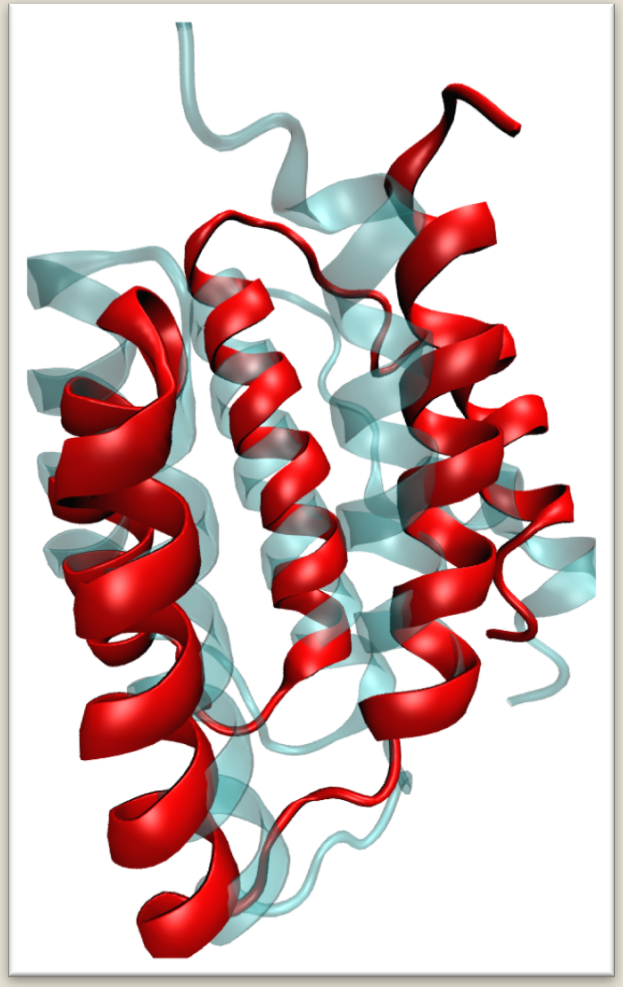
INTRO TO MACHINE LEARNING

Jennifer Van, ML Engineer with Center for Machine Learning

About Me

- **Current**:
  - Machine learning & front-end engineer 🦄
- ***Former***:
  - *Computational biology researcher* ➔
  - *Bartender* 🍻
  - *Bus driver* 🚌
  - *Artist* 🎨



Please note!

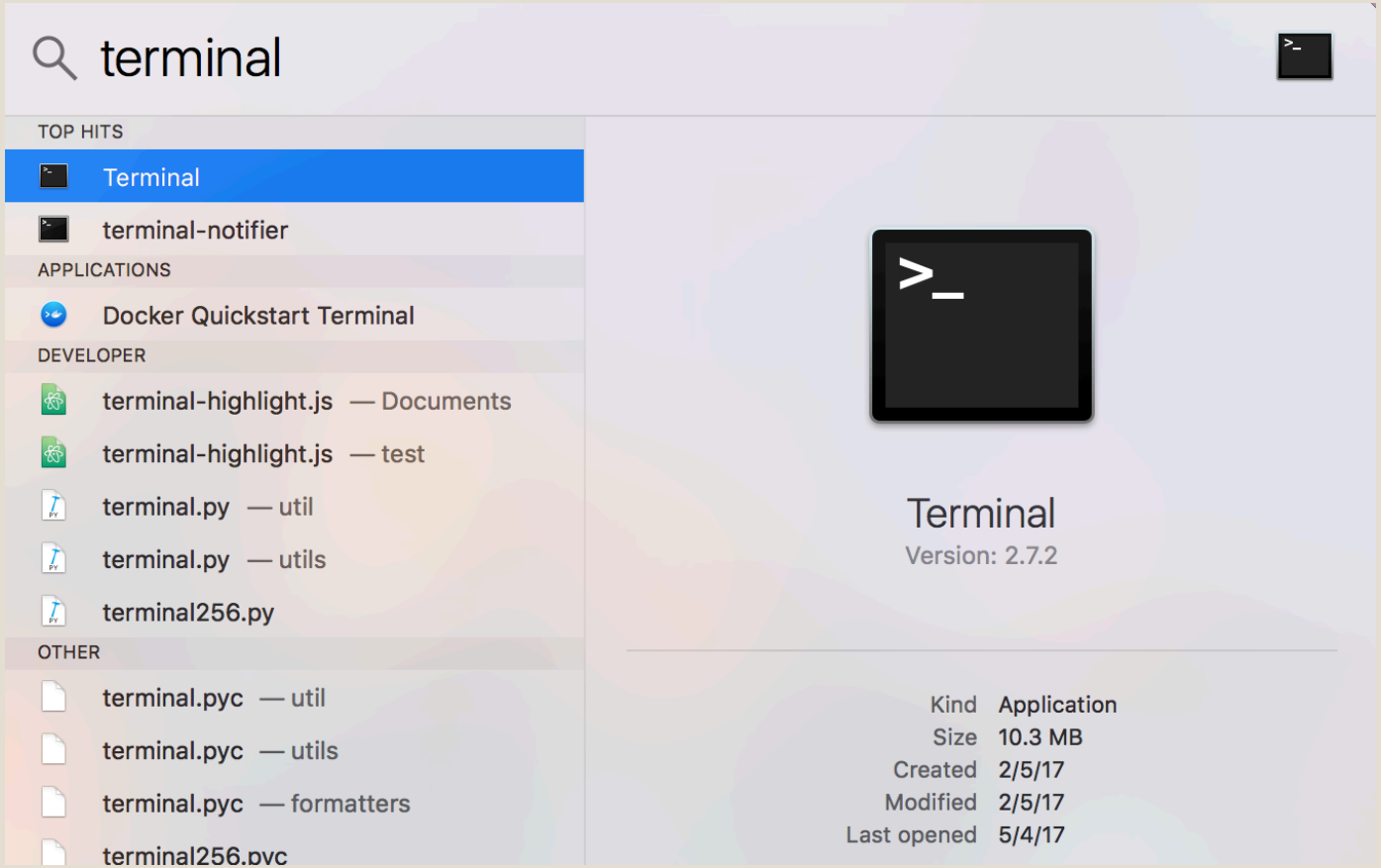
- Go at ***your own*** pace!
- This is **not** comprehensive!
- Try to walk away with the **big picture**.

Get the materials for this session!

TODO:

1. Open your terminal or command prompt

(Example: Search for terminal on Mac)



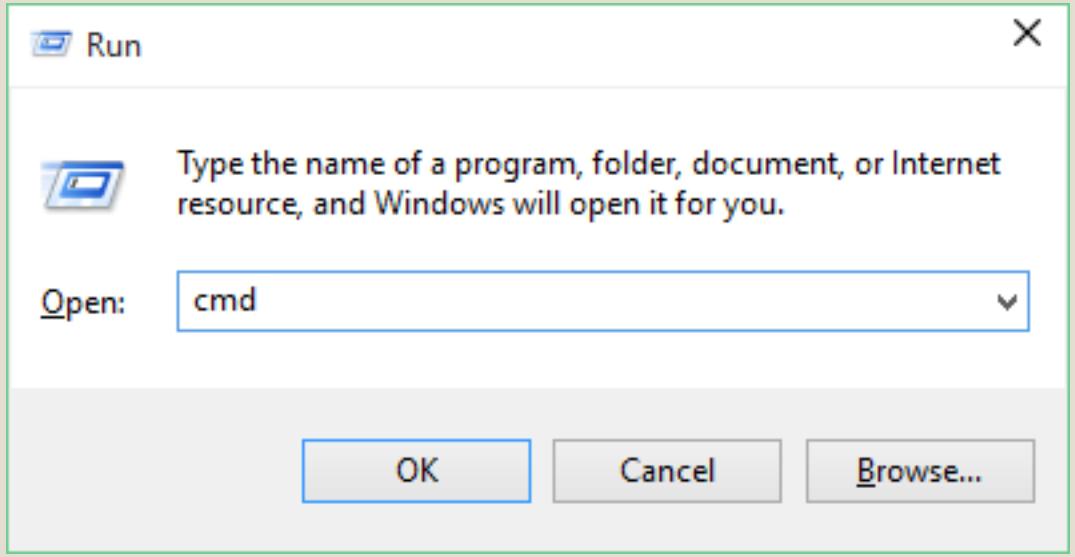
Get the materials for this session!

TODO:

1. Open your terminal or command prompt

(Example: search for command prompt on Windows)

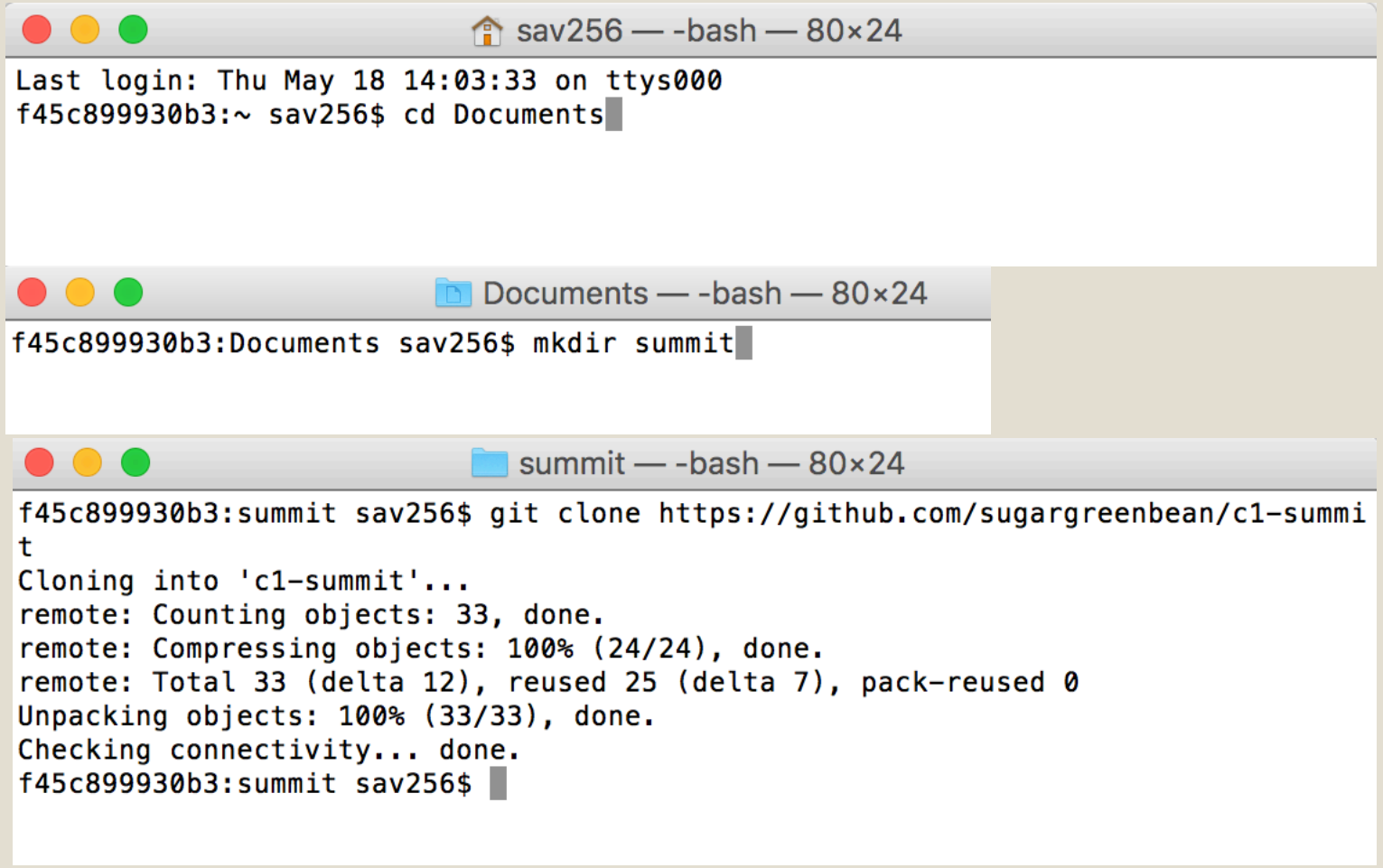
1. Press Windows+R
2. Type in ‘cmd’
3. Click ‘Ok’



TODO:

2. Clone the following repo wherever you want. I created a folder called ‘summit’ in my Documents folder and then cloned with the following:

git clone https://github.com/sugargreenbean/c1-summit



What *is* machine learning?

…"*witchcraft* that: researches fundamental principles from data (potions) and develops magical algorithms (spells to cast)

-Pascal Vincent, 2016

A machine is said to be ***learning*** if it improves with:

Experience * Task = Performance

- Each experience ***E***
- On specific tasks ***T***
- With specific performance ***P***

-Tom Mitchell, 1998

Two Common **Tasks** of ML

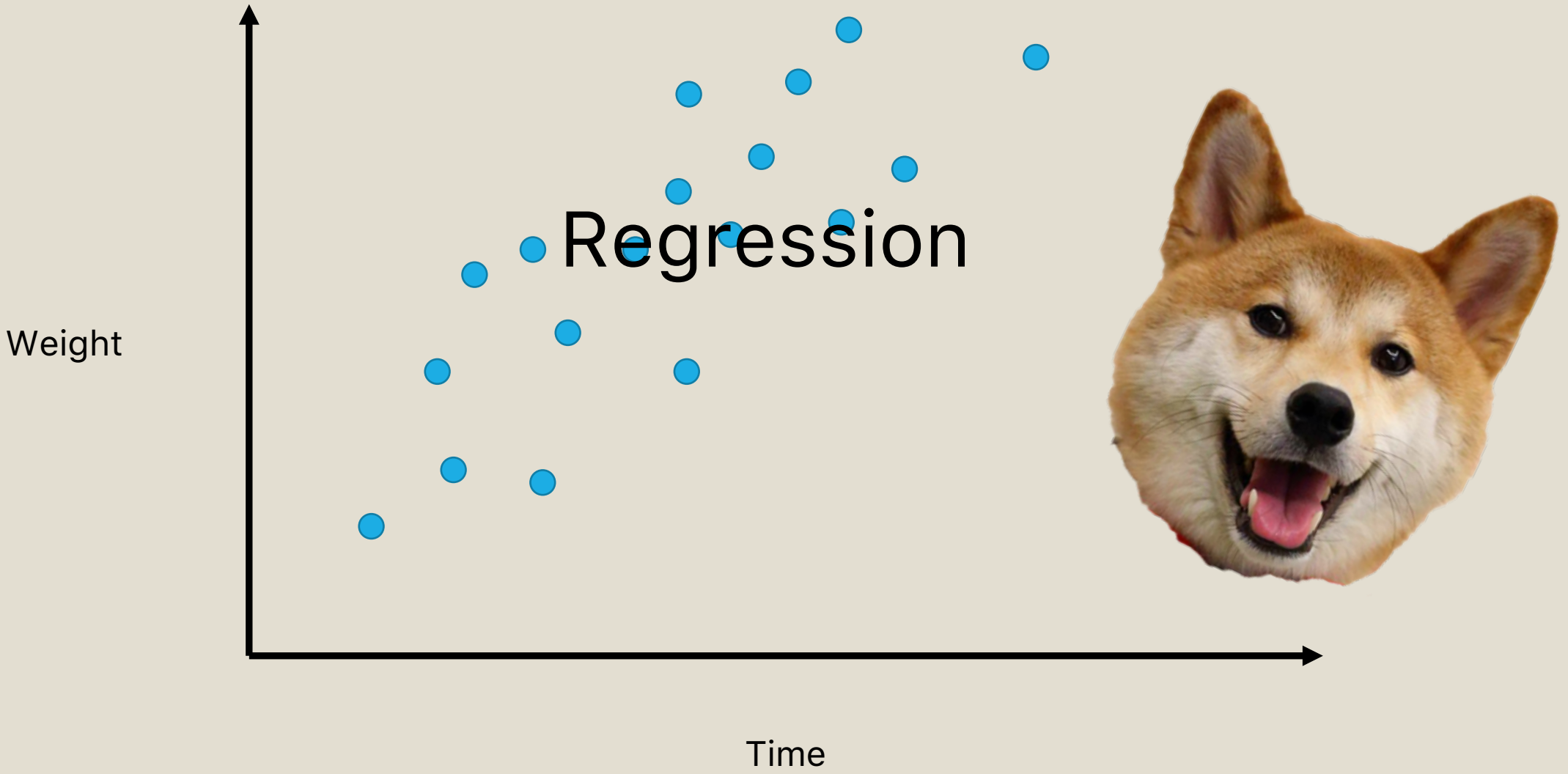
Classify these animals as either:

Classification

Dog

Not dog

Doggo Weight Over Time



SUPERVISED LEARNING

[ , dog]

[ , not a dog]

UNSUPERVISED LEARNING

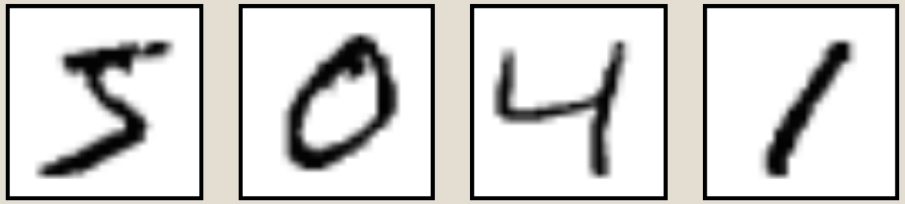
Extreme!!!!
NO LABELS!

Putting it all together…

	Input	Output
Supervised regression (ex: linear regression)	Labeled	Continuous
Unsupervised classification (ex: clustering)	Unlabeled	Discrete
Supervised classification (ex: support vector machine)	Labeled	Discrete

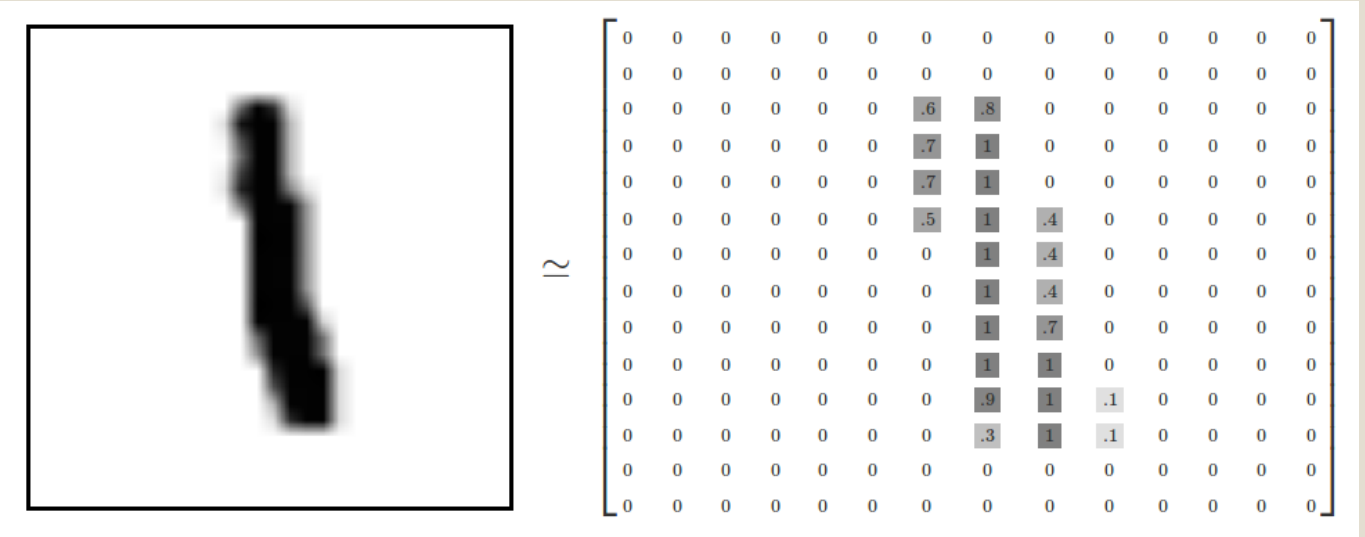
Let’s start!

MNIST Dataset: the ’Hello World’ of ML

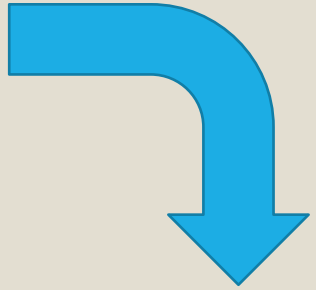


1. K-nearest neighbors (K-NN)
2. Support vector machine (SVM)

MNIST Dataset: How can we use images as input to a computer program?



Flatten matrix into vector



Source: https://rstudio.github.io/tensorflow/tutorial_mnist_beginners.html

[0, 0, 0, 0, 0, 0.6, 1, 0.4, …., 0, 0, 0, 0]

Length: 784 values

Getting started

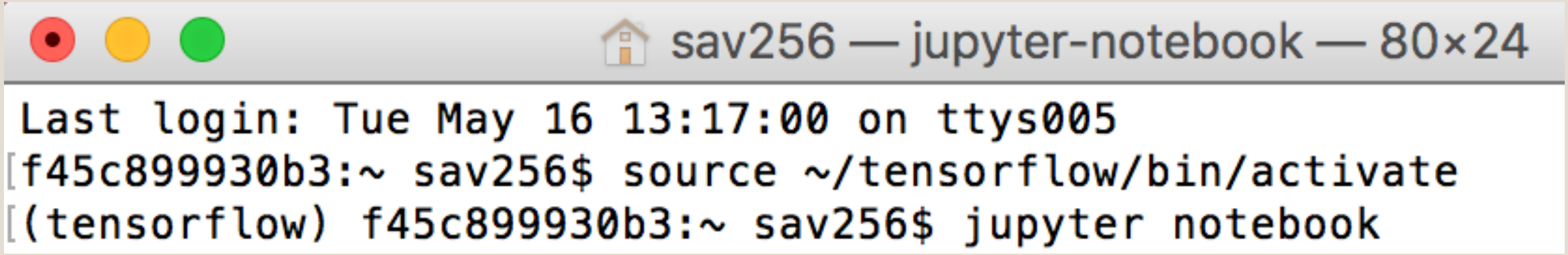
TODO: (On Mac only, skip this step if on Windows) Activate your virtual environment by typing the following into your terminal:

source ~/tensorflow/bin/activate

TODO: (On Mac and Windows) Start jupyter notebook by typing the following into your terminal:

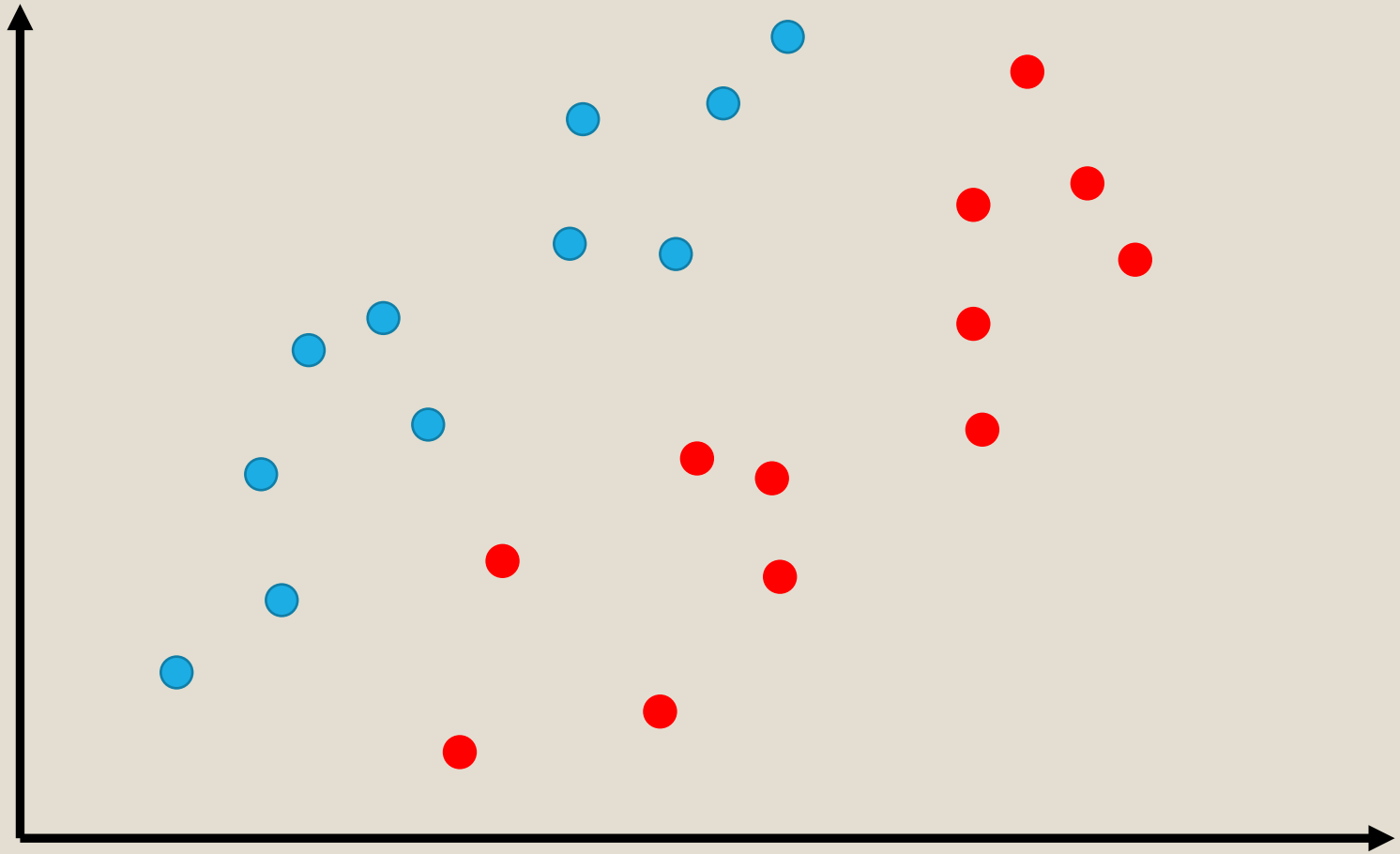
jupyter notebook

What your terminal/command prompt should look like!

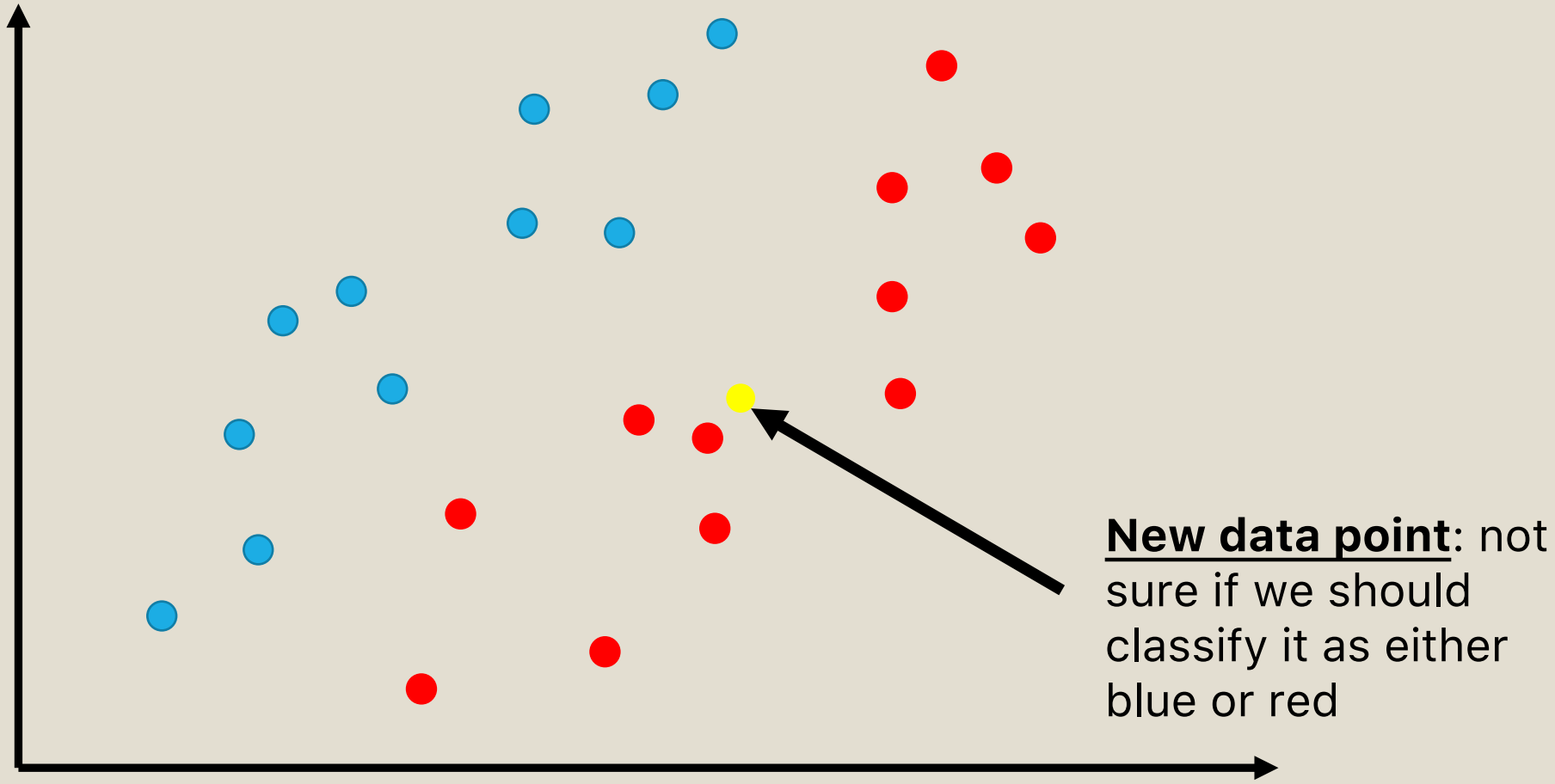


k-nearest neighbors

Two types of dots: **blue** or **red**



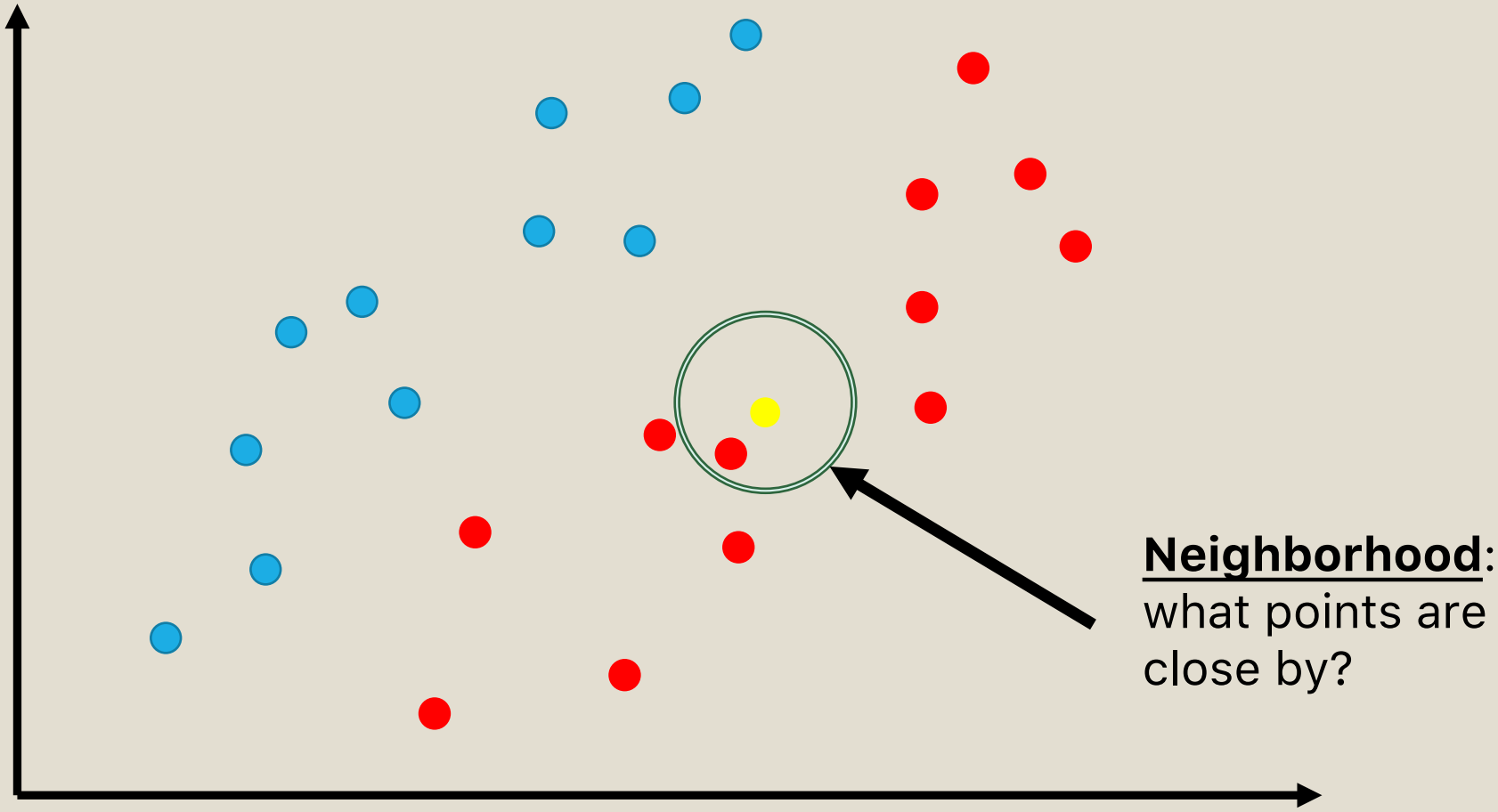
k-nearest neighbors



k-nearest neighbors

k=1

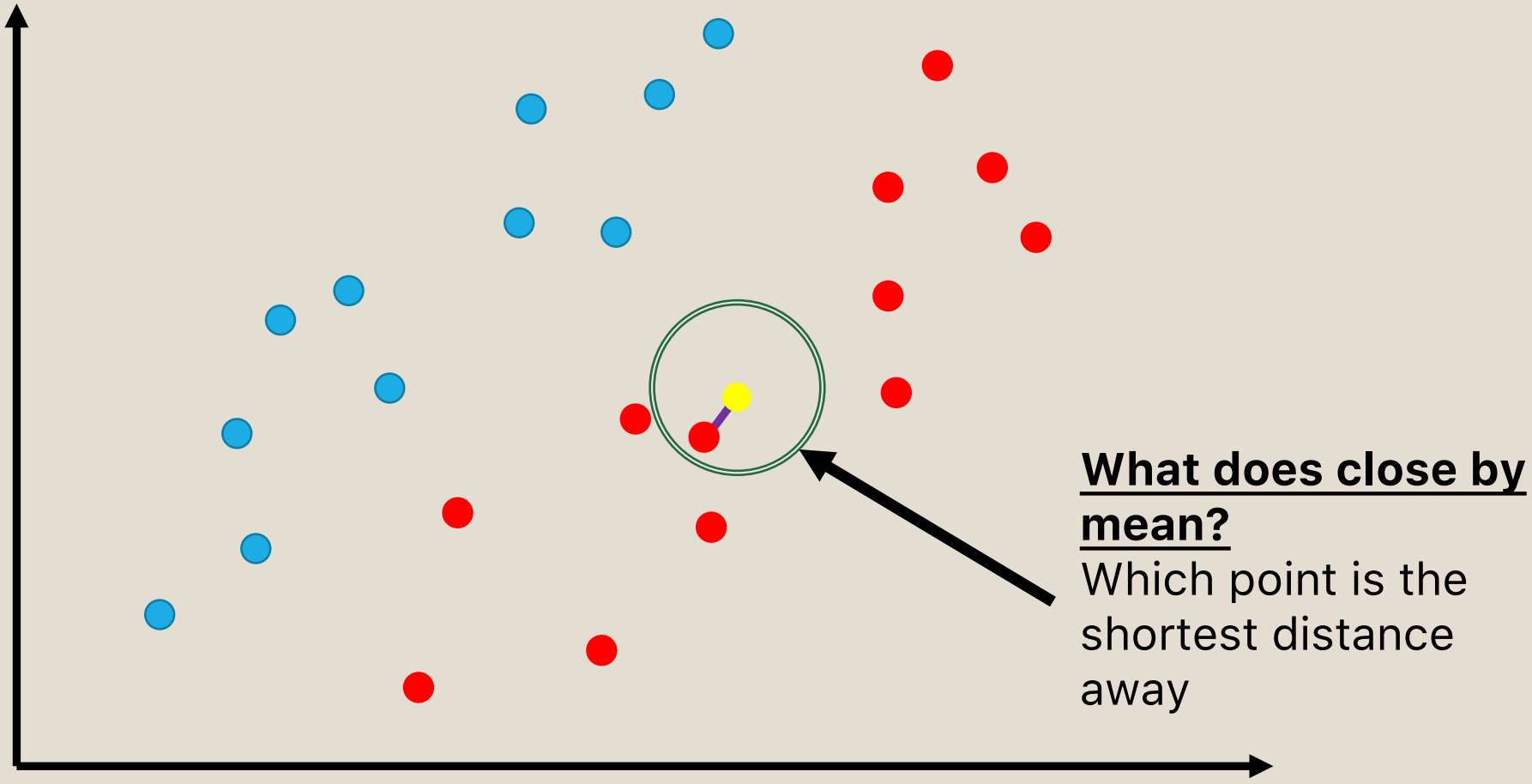
Find the closest neighbor



k-nearest neighbors

k=1

Find the closest neighbor



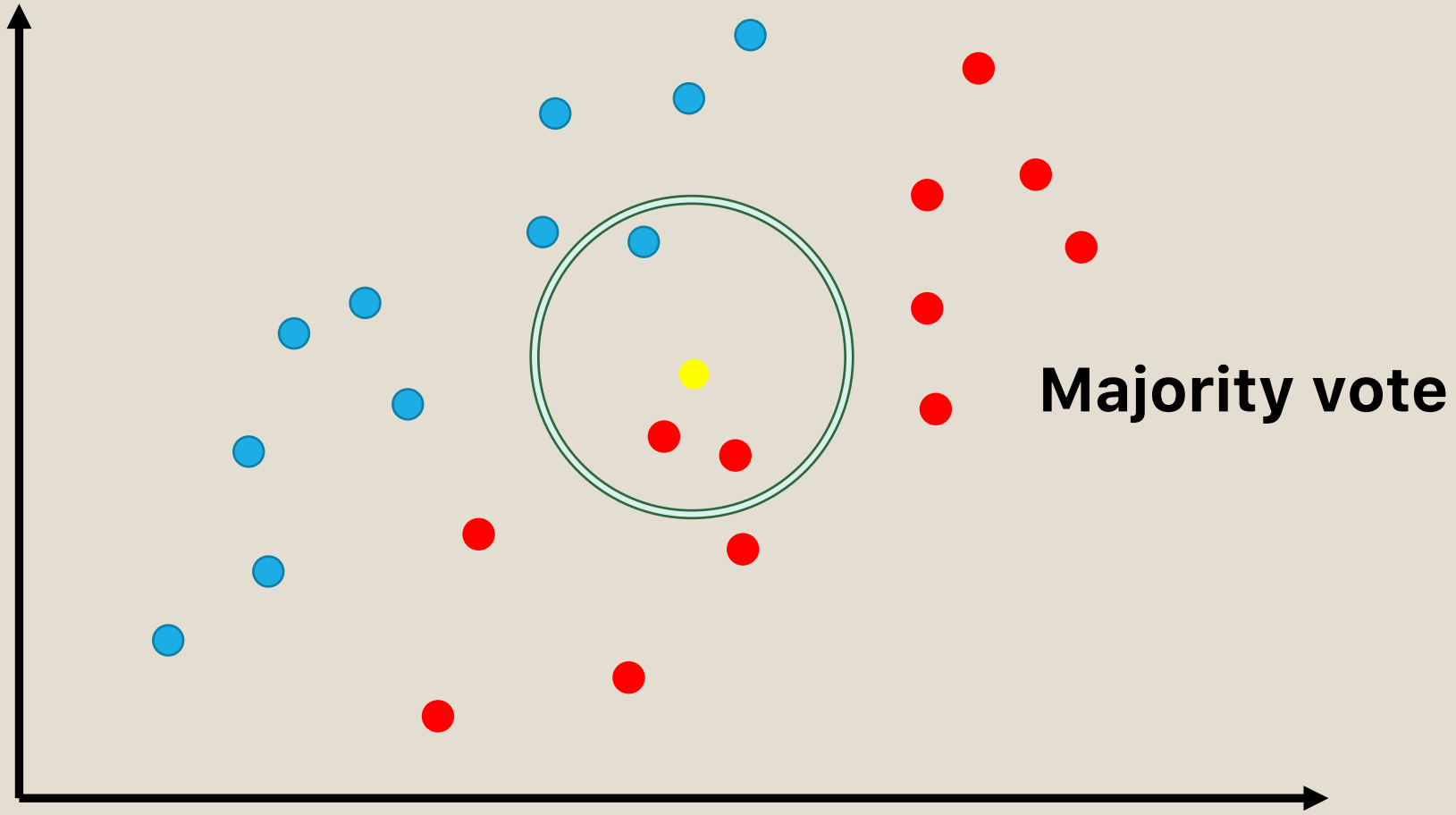
What does close by mean?

Which point is the shortest distance away

k-nearest neighbors

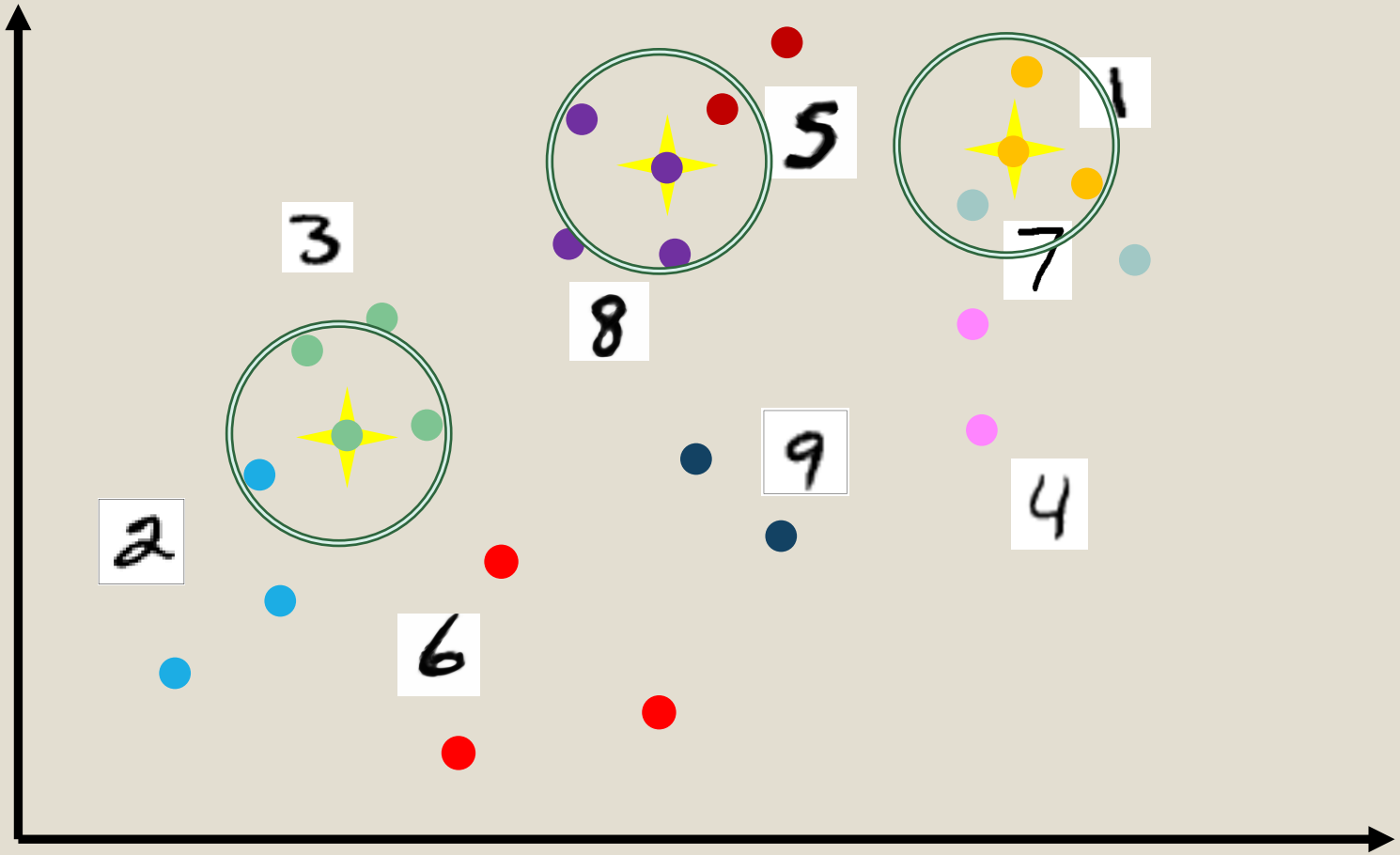
k=3

Find the 3 closest neighbors



k-nearest neighbors

= new data point that has not been classified yet



Getting started

Your browser should now have this page popped up, or localhost:8888. If not, navigate to it in your favorite browser!

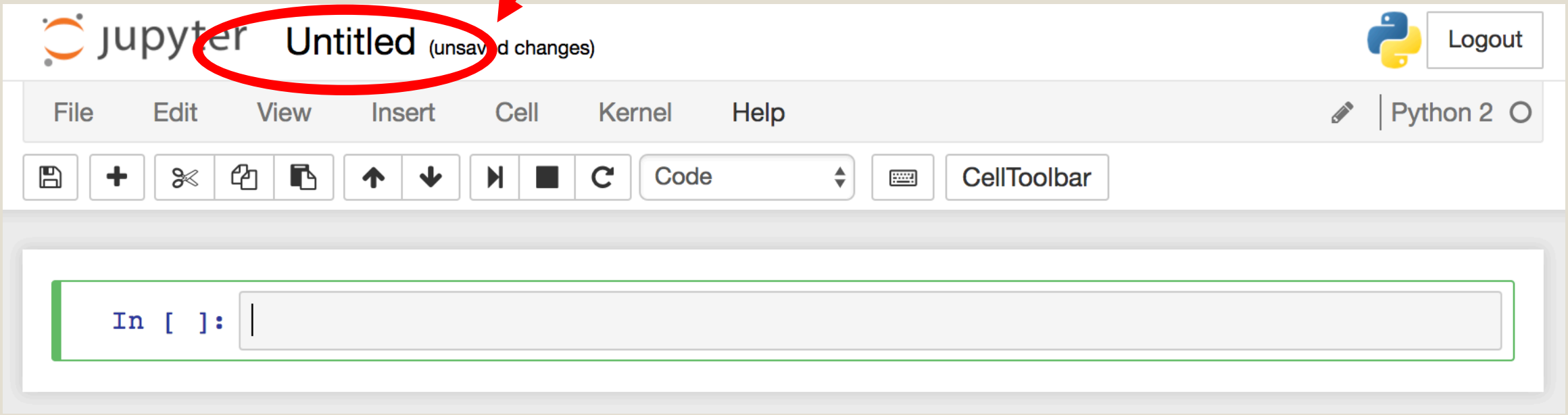
Jupyter Notebook 101

WHY IT'S COOL: Notebooks are a way for you to play around with Python projects and immediately see their result.

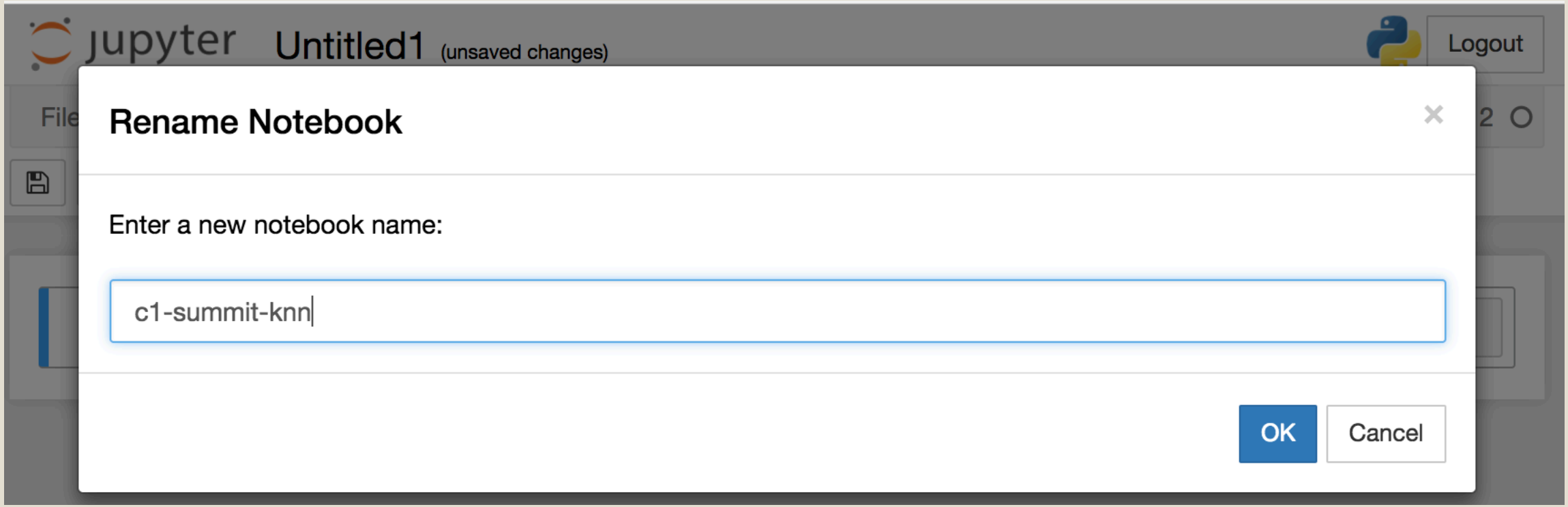
Click to create a new notebook!

Jupyter Notebook 101

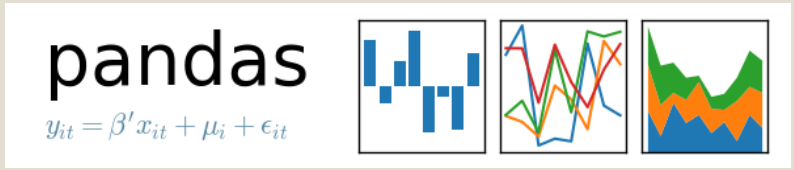
Click here to rename your notebook!



Jupyter Notebook 101



BASIC ML libraries



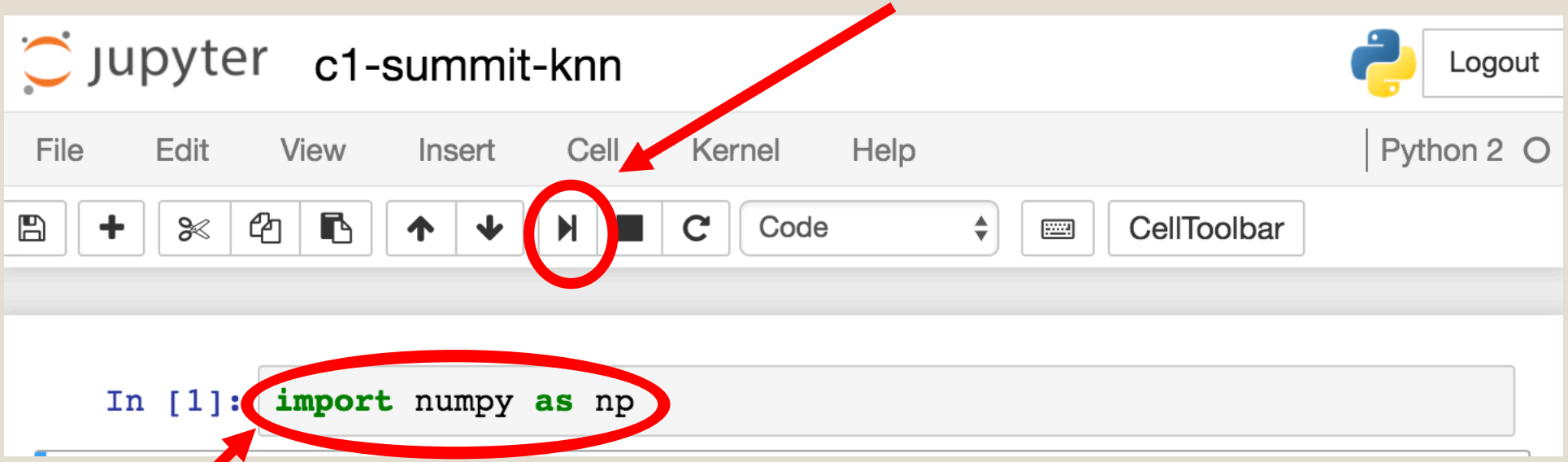
numpy and ***pandas*** let you play with numbers nicely

matplotlib plots visual graphs

sklearn is a general ML lib

Jupyter Notebook 101: How To Import

2. When you're done writing all your imports, click this to run the cell and import all that dankness



1. Write your imports here!

Now *you* try to import the following!

TODO IMPORT 1: ***numpy*** **Example**

```
In [1]: import numpy as np
```

TODO IMPORT 2: ***pandas***

TODO IMPORT 3: ***matplotlib.pyplot***

TODO IMPORT 4: train_test_split from ***sklearn.model_selection***

TODO IMPORT 5: ***time***

TODO IMPORT 4: collections from ***Counter***

TODO COMMAND: keep graphs inline instead of popping out into a new window

Completed imported libraries

jupyter c1-summit-knn

Logout

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import time
from collections import Counter
%matplotlib inline
```

Load and understand the data

TODO: Use panda's read_csv to read train.csv into a dataframe, save training data in a variable called 'labeled_images'

TODO: Type 'labeled_images' and press run cell to see what the input data looks like

TODO: Change this path to point to where your c1-summit repo is!

```
In [87]: # Read in data
         labeled_images = pd.read_csv('/Users/minhvan/Documents/c1-summit/data/train.csv')
         labeled_images
```

WOW! In Python, we don't have to declare types with our variables like in Java (Java would have required us to say something like, String[] labeled_images or int age. Python just knows!

Analysis: Input Data

42000 ***rows*** × 785 ***columns***

```
In [34]: labeled_images = pd.read_csv('train.csv')

In [35]: labeled_images
```

784 columns=784 pixel values

Out[35]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	p
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
6	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
10	8	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

42000 rows = 42000 images

1 column for labels of what number digit is in image, 0-9

Preprocessing of Input

42000 ***images***...That's a bit of overkill for us. I will reduce this down to 5,000 images for a reasonable computation time.

I *also want to* separate ***images*** and ***labels*** using indexing provided by panda's iloc

```
In [36]: images = labeled_images.iloc[0:5000,1:]
```

Get first 5000 ***images*** only!

Get only ***columns*** 2-785,
so only keep the pixel values

Your turn!

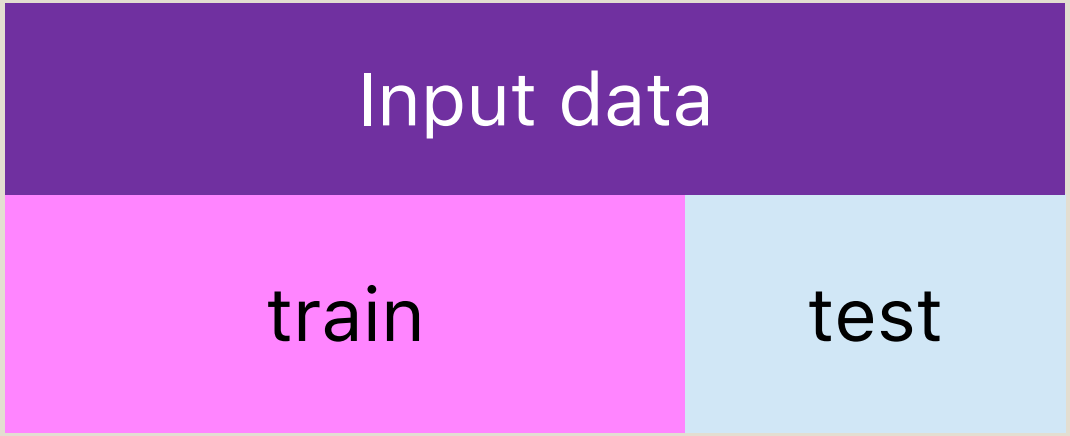
We just got the first 5000 ***images*** with their ***pixel values*** indexed from the input data.

TODO: get the first 5000 ***images*** indexed from the input data with their ***labels*** only.

Answer!

```
In [ ]: labels = labeled_images.iloc[0:5000,:1]
```

Now, we need to split our input data into train and test data.



Training: used to train your model

Test: estimate how well your model has been trained

Test/train split

Pixel values for training images

Pixel values for test images

Digit in train images

```
train_images, test_images, train_labels,
test_labels = train_test_split(images,
                               test_size=0.8, random_state=0)
```

Digit in test images

FYI: In Python, we can return a list or a tuple (look at the left hand side where we return 4 different things) and unpack them! Neat! It's called tuple unpacking.

Test/train split

```
train_images, test_images, tr    s,
test_labels = train_test_split(images,
labels, train_size=0.8, random_state=0)
```

Pixel values for all images

Labels for all images

How much of input arrays should be train (80% in this case) and test (20% in this case)

Choose whether the train and test are split randomly (1) or just in sequential order (0)

Test/train split

TODO: Type the following into a cell and press run:
train_images, test_images, train_labels, test_labels = train_test_split(images, labels, train_size=0.8, random_state=0)

In [4]:
```
# Separate input into train and test
train_images, test_images,train_labels, test_labels = train_test_split(images, labels, train_size=0.8, random_state=0)
```

Test/train split

TODO: Run the following in an individual cell, press run, and see what they look like!

- *train_images*
- *test_images*
- *train_labels*
- *test_labels*

```
In [82]: train_images
```

```
In [83]: test_images
```

```
In [84]: train_labels
```

```
In [85]: test_labels
```

Convert arrays into numpy arrays

Why? So we can run computations for K-NN on our input arrays with functions from numpy

TODO: convert test_images to numpy array using: as_matrix()

Completed: Convert arrays into numpy arrays

```
In [68]: # Convert pandas array into numpy array so we can run computations on it
         test_images_np_arr = test_images.as_matrix()
```

TODO: Convert train_images into a numpy array

TODO: Convert train_labels into a numpy array

Completed: Convert arrays into numpy arrays

In [69]:
```
# Convert pandas array into numpy array so we can run computations on it
train_images_np_arr = train_images.as_matrix()
```

In [70]:
```
# Convert pandas array into numpy array so we can run computations on it
train_labels_np_arr = train_labels.as_matrix()
```

Set training variables

TODO: Set a variable called ***batch_size*** to equal 50. This means that we will evaluate test input images in batches of 50, so images 0-50 will be evaluated by K-NN, then images 51-100, etc.

TODO: Set a variable to store the length of the ***test_images*** using shape of an array; this value will be used throughout K0NN calculations

TODO: Set an empty array called predictions. We will store our predictions of test image classes here.

Completed: Set training variables

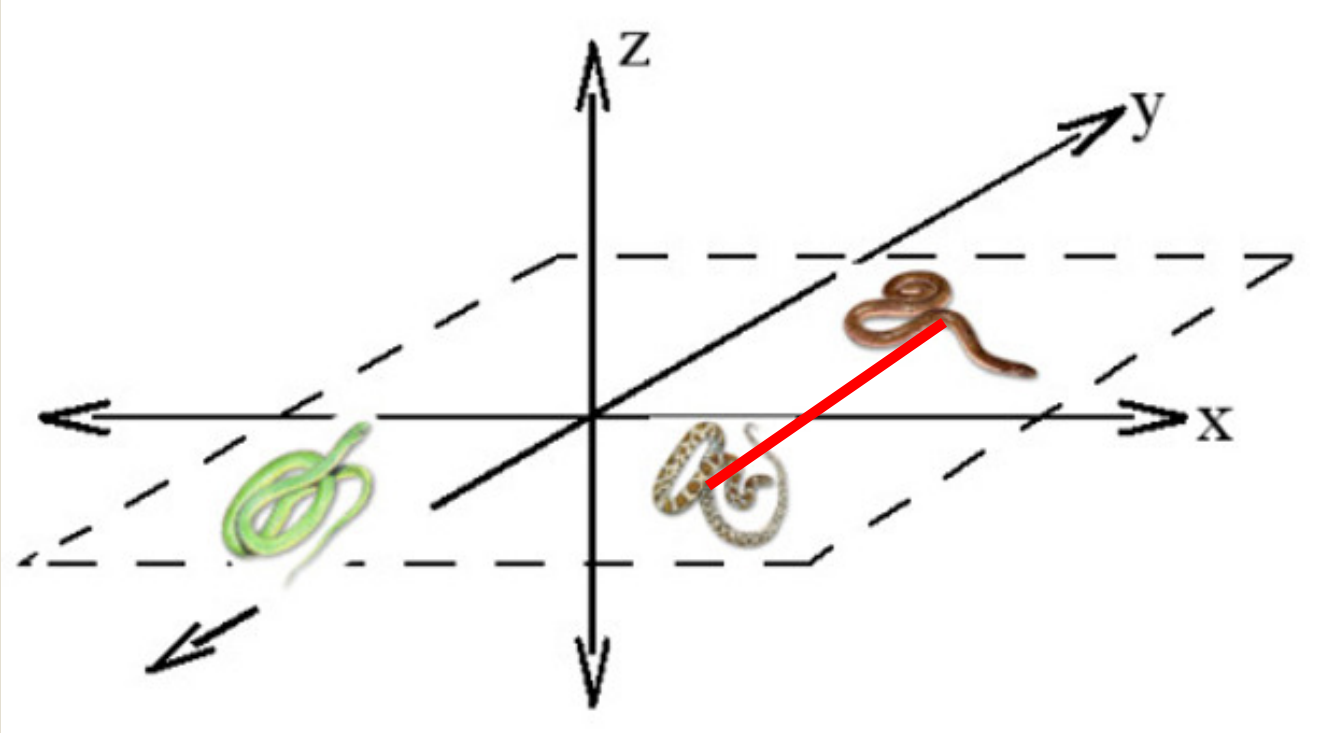
In [71]:
```
# Set batch size
batch_size=50
# Save test_images length
num_test = test_images.shape[0]
# Set empty predictions array for our predictions of test image class
predictions = []
```

K-NN algorithm: High-level

Need to:

1. Compute distances
   A. Compute dot product
   B. Compute square of test and train images
   C. Square root of 1&2 sum
2. Make a prediction based on k-nearest neighbors

1. Compute Distances: High-level



Distance metric: Euclidean

Distance(snake_1, snake_2)

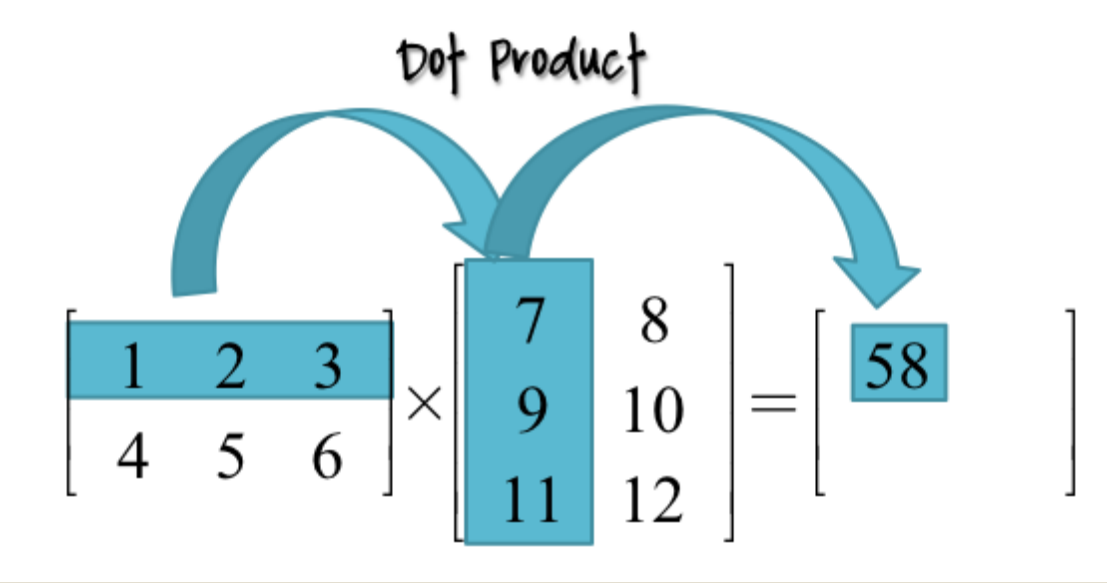
$= ((snake\_1-snake\_2)^2)^{-2}$

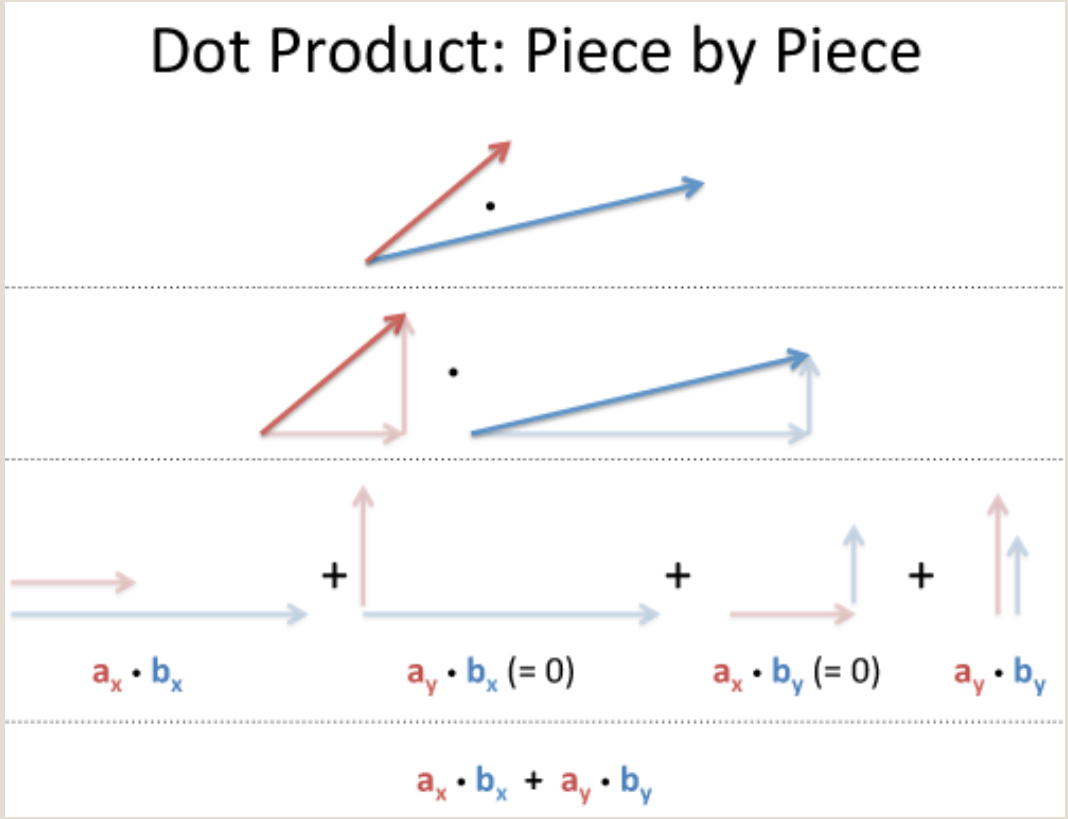
$=(snake\_1*snake\_1 + snake\_2*snake\_2 - 2*snake\_1*snake\_2)^{-2}$

Dot product! Calculate with np.dot(array1, array2)

1. Dot Product: High-level

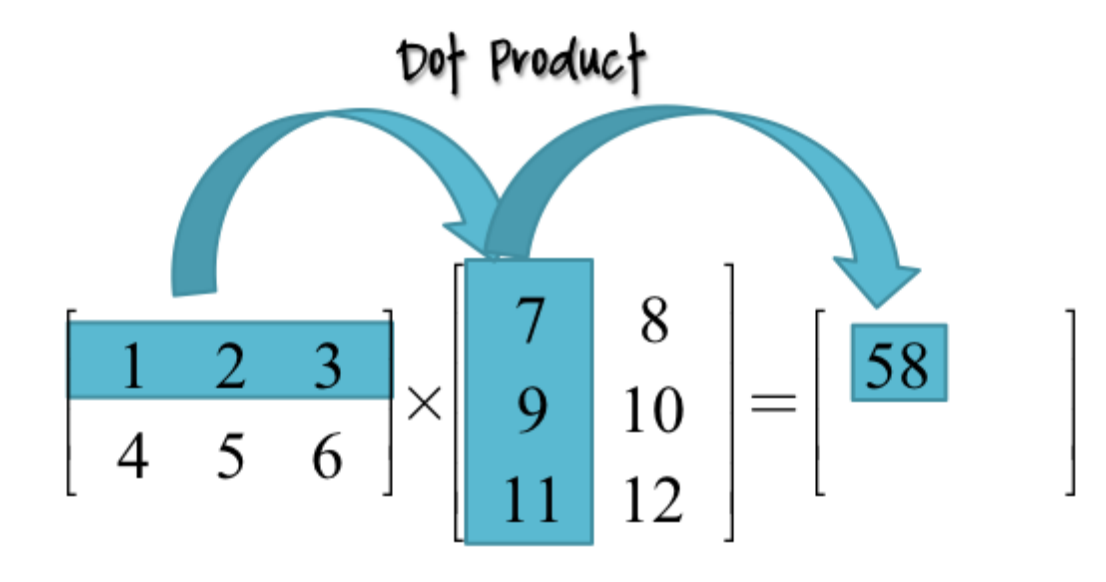
Dot Product: how similar are two vectors/matrices?

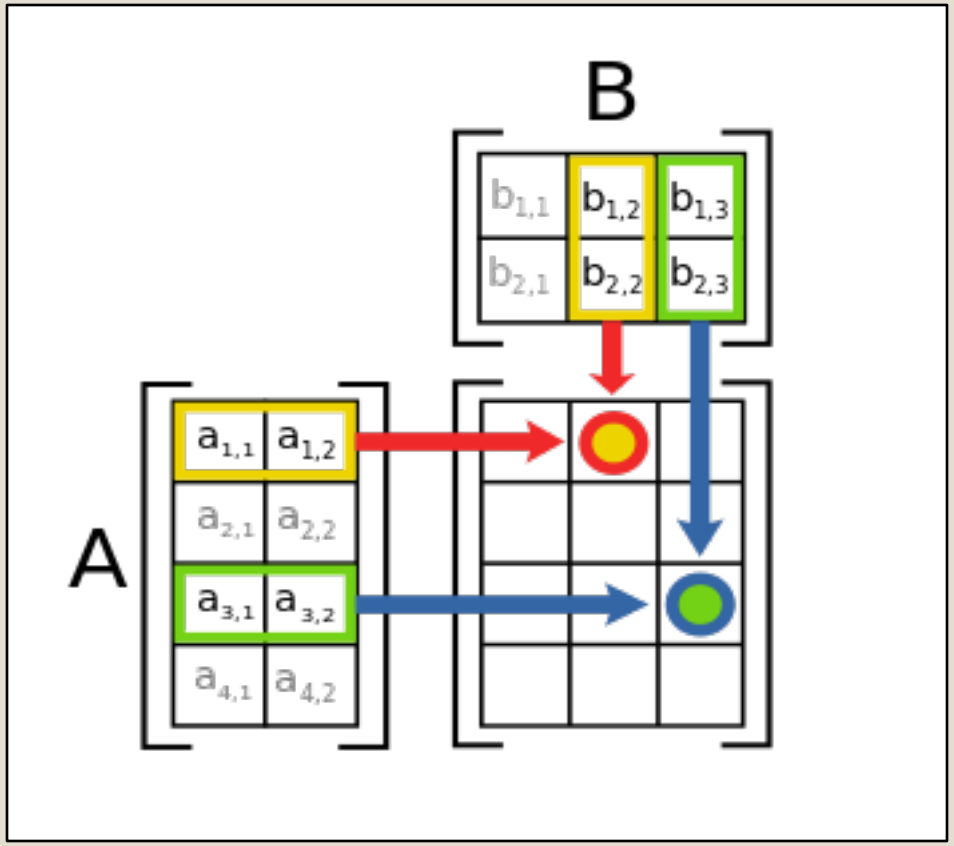




1. Dot Product: Matrix Multiplication

Matrix multiplication: sizes of your arrays matter!



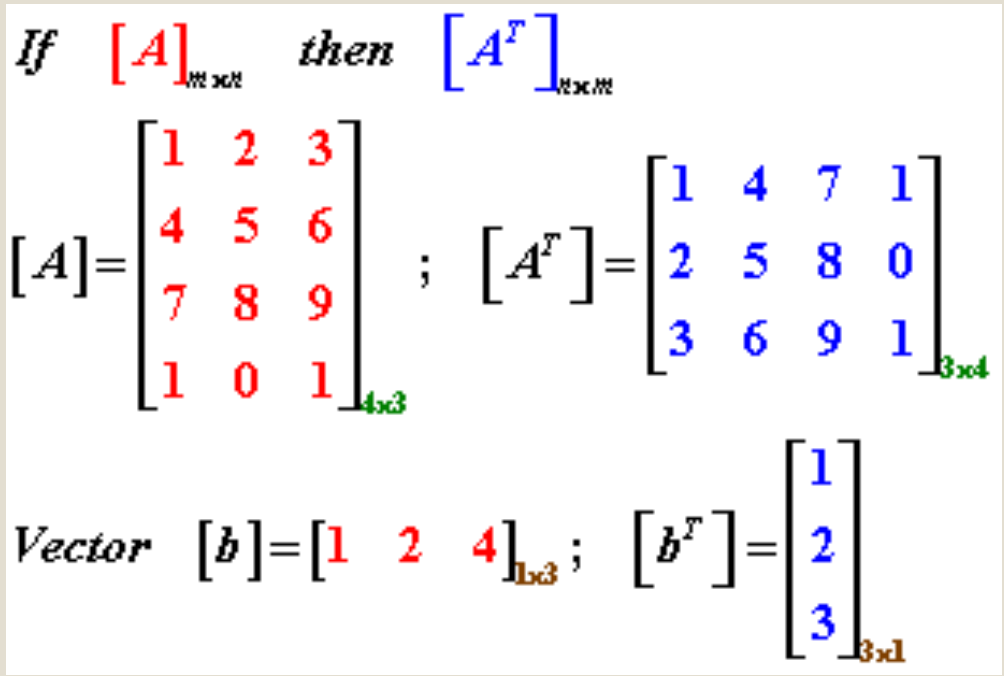


Matrix A size: 2x3 Matrix B size: 3x2

2x3 * 3x2 = 2x2

1. Dot Product: Transpose

Transpose : flip matrix row and column indices



Matrix A size: 2x3 Matrix B size: 2x3

2x3 * 2x3 does NOT work

Matrix A size: 2x3 Matrix B^T size: 3x2

2x3 * 3x2 = 2x2 does work!

1. Dot Product

Dot Product: Find distance between test points and train points

In [79]:
```
for i in range(int(num_test/batch_size)):
    print("Computing batch " + str(i+1) + "/" + str(num_test/batch_size) + "...")
    # begin recording time to calc how long batch processing takes
    tic = time.time()
    # calculate euclidean distance
    test_to_pred = test_images_np_arr[i*batch_size:(i+1)*batch_size]
    dot_prod = np.dot(test_to_pred, train_images_np_arr.T)
```

2. Square of test and train images

In [79]:

```
for i in range(int(num_test/batch_size)):
    print("Computing batch " + str(i+1) + "/" + str(num_test/batch_size) + "...")
    # begin recording time to calc how long batch processing takes
    tic = time.time()
    # calculate euclidean distance
    test_to_pred = test_images_np_arr[i*batch_size:(i+1)*batch_size]
    dot_prod = np.dot(test_to_pred, train_images_np_arr.T)

    sum_square_test = np.square(test_to_pred).sum(axis = 1)
    sum_square_train = np.square(train_images_np_arr).sum(axis = 1)
```

3. Square root of 1&2 sum

```
for i in range(int(num_test/batch_size)):
    print("Computing batch " + str(i+1) + "/" + str(num_test/batch_size) + "...")
    # begin recording time to calc how long batch processing takes
    tic = time.time()
    # calculate euclidean distance
    test_to_pred = test_images_np_arr[i*batch_size:(i+1)*batch_size]
    dot_prod = np.dot(test_to_pred, train_images_np_arr.T)
    sum_square_test = np.square(test_to_pred).sum(axis = 1)
    sum_square_train = np.square(train_images_np_arr).sum(axis = 1)
    dists = np.sqrt(-2 * dot_prod + sum_square_train + np.matrix(sum_square_test).T)
    # get number of distances calculated
    num_distances = dists.shape[0]
```

2. Make a prediction

TODO: Write the following in a cell

```
    # initialize empty numpy array of zeros to hold batch predictions
    label_prediction = np.zeros(num_distances)
    for j in range(num_distances):
        k_closest_y = []
        # get labels from points where distances were calculated
        found_labels = train_labels_np_arr[np.argsort(dists[j,:])].flatten()
        # find closest 3 neighbors
        k_closest_y = found_labels[:3]
        # count the unique neighbors
        c = Counter(k_closest_y)
        # find the most common neighbor and save as prediction
        label_prediction[j] = c.most_common(1)[0][0]
    predictions = predictions + list(label_prediction)
    # end recording time to calc how long batch processing takes
    toc = time.time()
    print("Completed this batch in " + str(toc-tic) + " Secs.")
```

Evaluate accuracy

In [74]:
```
pred_np_arr = np.asarray(predictions)
```

In [76]:
```
correct = 0
counter = 0
for index, row in test_labels.iterrows():
    observed_val = int(row['label'])
    if(observed_val == pred_np_arr[counter]):
        correct += 1
    counter += 1
```

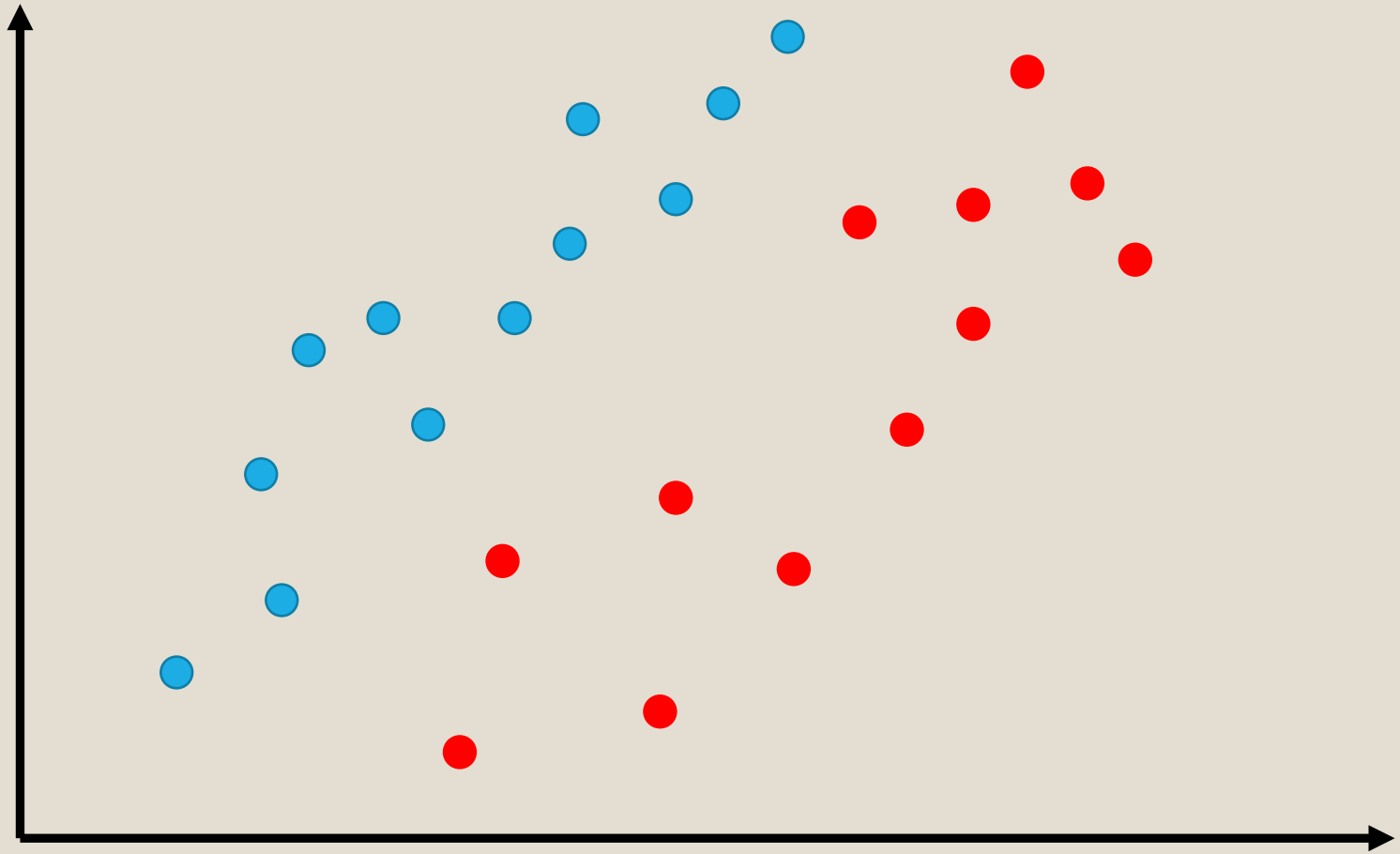
In [77]:
```
float(correct)/1000
```

Out[77]: 0.922

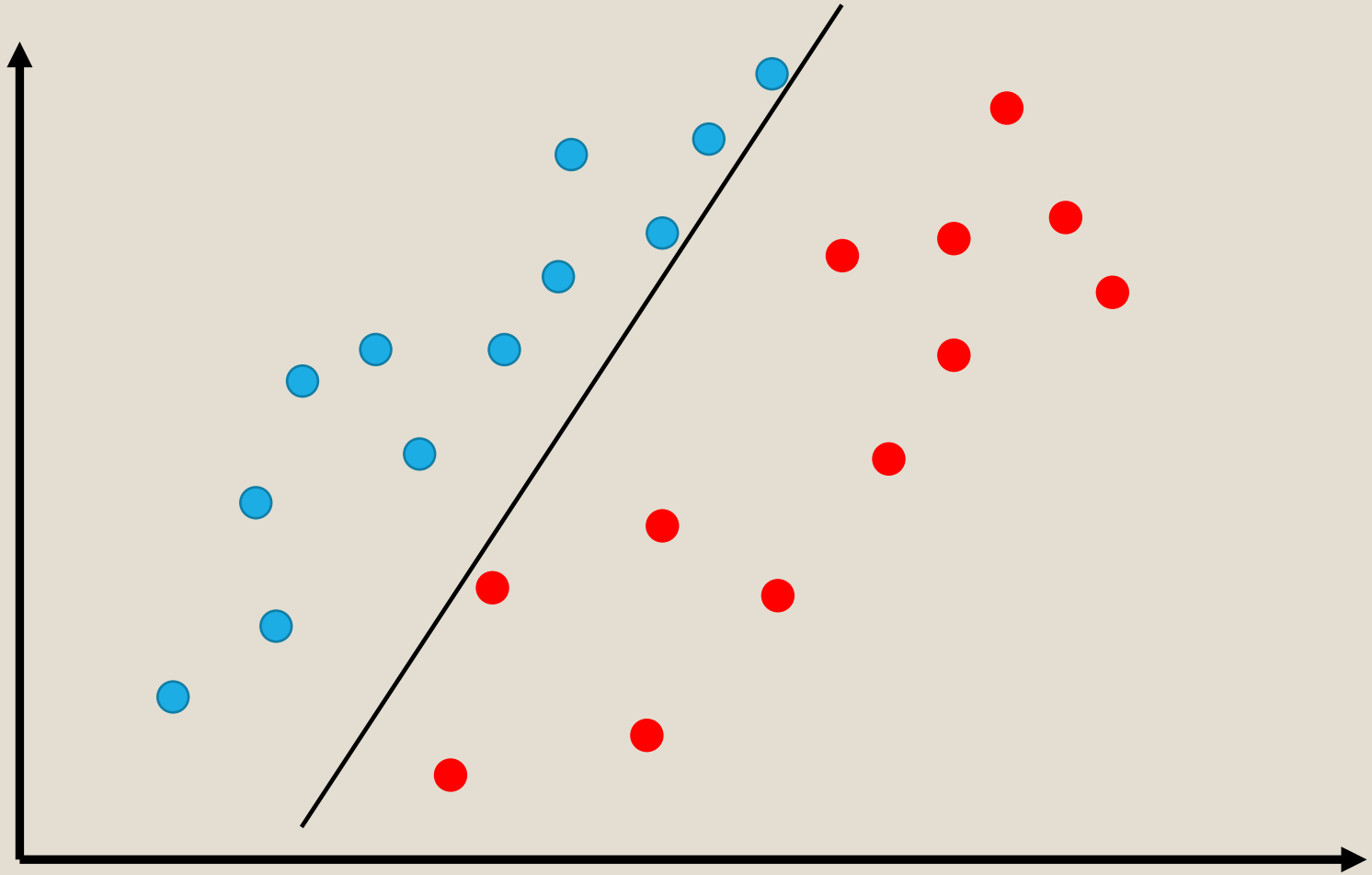
Pt 2: Support Vector Machines

Part 2, part 2, part 2!

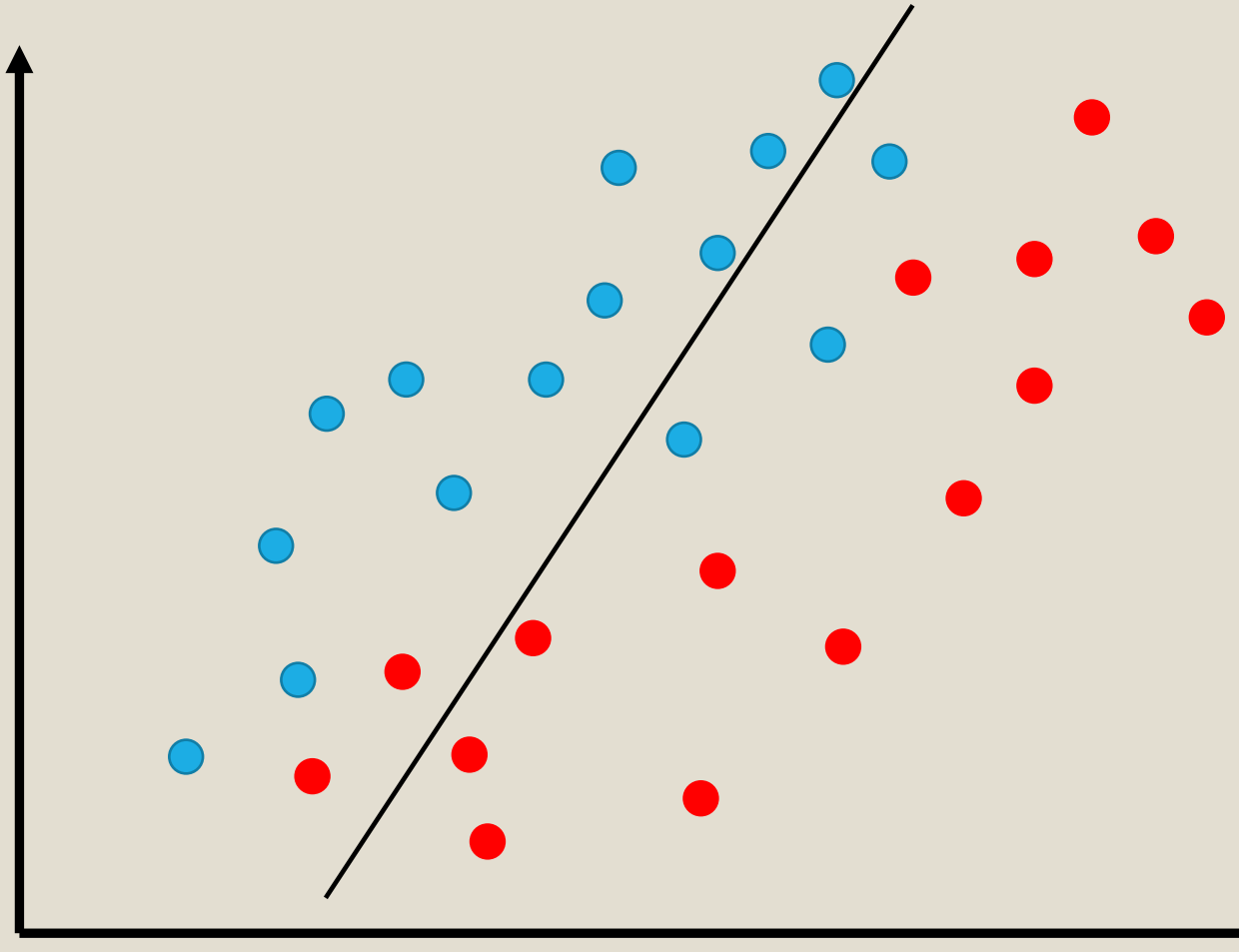
Support Vector Machine



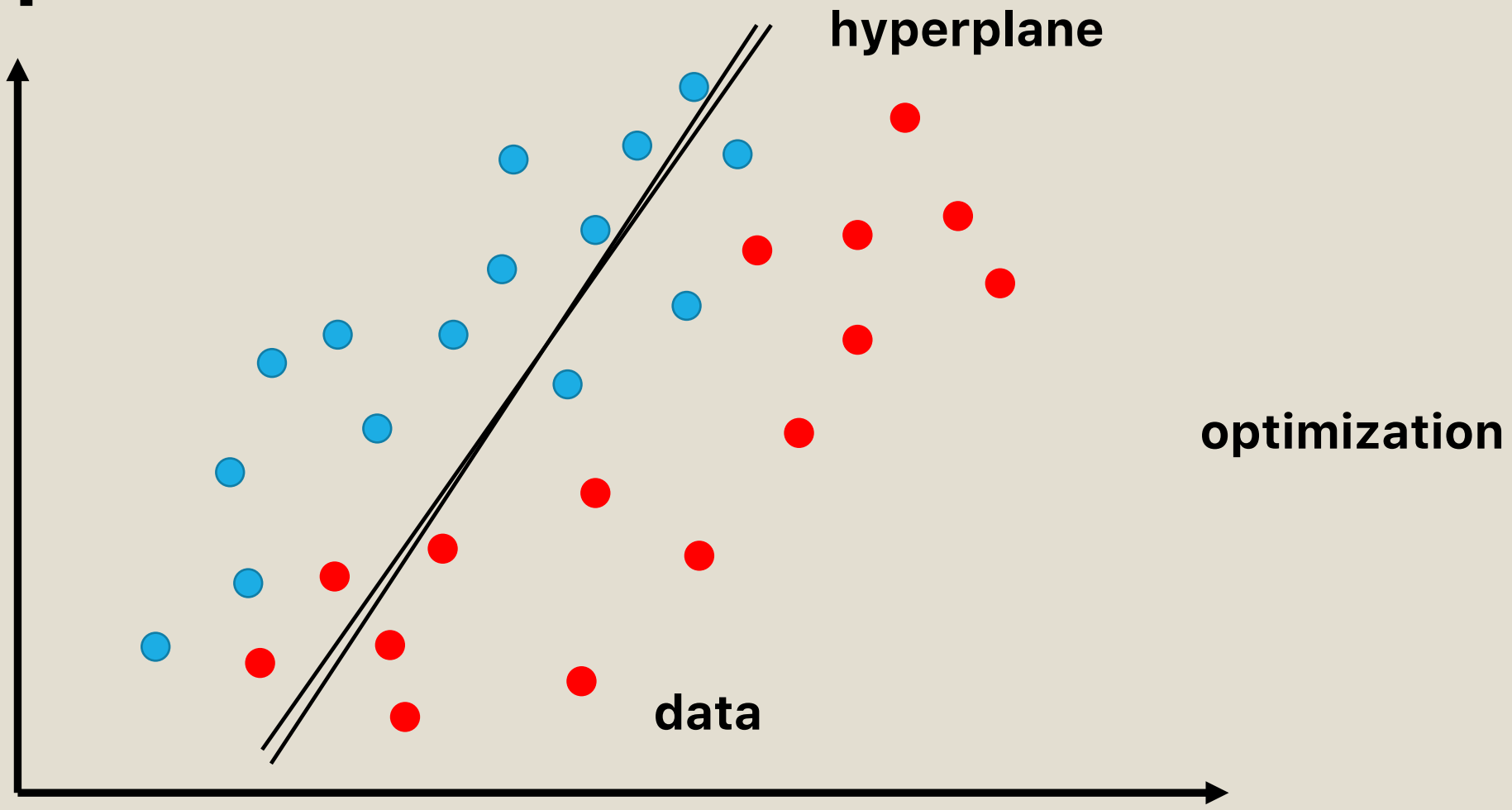
Support Vector Machine



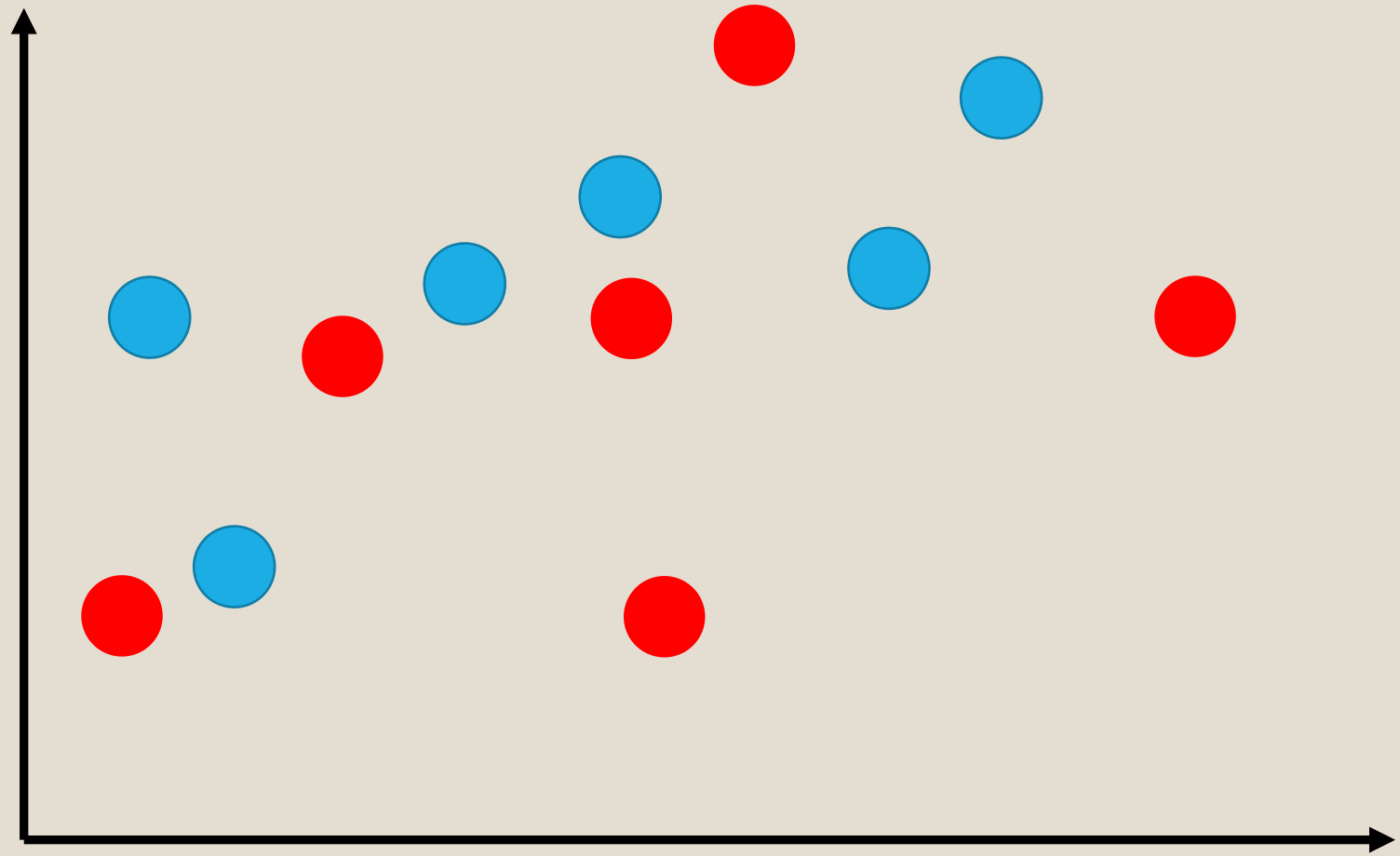
Support Vector Machine



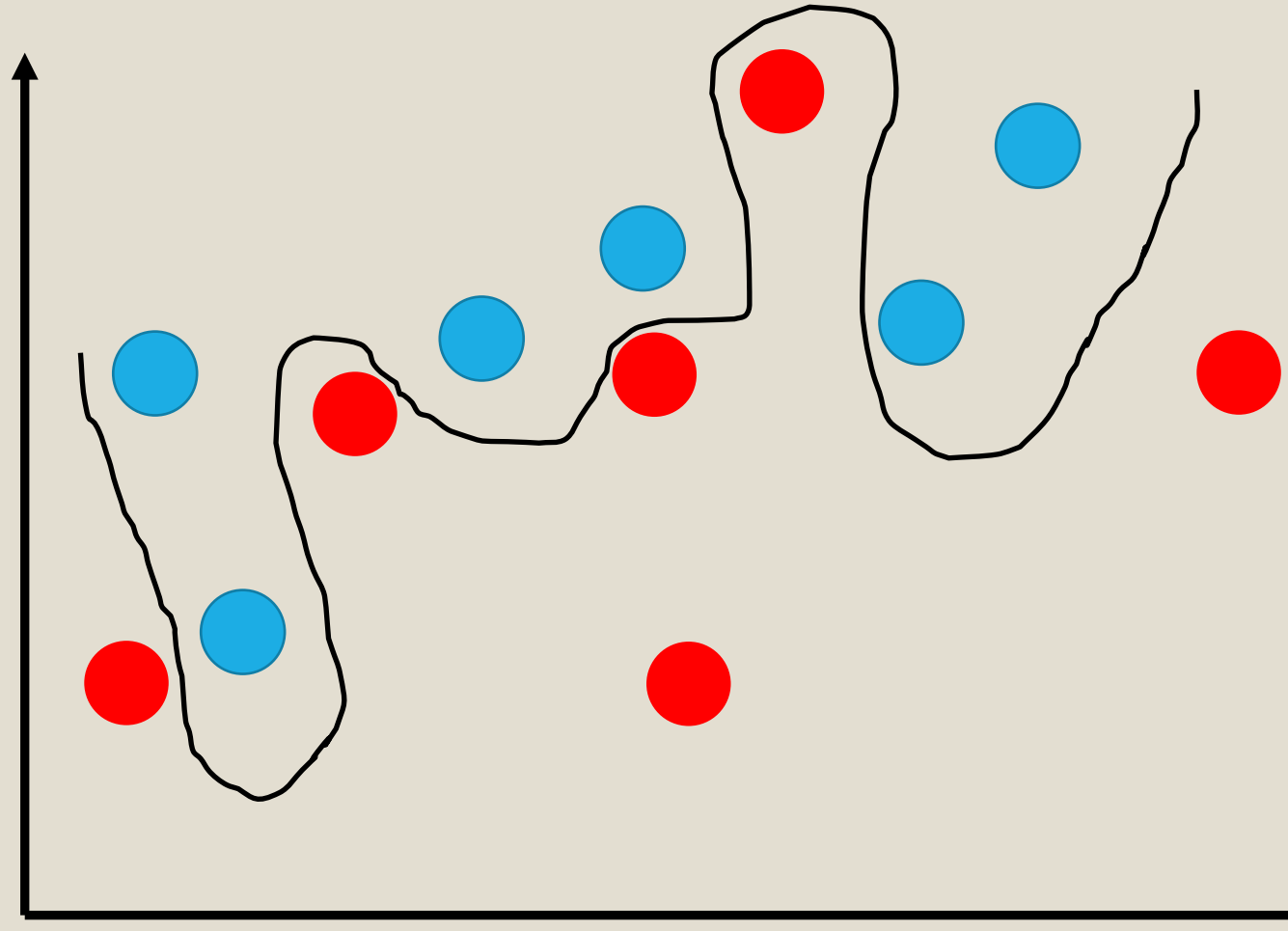
Support Vector Machine



Support Vector Machine

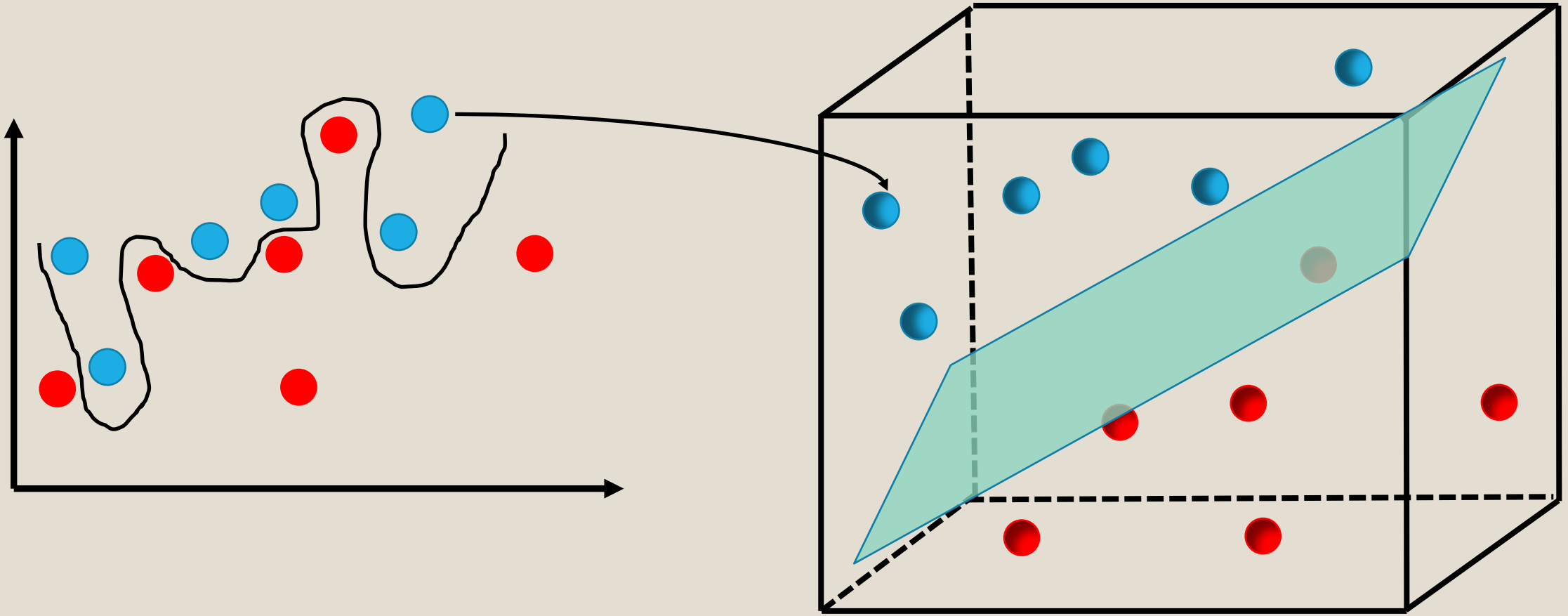


Support Vector Machine

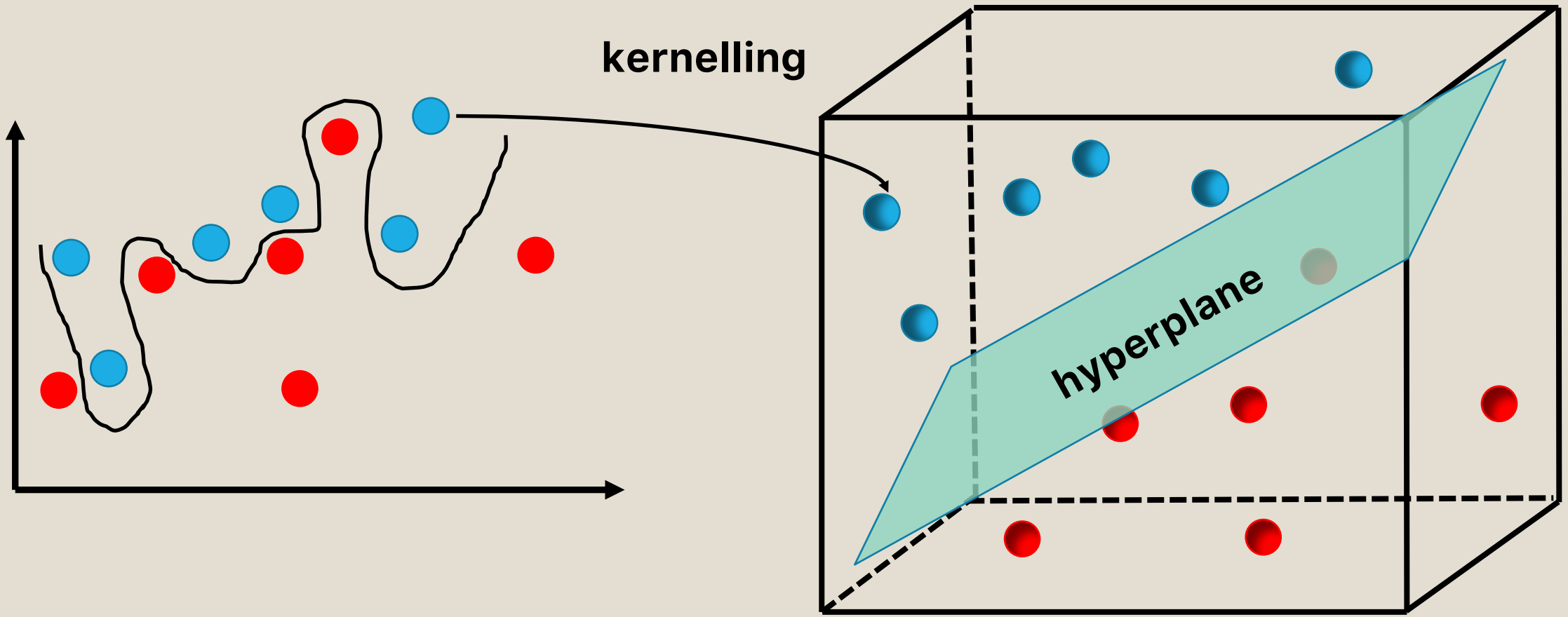


janky

Support Vector Machine



Support Vector Machine



Start SVM example

In [33]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt, matplotlib.image as mpimg
from sklearn.model_selection import train_test_split
from sklearn import svm
%matplotlib inline
```

In [34]:
```
labeled_images = pd.read_csv('/Users/minhvan/Documents/c1-summit/data/train.csv')
```

In [21]:
```
images = labeled_images.iloc[0:5000,1:]
labels = labeled_images.iloc[0:5000,:1]
train_images, test_images,train_labels, test_labels = train_test_split(images, labels, train_size=0.8, random_state=0)
```

Take a look at the input data!

In [22]:
```
i=1
img=train_images.iloc[i].as_matrix()
img=img.reshape((28,28))
plt.imshow(img,cmap='gray')
plt.title(train_labels.iloc[i,0])
```

Examine the pixel values

```
In [23]: plt.hist(train_images.iloc[i])
```

Note: the histogram of the image's pixel values are not black and white, or 0 and 1. The image is actually in gray scale, values between 0 and 255.

Training our model

```
In [24]: clf = svm.SVC()
         clf.fit(train_images, train_labels.values.ravel())
         clf.score(test_images,test_labels)
```

1. Invoke SVM classifier
2. Train our model on our training images and labels
3. Score our model against test images and test labels. Value between 0-1, 0 being a terrible predictor and 1 being a perfect predictor.

How did our model do?

Gray-scale to black and white

```
In [25]: test_images[test_images>0]=1
         train_images[train_images>0]=1

         img=train_images.iloc[i].as_matrix().reshape((28,28))
         plt.imshow(img,cmap='binary')
         plt.title(train_labels.iloc[i])
```

```
In [26]: plt.hist(train_images.iloc[i])
```

Note: the histogram of the image's pixel values are now black and white. The image pixels are now either 0 or 1.

Retraining our model

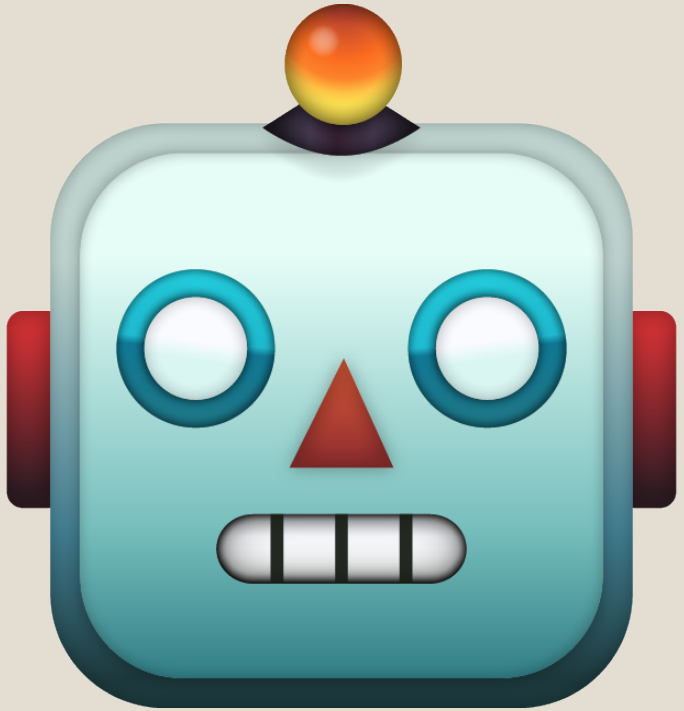
```
In [24]: clf = svm.SVC()
         clf.fit(train_images, train_labels.values.ravel())
         clf.score(test_images,test_labels)
```

How did our model do meow?

Summary

What did we learn today?

- Definition of machine learning
  - Classification vs. regression
  - Supervised vs. unsupervised
- Jupyter Notebook 101
- Python 101
- K-nearest neighbors
- SVM



Follow-up

- Go back and change the following parameters and see how it affects your accuracy
  - The k-value in the KNN example
  - Increase the number of train/test set (we only used a small portion!)
- Further resources:
  - Elements of Statistical Learning: http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf
  - Intro to ML: https://www.coursera.org/learn/machine-learning/home/info
  - Deep Learning book: https://www.deeplearningbook.org/front_matter.pdf
  - Keep up with current research: https://arxiv.org/, http://www.arxiv-sanity.com/