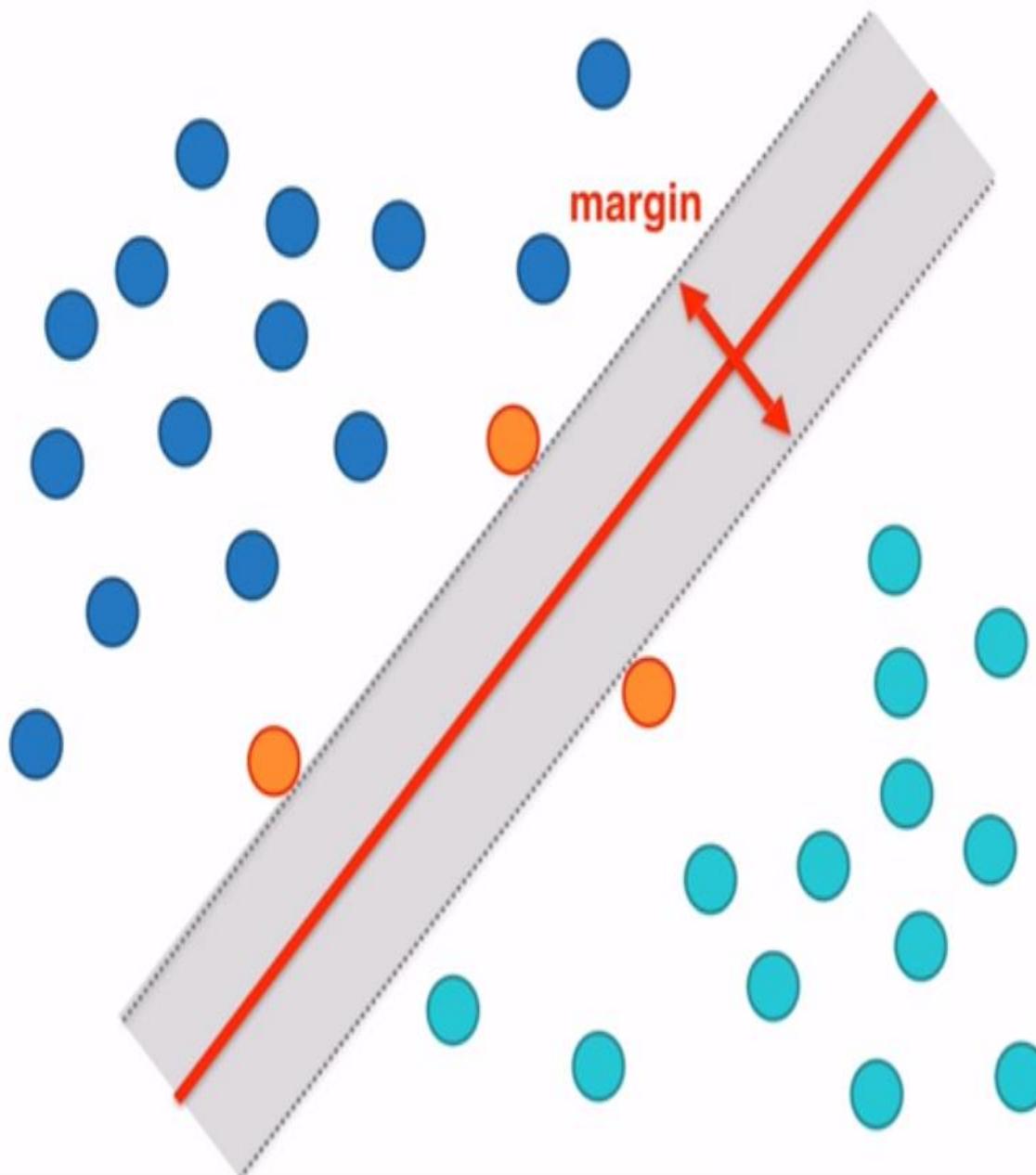
Support Vector Machine





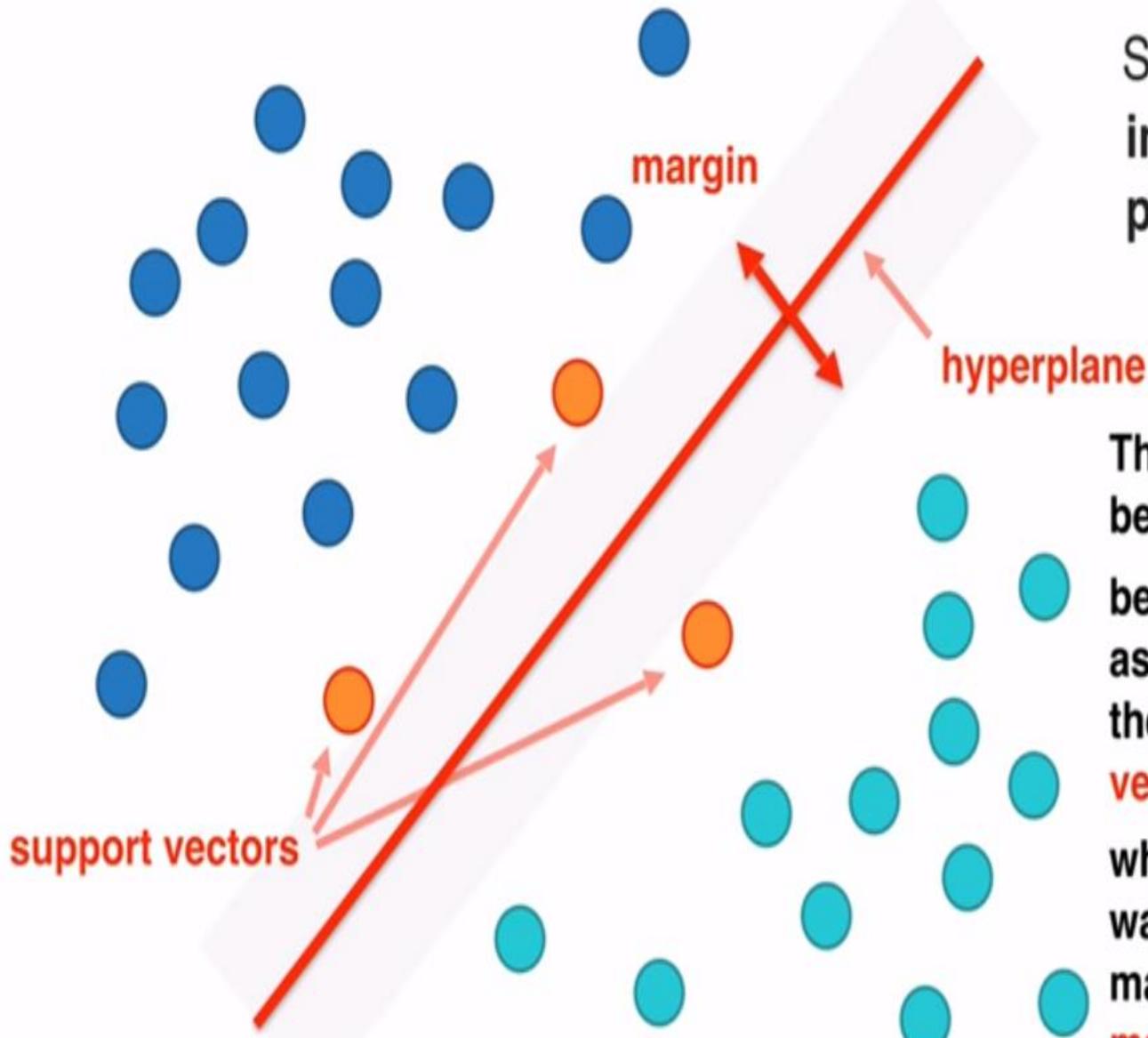


w
b
The
be
per
lin
per



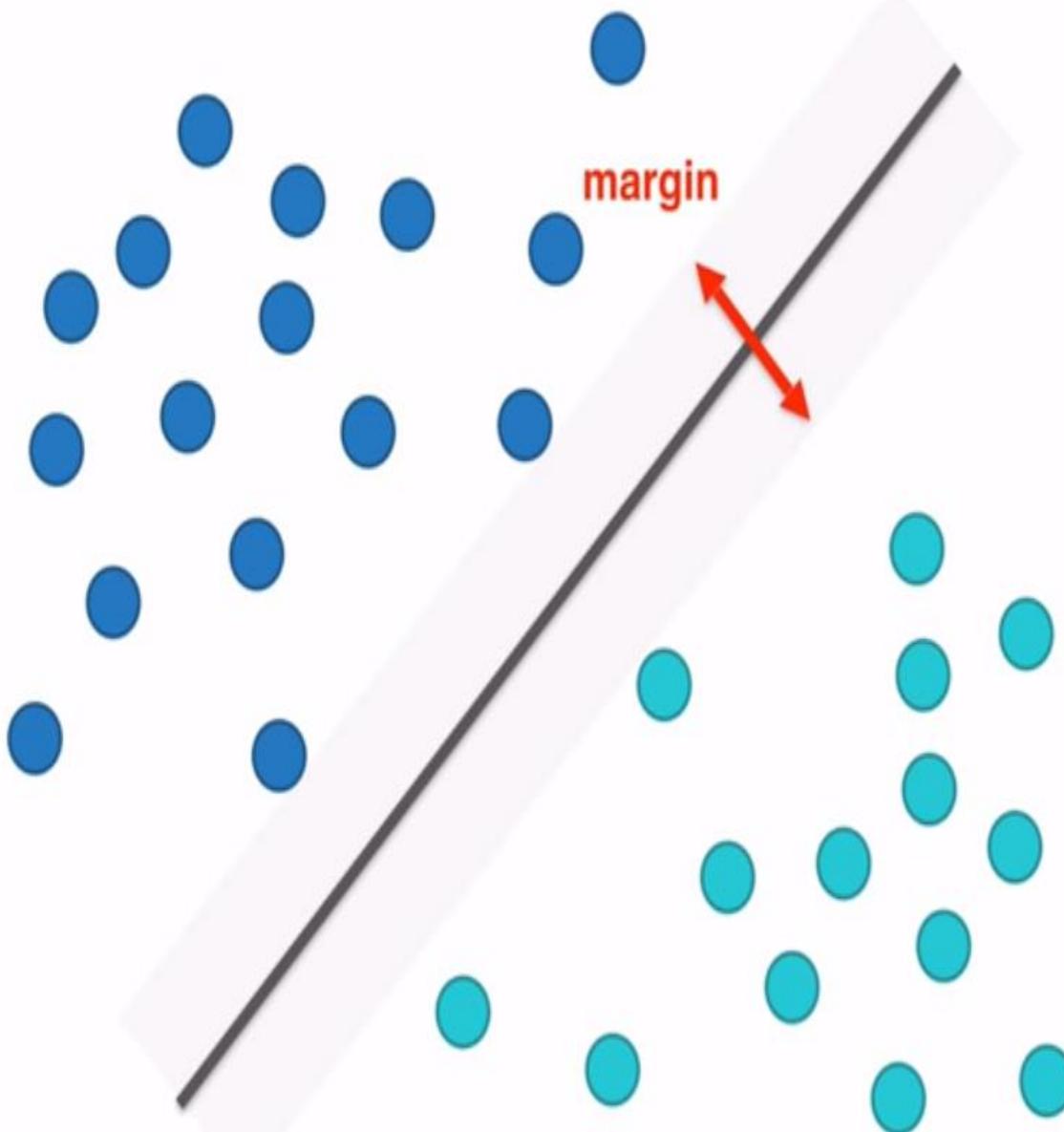
Why is this the best split?

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



Split the data
in the best
possible way

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



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



Muffins



versus

“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



Recorded the top 10 muffin and top 10 cupcake recipes

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 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

jupyter 8.1.0 - Demo Lab (ipython3) [running]

File Edit Insert Cell Kernel Help

Step 1: Import Package

```
# Step 1: Import libraries to support our code
import pandas as pd
import numpy as np
from sklearn import linear_model
```

Step 2: Import Data

```
# Step 2: Load the muffin and cupcakes dataset
muffins = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/muffins.csv')
muffins.head()
```

	Type	Fat	Sugar	Egg	Baking Powder	Vinegar	Lat
0	Muffin	0.5	1.0	0.5	0.5	0.5	0.5
1	Muffin	0.5	1.0	0.5	0.5	0.5	0.5
2	Muffin	0.5	1.0	0.5	0.5	0.5	0.5
3	Muffin	0.5	1.0	0.5	0.5	0.5	0.5
4	Muffin	0.5	1.0	0.5	0.5	0.5	0.5

Step 3: Prepare the Data

```
# Step 3: Drop the unnecessary columns
muffins.drop(['Type', 'Baking Powder', 'Vinegar', 'Lat'], axis=1, inplace=True)
```

Step 4: Visualize the Data

Scatter plot showing the relationship between Egg and Sugar for muffins. The x-axis is labeled 'Egg' and ranges from 0 to 1. The y-axis is labeled 'Sugar' and ranges from 0 to 1. The data points are mostly clustered in the lower-left quadrant, with a few outliers in the upper-right quadrant.

```
# Step 5: Specify the regression line model
regressor = linear_model.LinearRegression()
regressor.fit(muffins[['Egg']], muffins['Sugar'])
```

Step 6: Fit the Model

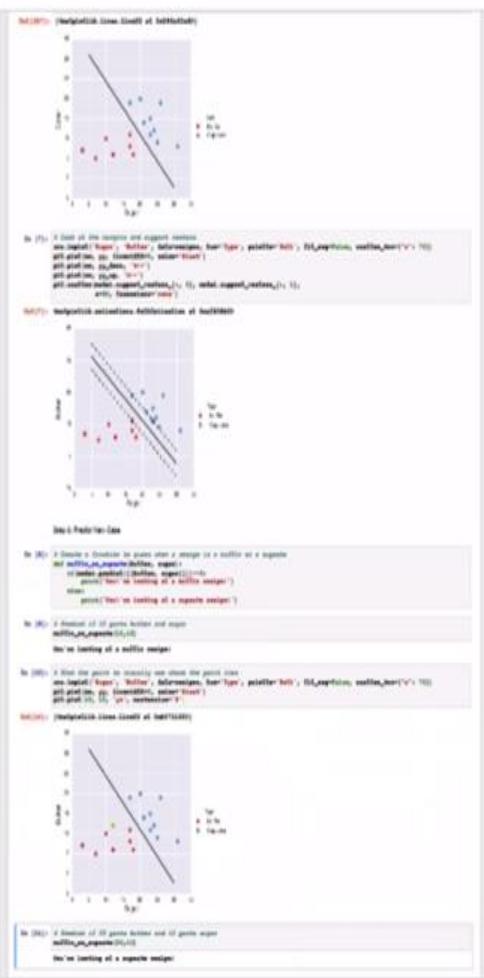
```
# Step 6: Fit the regression line model
regressor = linear_model.LinearRegression()
regressor.fit(muffins[['Egg']], muffins['Sugar'])
```

Step 7: Visualize Results

```
# Step 7: Plot the regression line
x = np.linspace(0, 1, 100)
y = regressor.intercept_ + regressor.coef_[0] * x
plt.plot(x, y)
```

Step 8: Predict New Case

```
# Step 8: Predict the regression line
new_case = np.array([0.5])
new_case = new_case.reshape(1, -1)
new_case[0][0] = 0.5
new_case[0][0] = 1.0
new_case[0][0] = 0.5
new_case[0][0] = 0.5
new_case[0][0] = 0.5
new_case[0][0] = 0.5
```



a. Import Libraries

b. Import Data

c. Prepare the Data

d. Fit the Model

e. Visualize Results

f. Predict New Case

a. Import Libraries

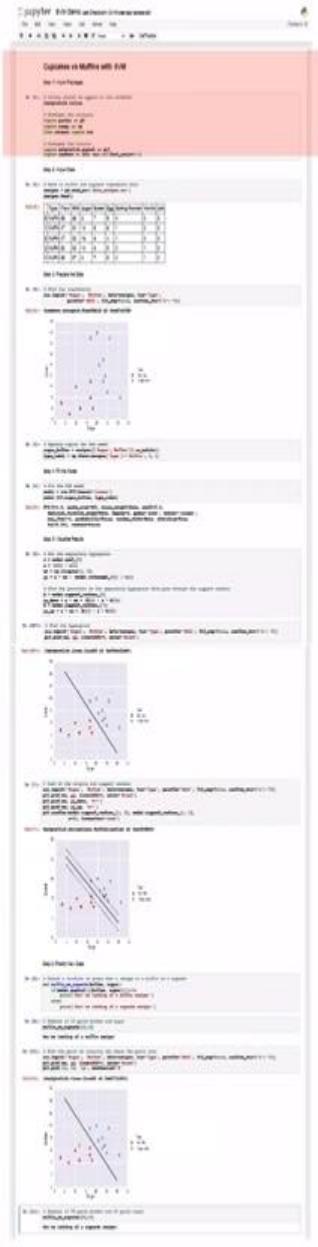
Cupcakes vs Muffins with SVM

Step 1: Import Libraries

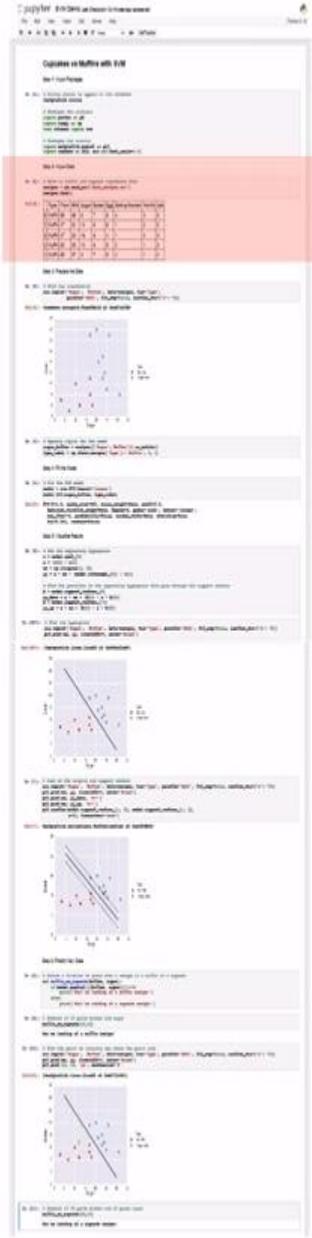
```
In [13]: # Allows charts to appear in the notebook
%matplotlib inline

# 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]:

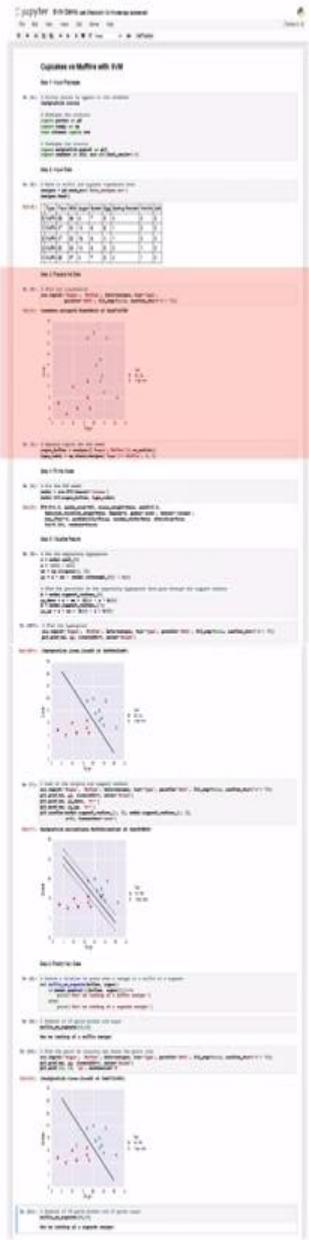
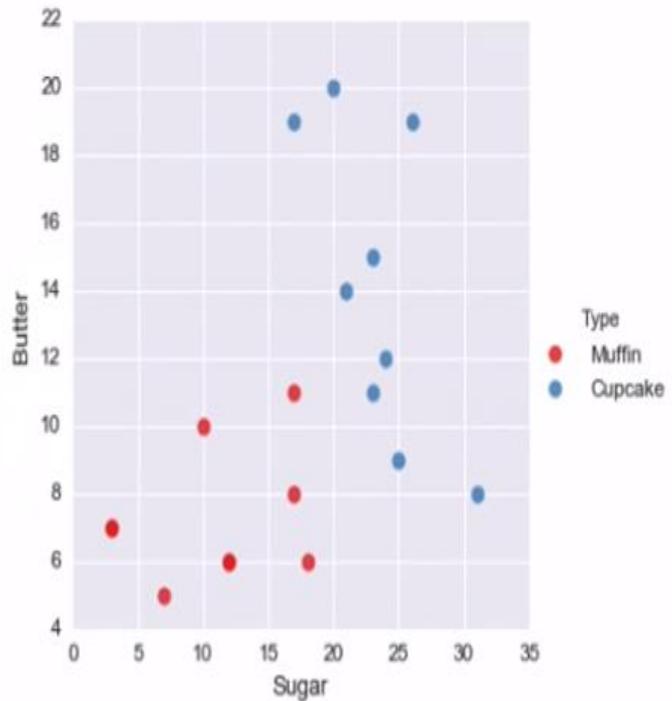
	Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
0	Muffin	55	28	3	7	5	2	0	0
1	Muffin	47	24	12	6	9	1	0	0
2	Muffin	47	23	18	6	4	1	0	0
3	Muffin	50	25	12	6	5	2	1	0
4	Muffin	55	27	3	7	5	2	1	0

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>
```



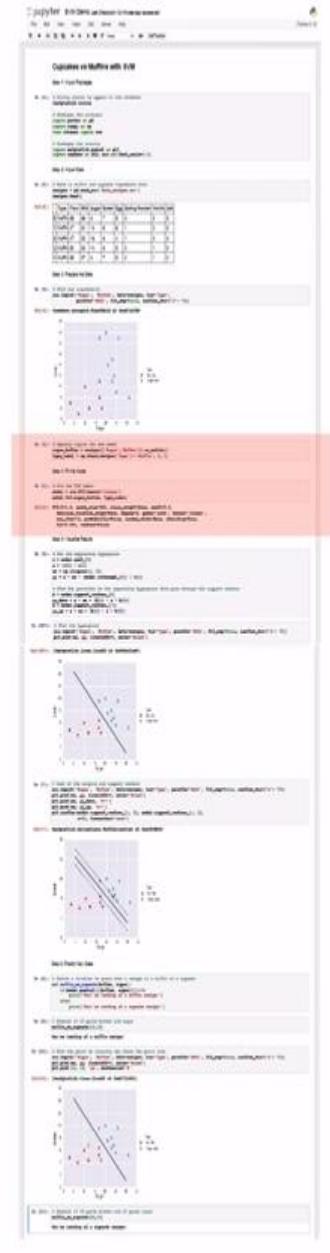
d. Fit the Model

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)
```

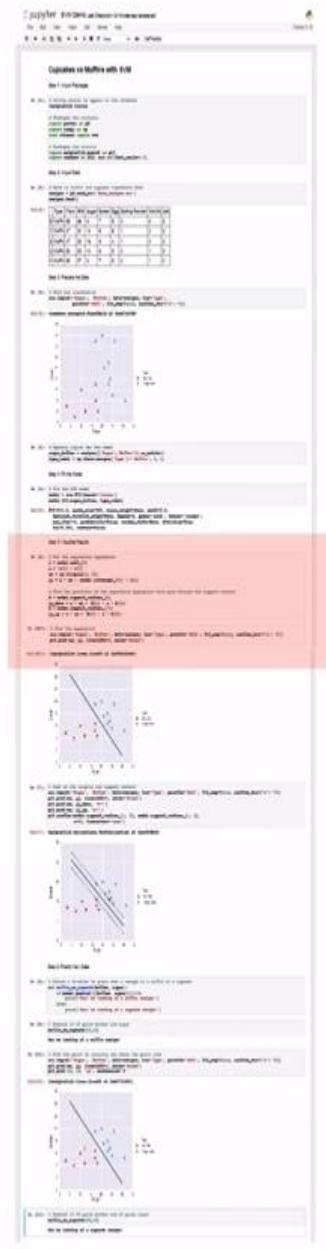


e. Visualize Results

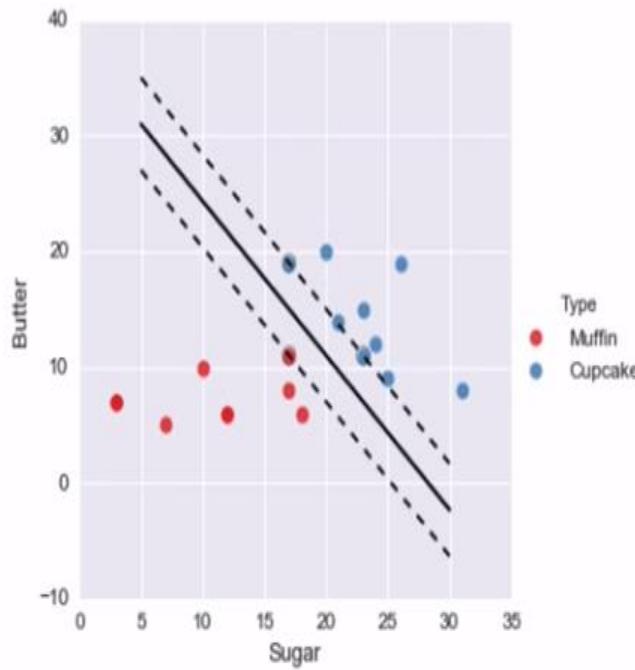
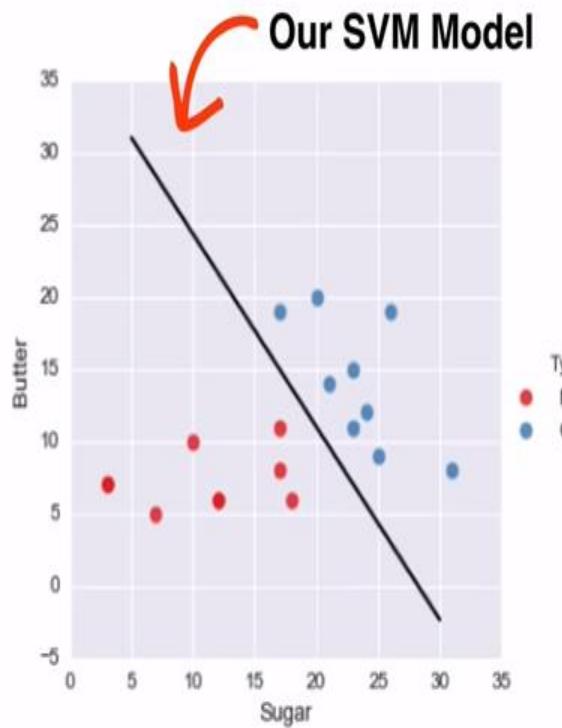
Step 5: Visualize Results

```
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



Jupyter Notebook titled "Cupcakes vs Butter with SVM". The notebook contains the following code:

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

The notebook also includes several plots showing the SVM model's performance, including a confusion matrix and classification report.

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

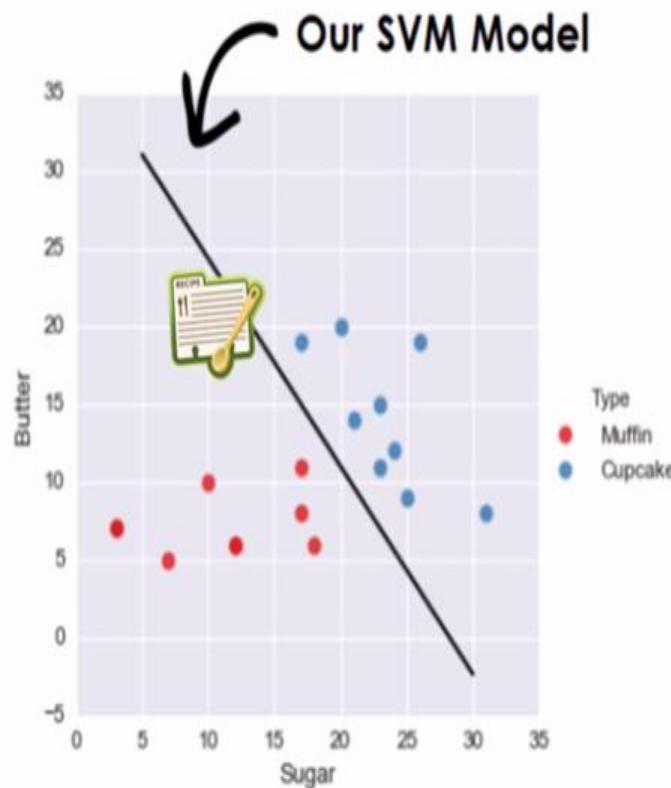
1. Find the data ✓
2. Apply a data science model ✓
3. Review the results

3. Review the Results

The Challenge

Classify recipes as cupcakes or muffins. ✓

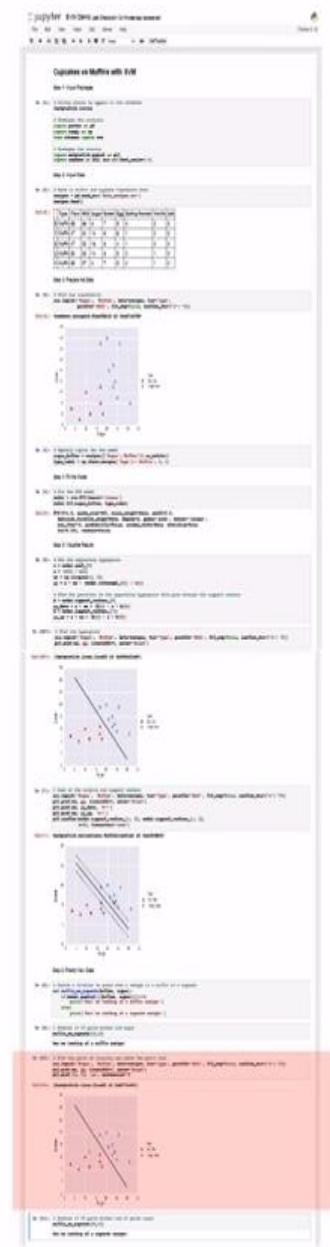
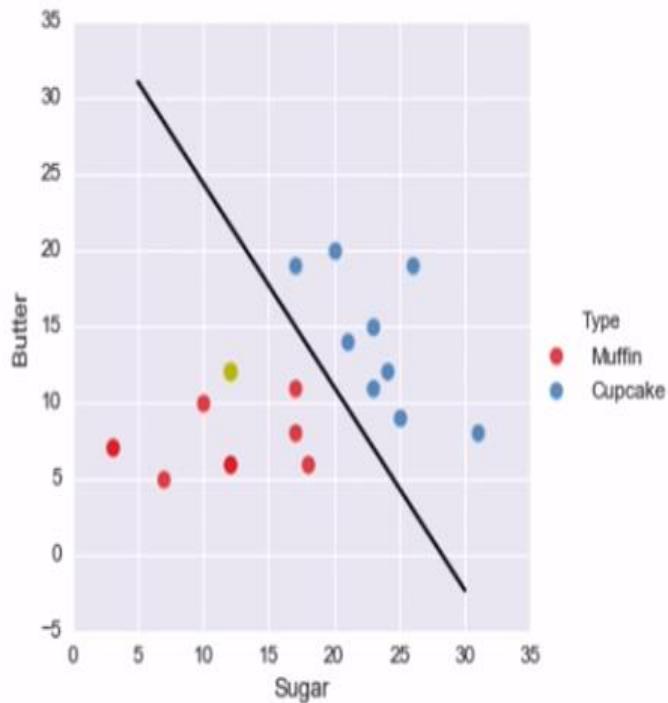
When given a new recipe, determine if it's a cupcake or a muffin.



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]: [
```



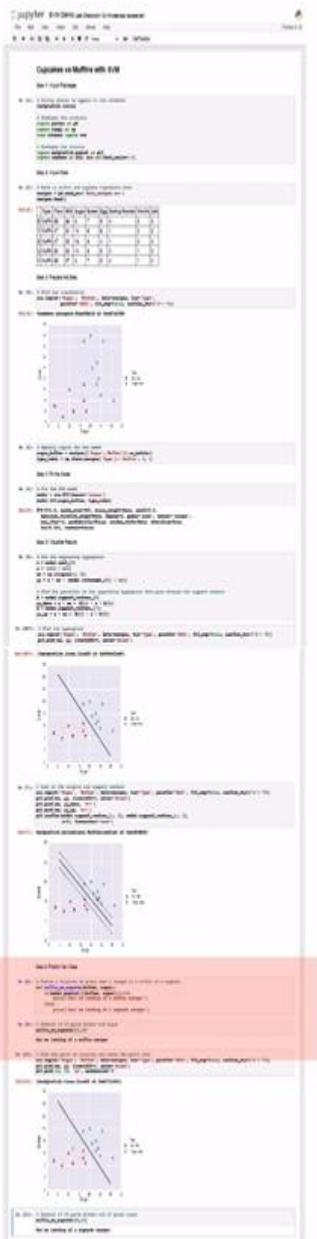
f. Predict New Case

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



Muffins



versus

~~"a cupcake is just a muffin with frosting"~~

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

Basic case ✓

**Up next: How to make SVM
even more powerful**

Agenda

1. SVM Basics

- Visual introduction
- Example in Python

2. Additional Complexities

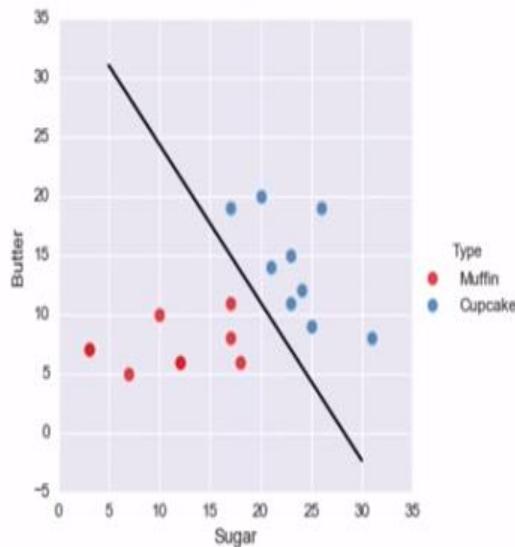
- Higher dimensions
- Multiple classes
- C parameter
- Kernel trick

3. Closing Remarks

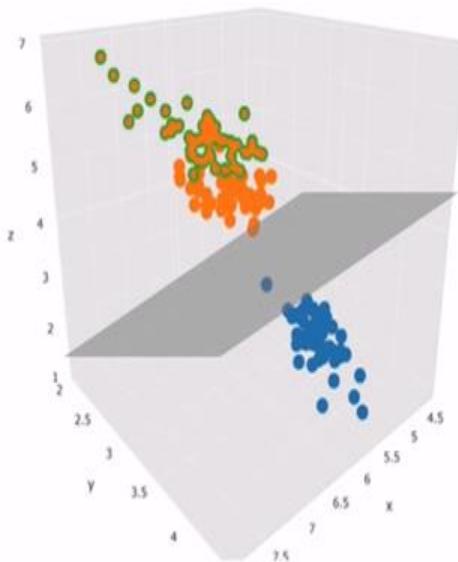
- Pros and cons
- Other techniques

Higher Dimensions: Visual

2D: Separate
with Line



3D: Separate
with Plane



4D+: Separate
with Hyperplane

Hard to
Visualize

Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick



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

C Parameter

Multiple Classes

Kernel Trick

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)
```

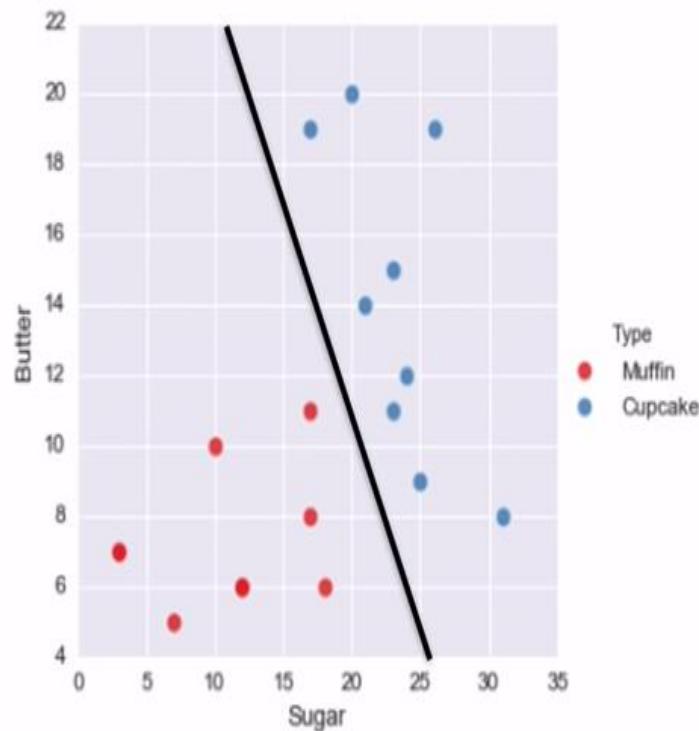
```
[ 3,  7],  
[12,  6],  
[18,  6],  
[12,  6],  
[ 3,  7],...
```

all_ingredients

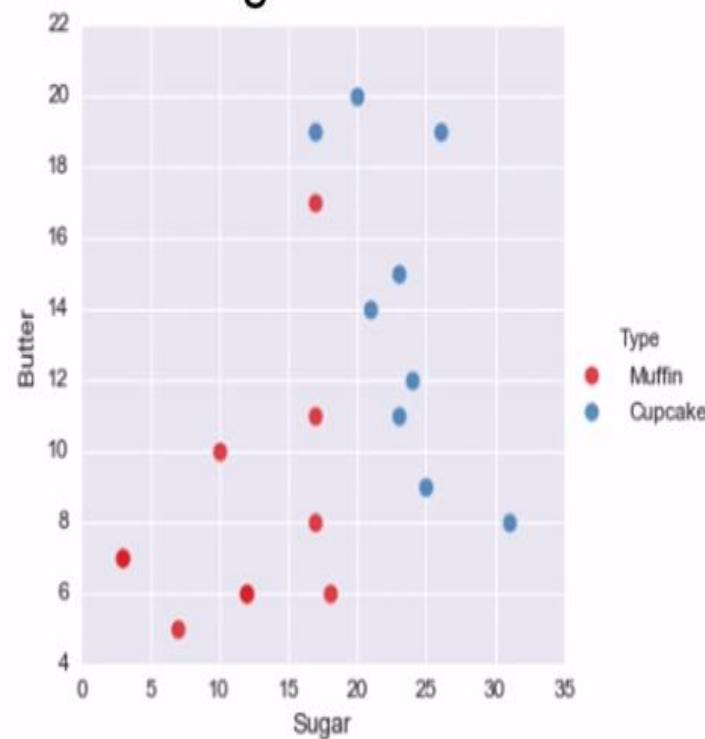
```
[55, 28,  3,  7,  5,  2,  0,  0],  
[47, 24, 12,  6,  9,  1,  0,  0],  
[47, 23, 18,  6,  4,  1,  0,  0],  
[50, 25, 12,  6,  5,  2,  1,  0],  
[55, 27,  3,  7,  5,  2,  1,  0],...
```

C Parameter: Visual

Demo Data



Original Data



Higher Dimensions

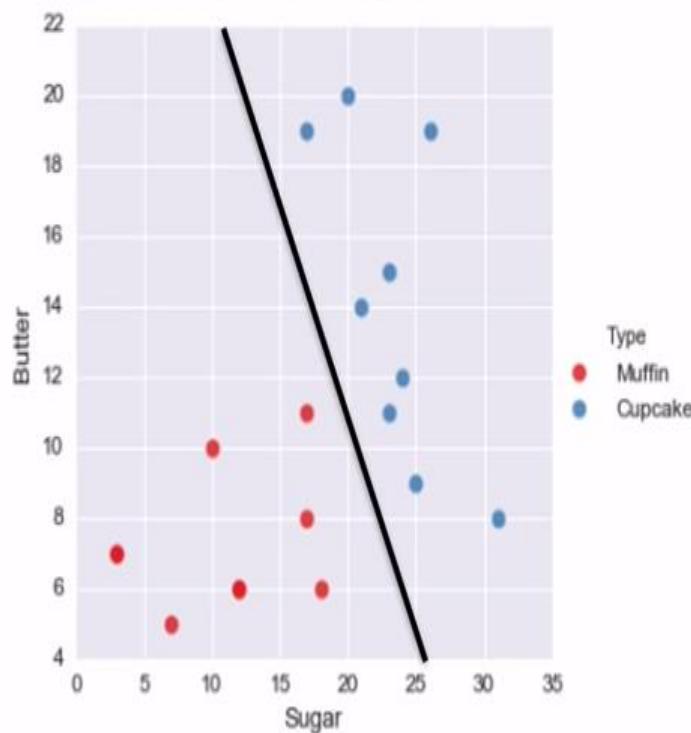
C Parameter

Multiple Classes

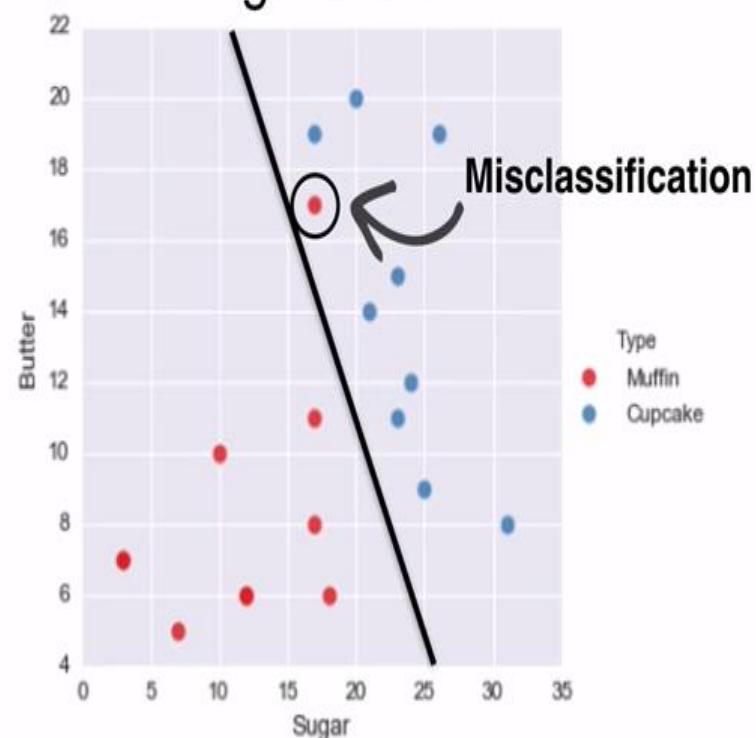
Kernel Trick

C Parameter: Visual

Demo Data



Original Data



Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

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

Higher Dimensions

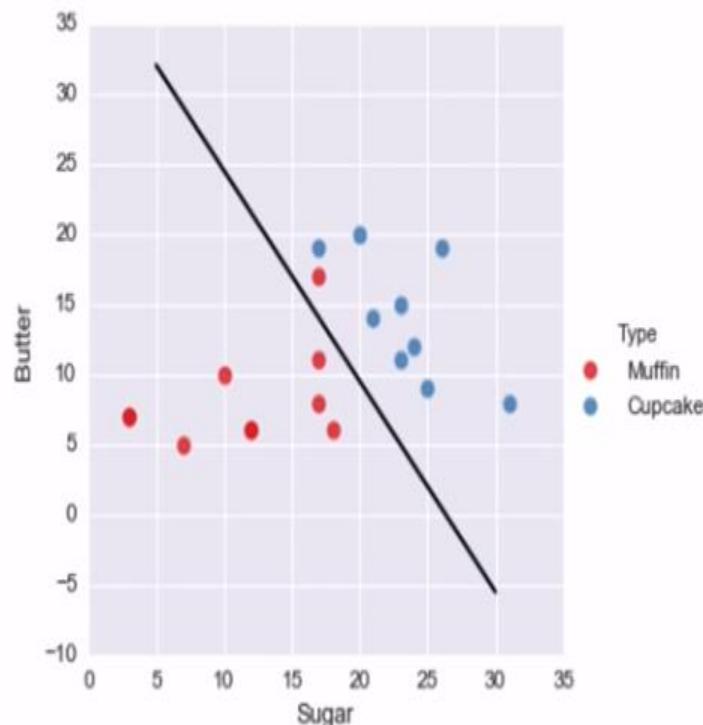
C Parameter

Multiple Classes

Kernel Trick

C Parameter: Comparison

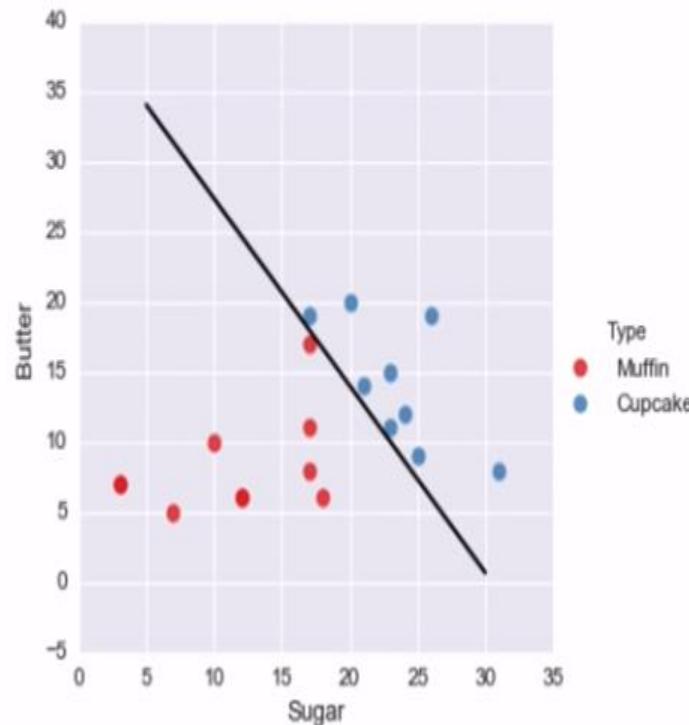
```
# Fit the SVM model with a LOW C  
model = svm.SVC(kernel='linear', C=2**-5)  
model.fit(sugar_butter, type_label)
```



Higher Dimensions

C Parameter

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

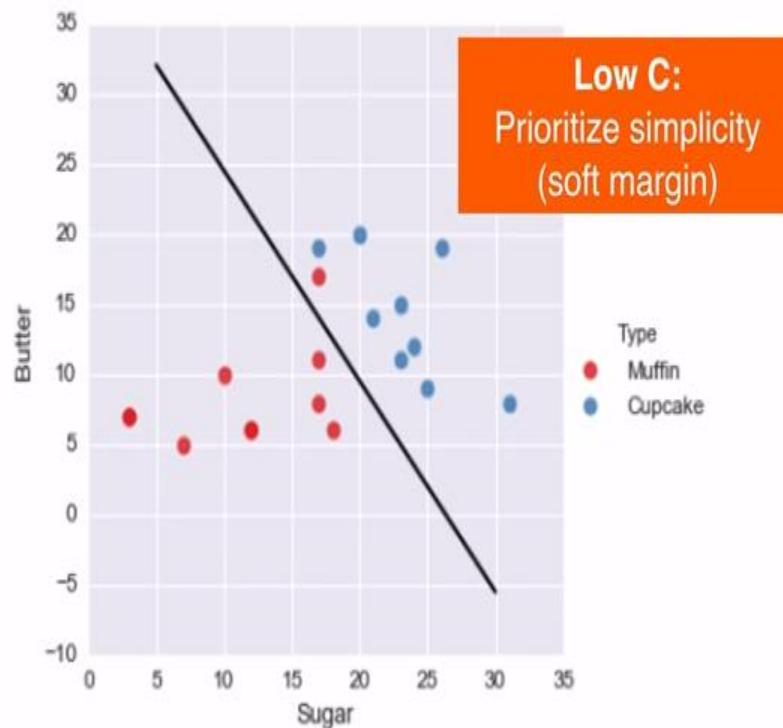


Multiple Classes

Kernel Trick

C Parameter: Comparison

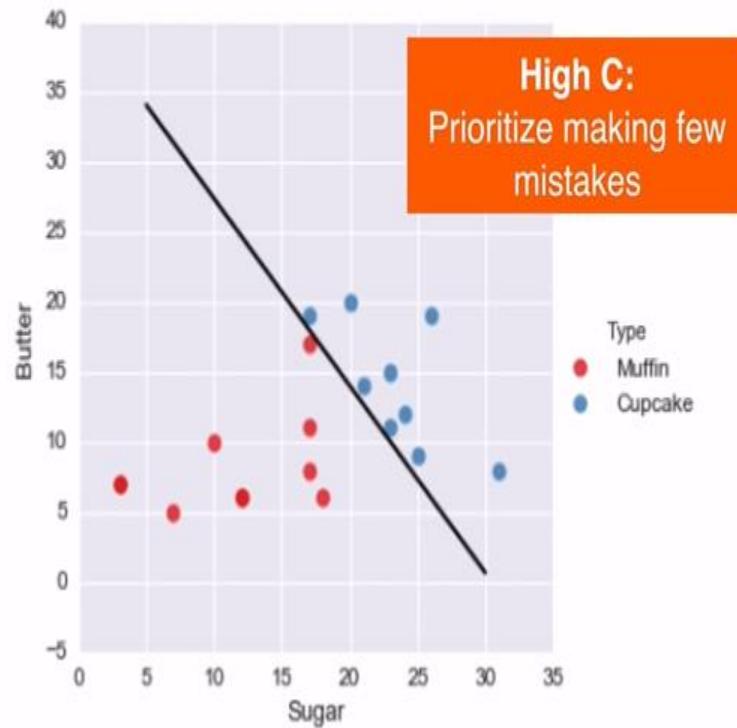
```
# Fit the SVM model with a LOW C  
model = svm.SVC(kernel='linear', C=2**-5)  
model.fit(sugar_butter, type_label)
```



Higher Dimensions

C Parameter

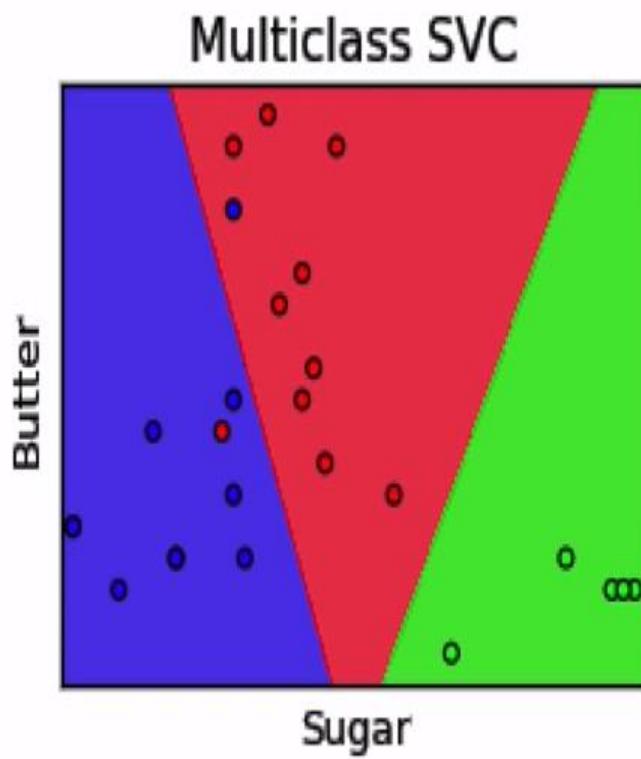
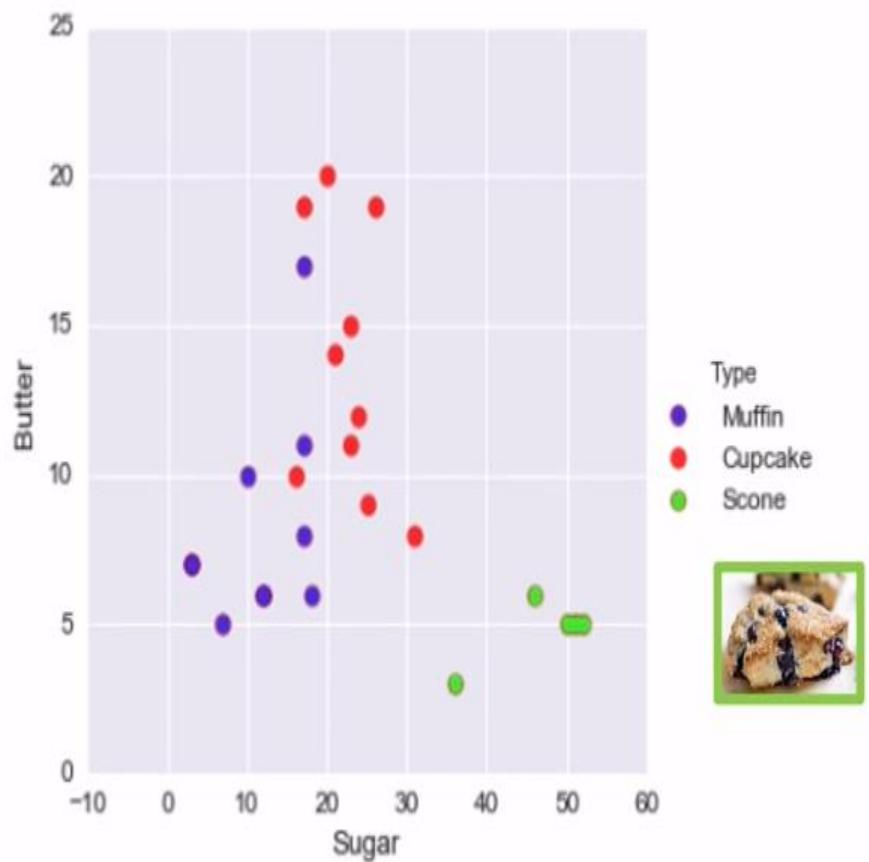
```
# Fit the SVM model with a HIGH C  
model = svm.SVC(kernel='linear', C=2**5)  
model.fit(sugar_butter, type_label)
```



Multiple Classes

Kernel Trick

Multiple Classes: Visual



Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

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)
```

Higher Dimensions

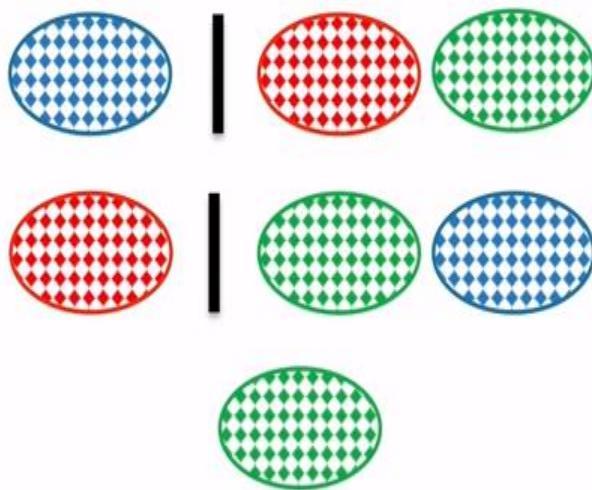
C Parameter

Multiple Classes

Kernel Trick

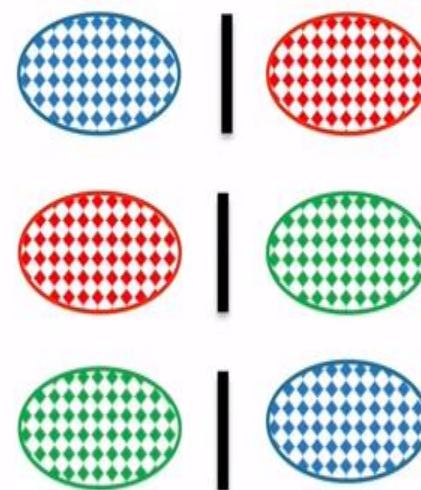
Multiple Classes: Comparison

OVR: One vs Rest



Pros: Fewer classifications
Cons: Classes may be imbalanced

OVO: One vs One



Pros: Less sensitive to imbalance
Cons: More classifications

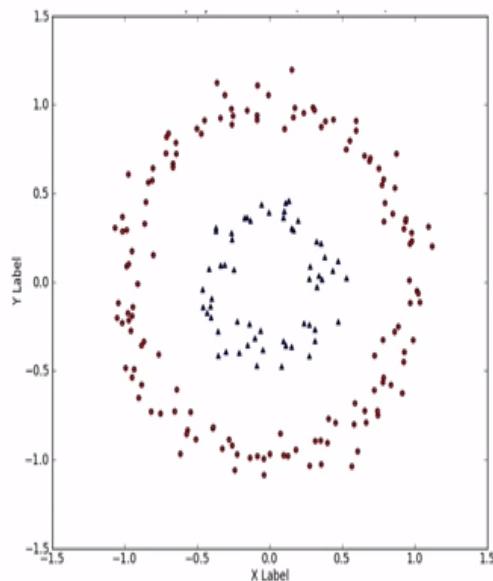
Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

Kernel Trick: Visual



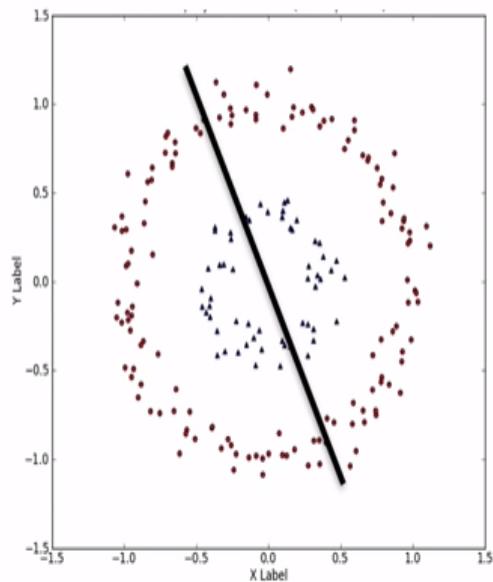
Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

Kernel Trick: Visual



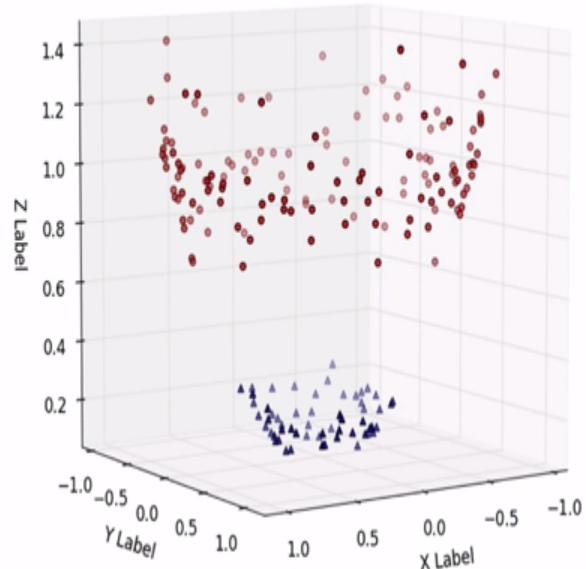
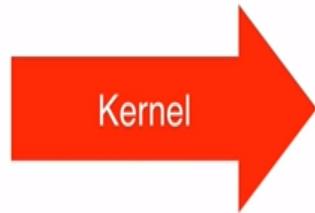
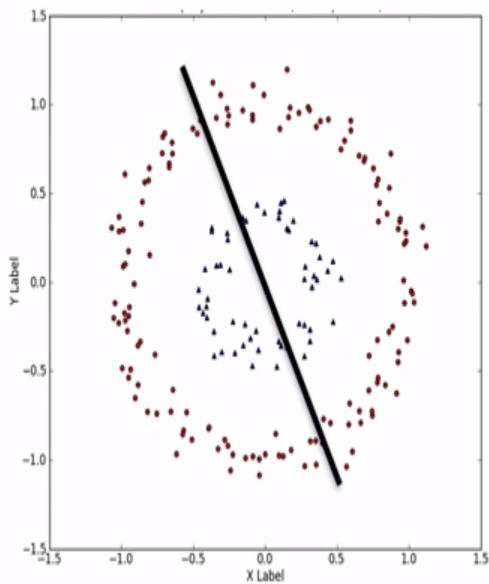
Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

Kernel Trick: Visual



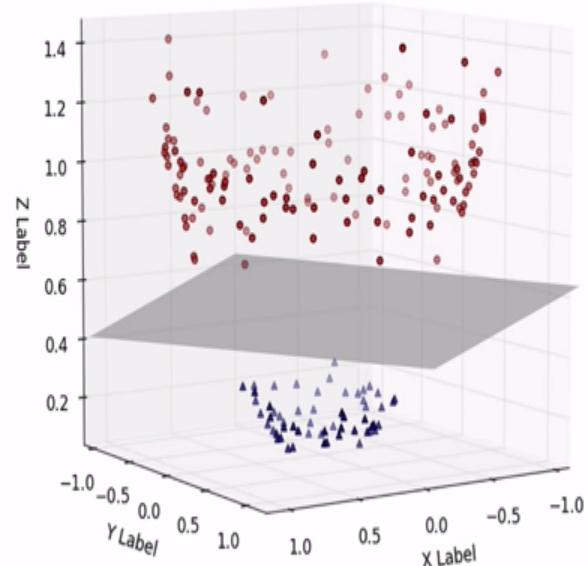
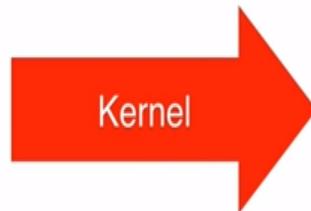
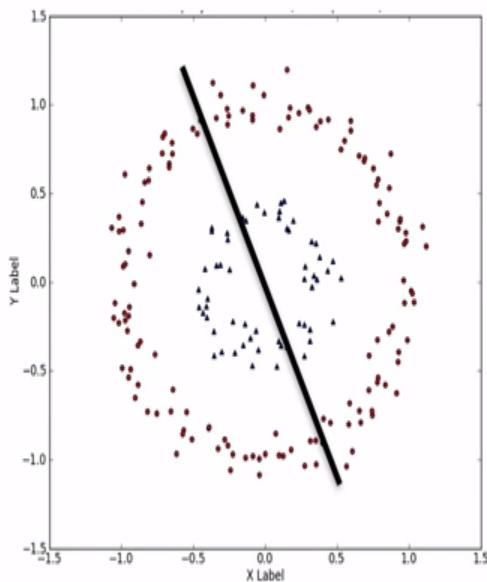
Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

Kernel Trick: Visual



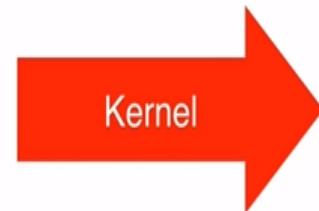
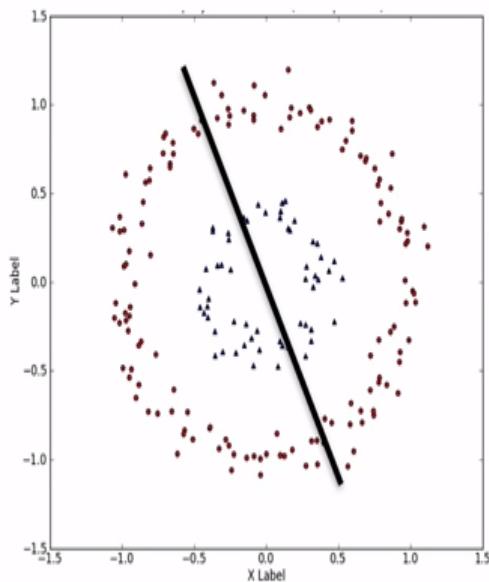
Higher Dimensions

C Parameter

Multiple Classes

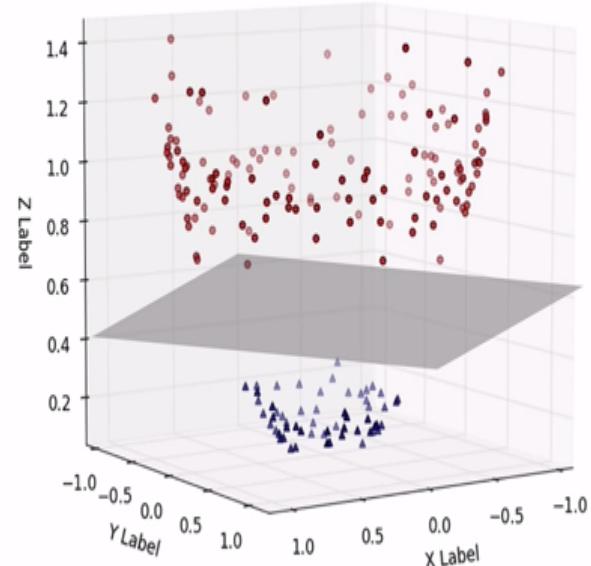
Kernel Trick

Kernel Trick: Visual



Kernel Options

- Linear
- Radial Basis Function
- Polynomial
- Sigmoid



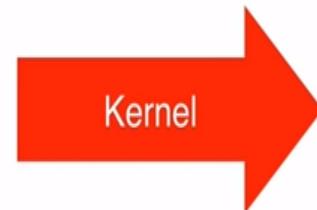
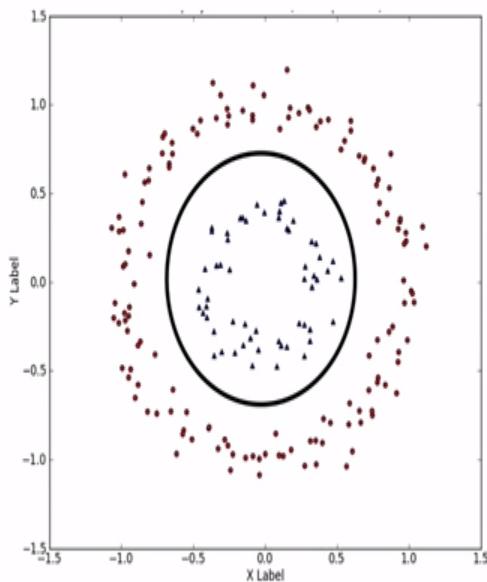
Higher Dimensions

C Parameter

Multiple Classes

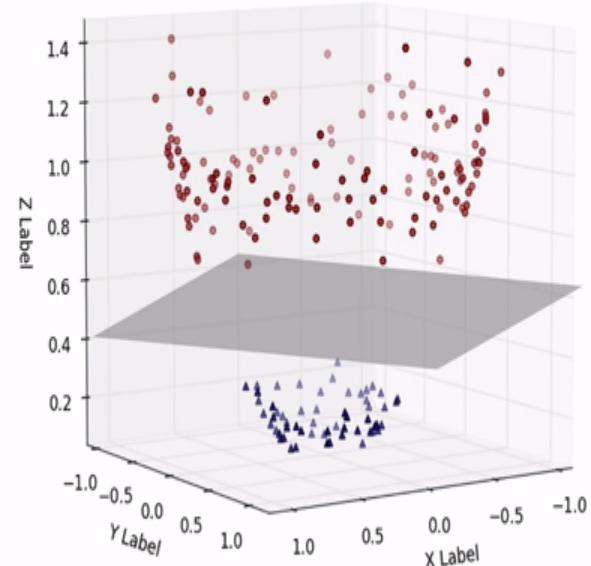
Kernel Trick

Kernel Trick: Visual



Kernel Options

- Linear
- Radial Basis Function
- Polynomial
- Sigmoid



Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

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)
```

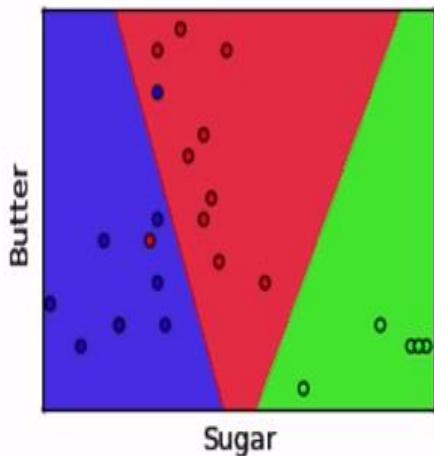
Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

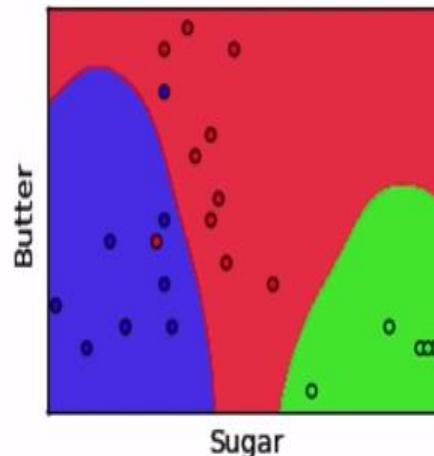
Kernel Trick: Comparison



Kernel: Linear
C: 1

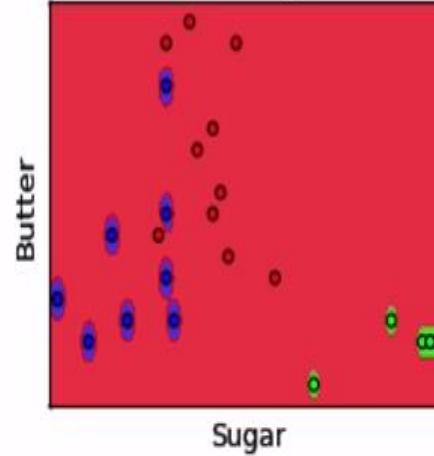
- Muffin
- Cupcake
- Scone

Higher Dimensions



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

C Parameter

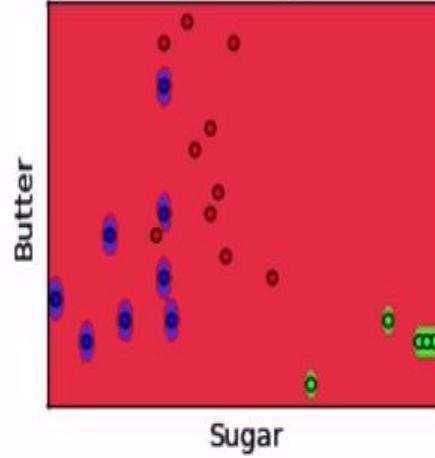
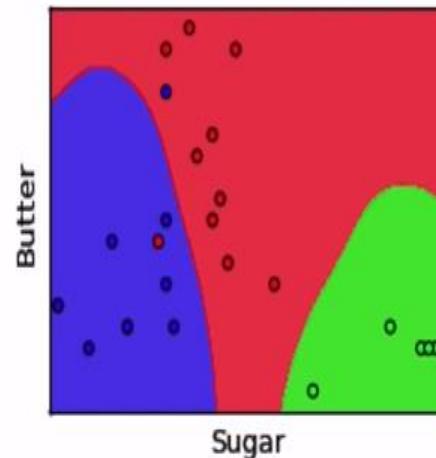
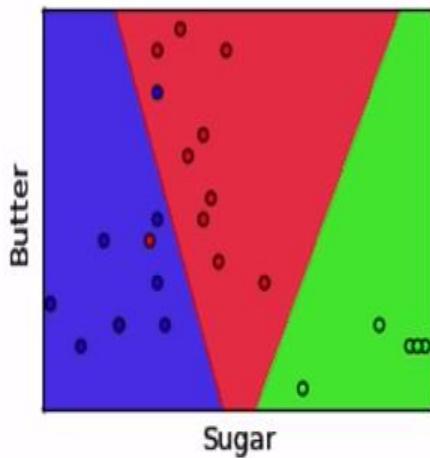


Kernel: RBF
C: 1
Gamma: 2^1

Multiple Classes

Kernel Trick

Kernel Trick: Comparison



Higher Dimensions

C Parameter

Multiple Classes

Kernel Trick

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

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

Classification Techniques

- **SVM is one of many classification techniques**
 - Logistic Regression
 - K Nearest Neighbors
 - Decision Trees
 - Naïve Bayes
 - Neural Networks
- **Advice: try multiple techniques on your data set**

A Cautionary Example



Image classification of tanks. Autofire when an enemy tank is spotted.

Input data: Photos of own and enemy tanks.

Worked really good with the training set used.

In reality it failed completely.

Reason: All enemy tank photos taken in the morning. All own tanks in dawn.

The classifier could recognize dusk from dawn!!!!

References

<http://www.support-vector.net/>

AN INTRODUCTION TO SUPPORT VECTOR MACHINES

(and other kernel-based learning methods)

N. Cristianini and J. Shawe-Taylor

Cambridge University Press

2000 ISBN: 0 521 78019 5

Support Vector Machines

A Visual Explanation + Sample Python Code

https://github.com/MrinalGitHub/GyanVriksh_Course