University of Virginia

Research Computing

MACHINE LEARNING WITH PYTHON

Jacalyn Huband

29 May 2020

What is machine learning?

A branch of artificial intelligence where computers can learn from data, and adapt the computational models to enhance performance.

A method of analysis that allows computers to reveal information within data.

Why machine learning?

- Computers can sort through data faster than humans can.
- Computers can identify patterns quickly and use these patterns for predictions or classifications.
- Machine learning can handle noisy data – it doesn’t find a perfect answer, but rather a “really good” answer.

Types of Machine Learning. . .

- Supervised Learning:
  - A data set exists where the samples can be categorized into two or more classifications.
  - The computer uses the data set to learn how to predict the classification of an unknown sample.
  - Examples include K Nearest Neighbors, and Decision Trees
- Unsupervised Learning:
  - The collected data has no known classification or pattern.
  - The computer must identify the groups or hidden structures within the data.
  - Examples include Dendograms, K-means clustering, Self-organizing Maps
- Reinforcement Learning:
  - Computer learns from positive or negative feedback
  - Example includes Swarm intelligence

Types of Machine Learning. . .

- Supervised Learning:
  - A data set exists where the samples can be categorized into two or more classifications.
  - The computer uses the data set to learn how to predict the classification of an unknown sample.
  - Examples include K Nearest Neighbors, and Decision Trees
- Unsupervised Learning:
  - The collected data has no know classification or pattern.
  - The computer must identify the groups or hidden structures within the data.
  - Examples include Dendograms, K-means clustering, Self-organizing Maps
- Reinforcement Learning:
  - Computer learns from positive or negative feedback
  - Example includes Swarm intelligence

Types of supervised ML problems

- Regression techniques
  - Determines a mathematical model for the relationship among variables or attributes so that an outcome can be predicted.
  - Results can be any value within a possible range (e.g., what will the average Earth temperature be in 2050?)

- Classification problem
  - Identifies a combination of attributes that best fits a class or category so that an object can be classified.
  - Results can be from a list of known possibilities (e.g., is the tumor benign or malignant?)

Types of supervised ML problems

- Regression techniques
  - Determines a mathematical model for the relationship among variables or attributes so that an outcome can be predicted.
  - Results can be any value within a possible range (e.g., what will the average Earth temperature be in 2050?)

- Classification problem
  - Identifies a combination of attributes that best fits a class or category so that an object can be classified.
  - Results can be from a list of known possibilities (e.g., is the tumor benign or malignant?)

Data for Machine Learning

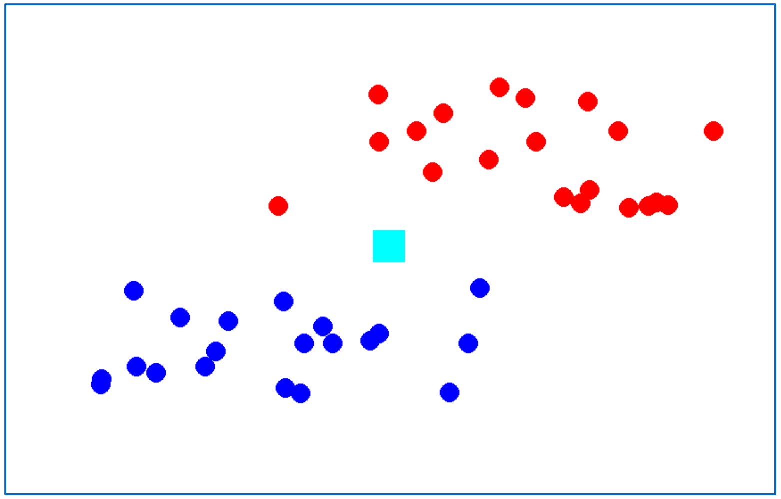
- For many Machine Learning algorithms, the data are expected to be in a table format, where
    - each row represents an object, and
    - each column has the measurements for a specific attribute or feature of the object
- For supervised learning, the classifications of the objects are known.
- The data with known classifications are divided into a training set and a testing set.
- The data are used to develop a model.
    - The training data are submitted to an algorithm that will fit a model to the data.
    - The test data are submitted to the model to produce predicted classifications and determine the accuracy of the model.
- Finally, the model can be used to predict classifications for “unknown” data.

K-NEAREST NEIGHBORS (KNN)

KNN Overview

- An example of supervised learning.

- Given a set of data with attributes and classifications, can we predict a category or outcome for an object of unknown classification?



KNN: Underlying Concepts

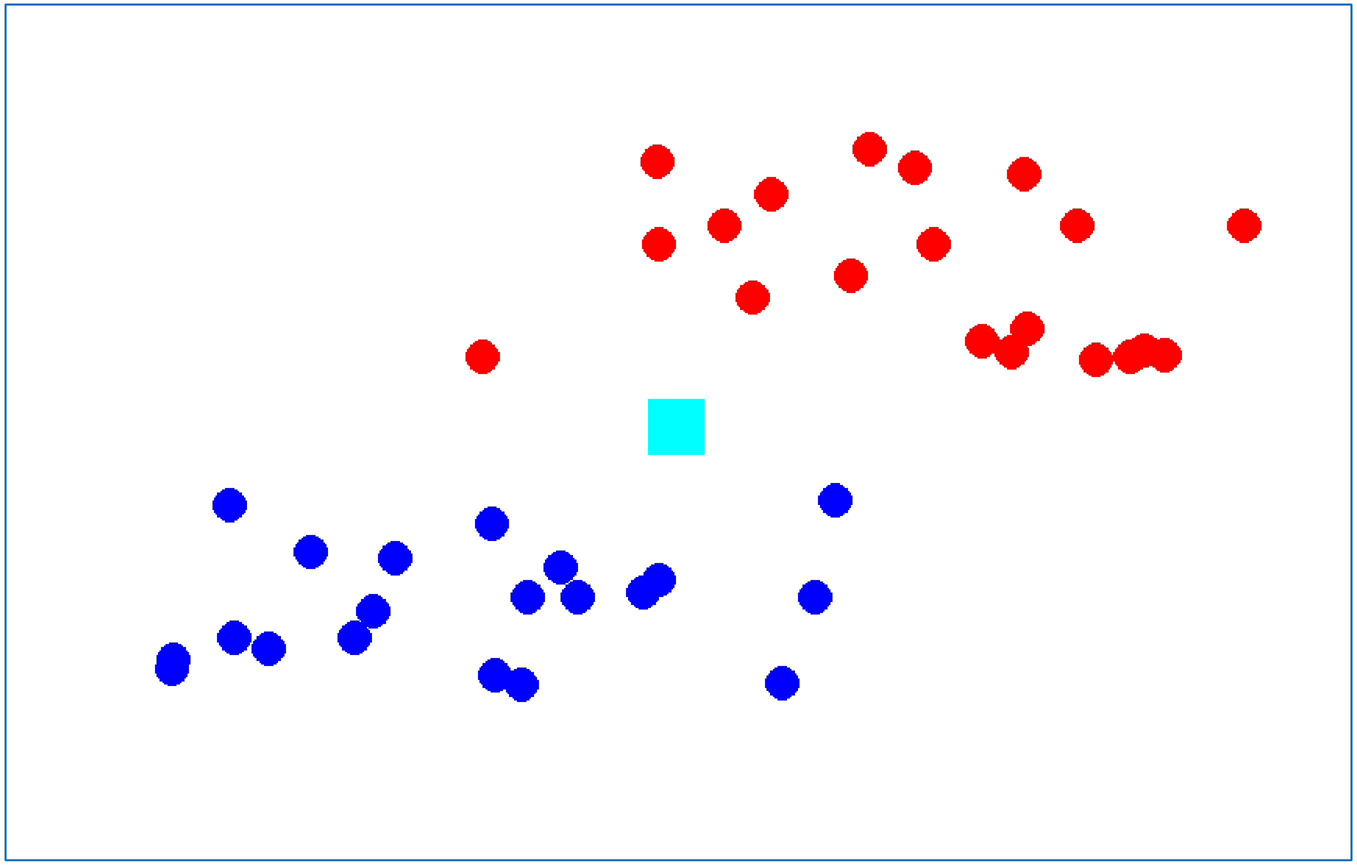
- Objects that fall within the same category should share similar features.
- Define a measurement for determining the “similarity” between two objects.
- Search for the object that is the most similar (i.e., nearest neighbor) to the unknown object.
- Assign the classification of the nearest neighbor to the unknown object.

KNN: Underlying Concepts

- What if the nearest neighbor is an outlier?
  - Instead of comparing an unknown object to its nearest neighbor, a better approach would be to compare it to the K nearest neighbors. (Replace K with your favorite integer.)
  - Usually, with best practices, we let K be an odd.

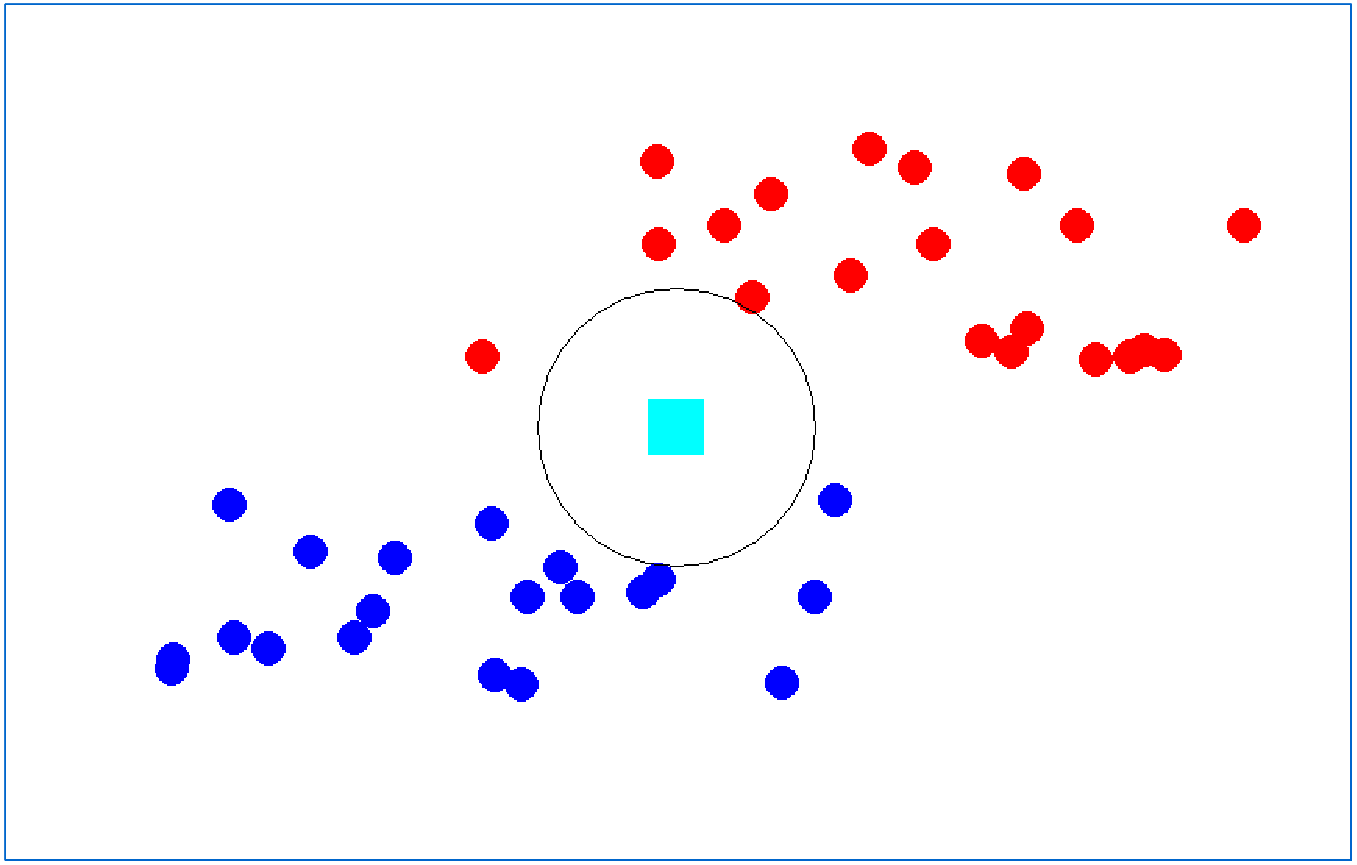
Nearest neighbors

- To which group should the cyan square belong?



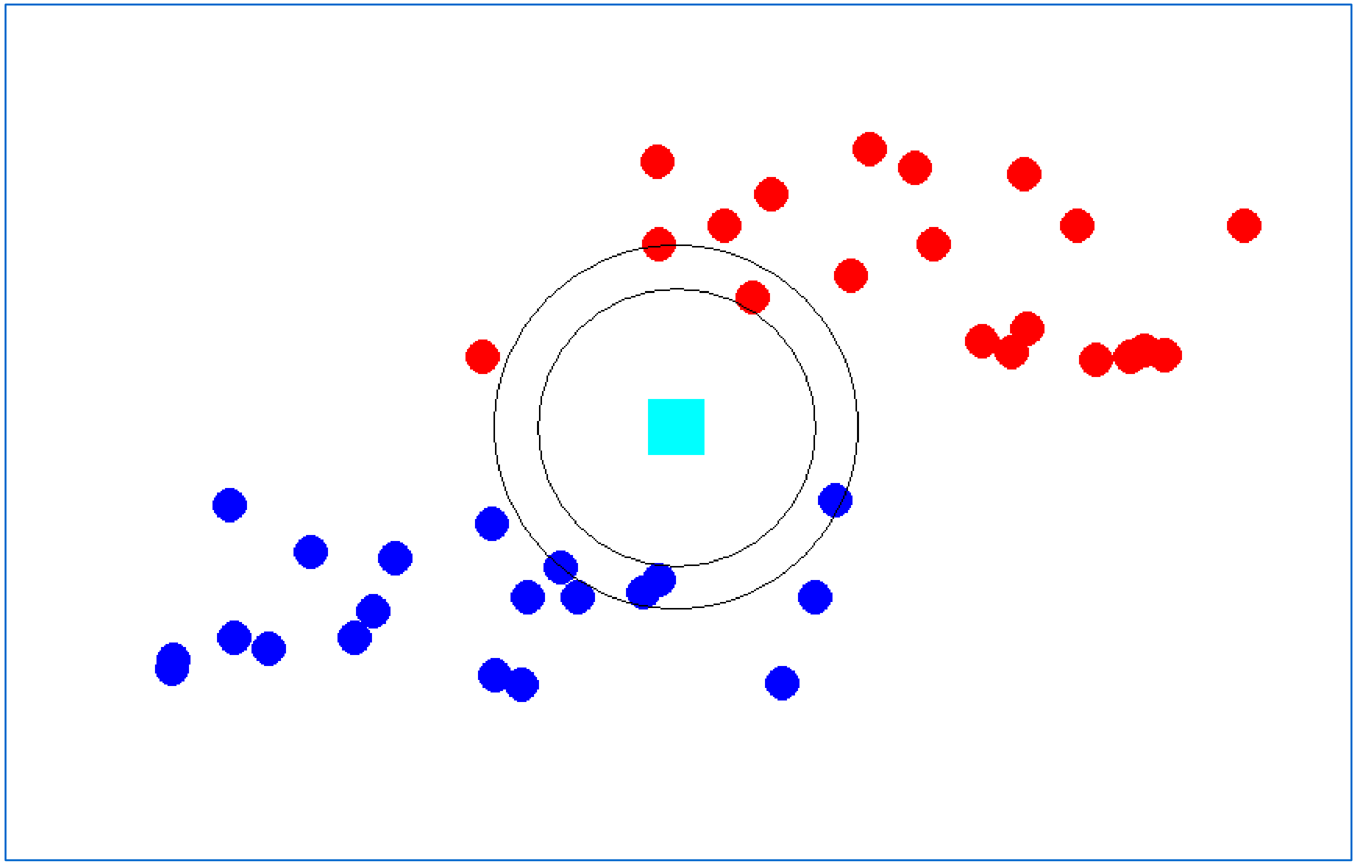
Nearest neighbors

- To which group should the cyan square belong?



Nearest neighbors

- To which group should the cyan square belong?



CODING A KNN

The Data

For our first example, we will be using a set of measurements taken on three species of iris.

The original data set is from a 1936 paper, *The use of multiple measurements in taxonomic problems,* aby Ronald Fisher, a British statistician and biologist.

The data set is so commonly used to test machine learning algorithms that it is often included as a test set in various packages and modules

The data set includes 4 measurements, taken on 150 irises (from three known species).

The Data

- iris.target_names

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

- iris.feature_names

```
['sepal length (cm)',
'sepal width (cm)',
'petal length (cm)',
'petal width (cm)']
```

Coding KNN: General Steps

1. Load the KNN packages
2. Read in the data
3. Identify the target feature
4. Divide the data into a training set and a test set.
5. Fit the random forest model
6. Apply the model to the test data
7. Display the feature importance

1. Load KNN Package

Python

```
from sklearn.neighbors import KNeighborsClassifier
```

2. Read in the data

Python

```
from sklearn import datasets
iris = datasets.load_iris()
```

The iris data set is built into sklearn.

3. Identify the Target Feature

Python

```
print(type(iris))
print(dir(iris))
print(iris.target_names)
```

The “target” (i.e., classification) is already separated from the measured data

4. Divide Data

Python

```
from sklearn import model_selection

test_size = 0.10
seed = 7
train_data, test_data, train_target, test_target =      \
        model_selection.train_test_split(iris.data,
        iris.target, test_size=test_size,
        random_state=seed)
```

5. Fit the KNN model

Python

```
n_neighbors = 3
knn = KNeighborsClassifier(n_neighbors)
knn.fit(train_data, train_target)
```

6. Apply the Model to the Test Data

Python

```
prediction = knn.predict(test_data)
```

7. Display Confusion Matrix

Python

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
print('\nAccuracy Score:  %.3f'  %          \
        accuracy_score(test_target, predictions))
print('\n*****************', \
      '\nConfusion Matrix',   \
      '\n******************')
print(confusion_matrix(test_target, predictions))
```

8. Classify an unknown object

Python

```
unknown_iris = [[15, 50, 30, 50 ]]
predicted_y = knn.predict(unknown_iris)

print('\nPrediction for unknown iris:',
      iris.target_names[predicted_y][0])
```

Activity #1: KNN Program

- Make sure that you can run the KNN code:

Python
`Py_ex1_KNN.ipynb`

DECISION TREES

Decision Tree: Overview

- A classification algorithm within supervised learning.

- Given a set of data, can we determine which attributes should be tested first to predict a category or outcome (i.e., which attributes lead to “high information gain”) ?
- We want to determine a set of questions that will guide us toward a classification of an observation.

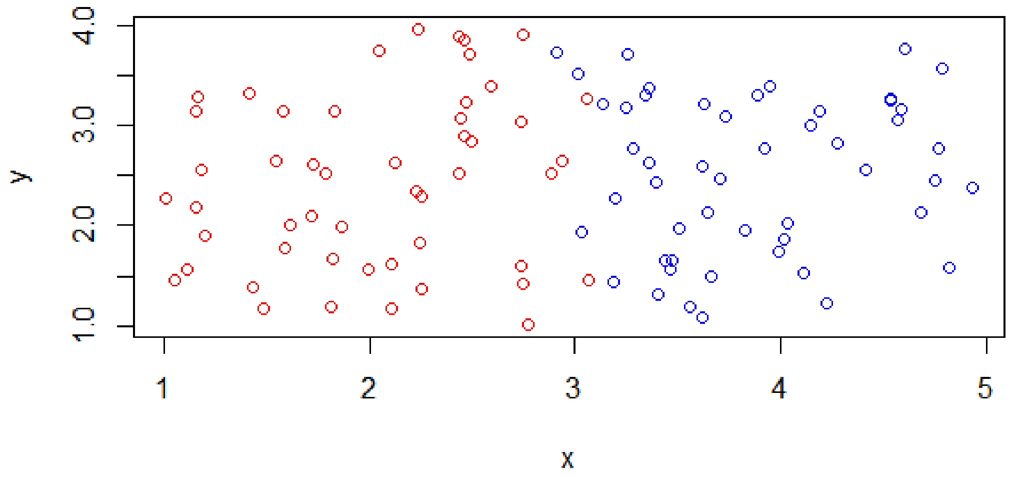
- Organizes a series of attribute tests in a tree-structure to help determine classification of the unlabeled data.

Decision Trees: How do they work?

Suppose we have

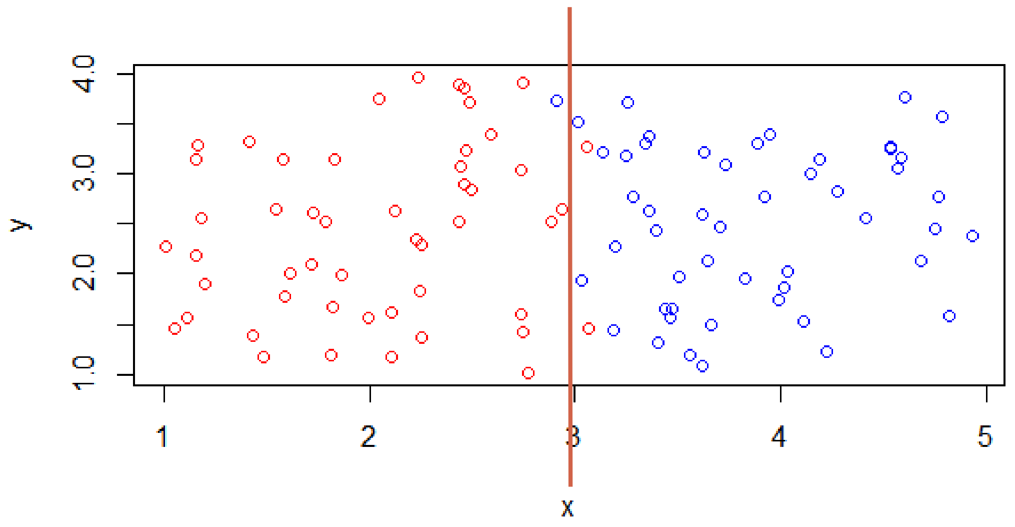
- a group of people, each one with a tumor, and
- two measurements (x, y) for each person.

Plotting the data, and coloring the points red for malignant tumors and blue for benign tumors, we might see a plot as follows:



Clearly, something happens near x = 3

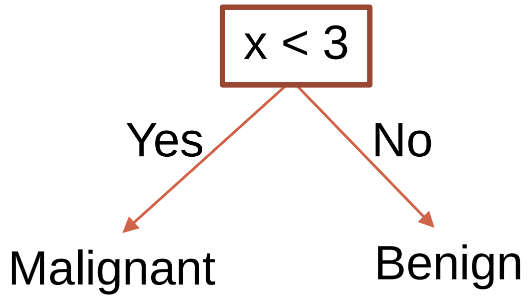
Decision Trees: How do they work?



With very few errors, we can use x=3 as our “decision” to categorizing malignant vs. benign

Unfortunately, it is not always that easy, especially if we have much more complex data.

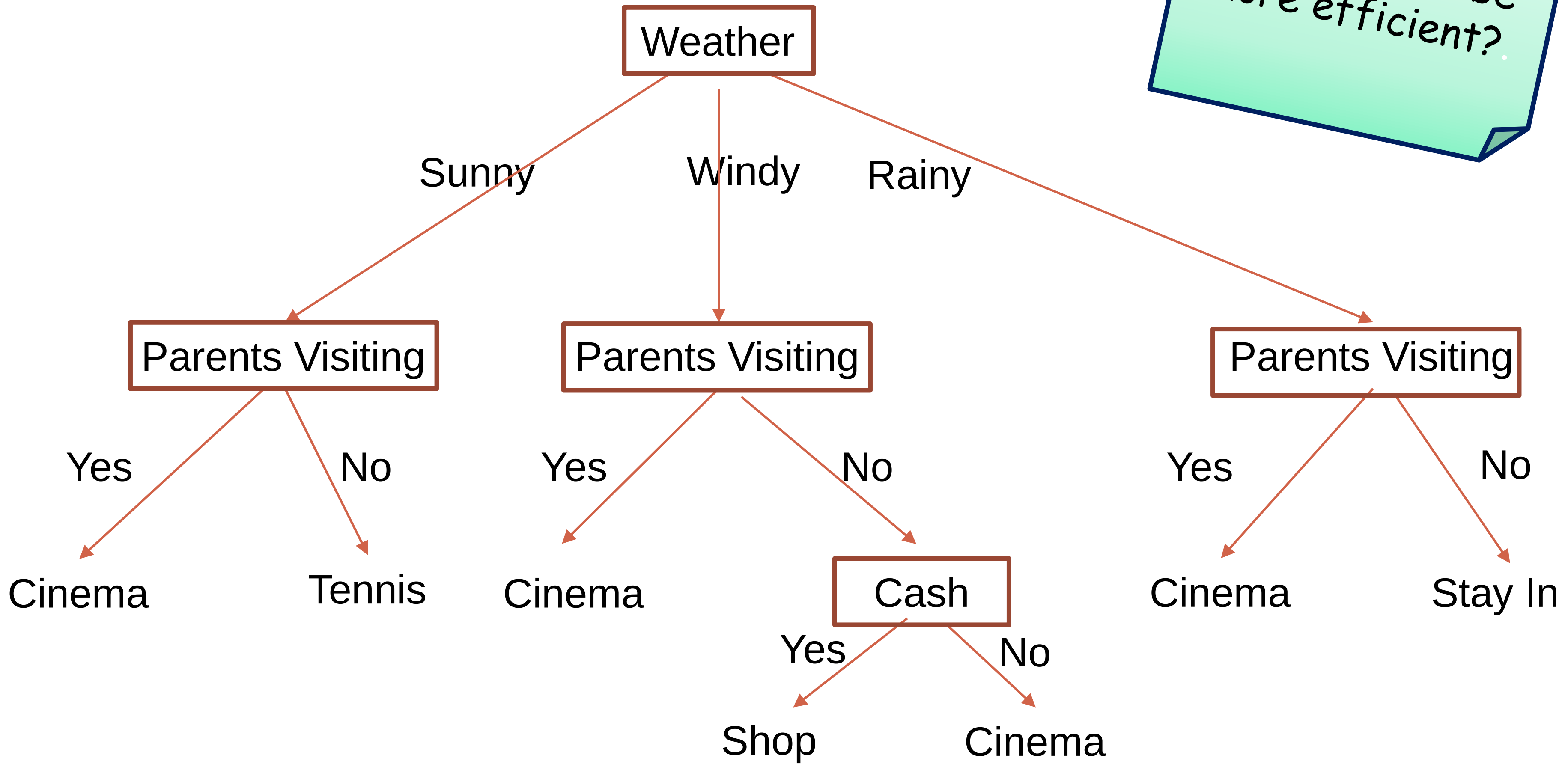
Resulting decision tree:



Example: What should you do this weekend?

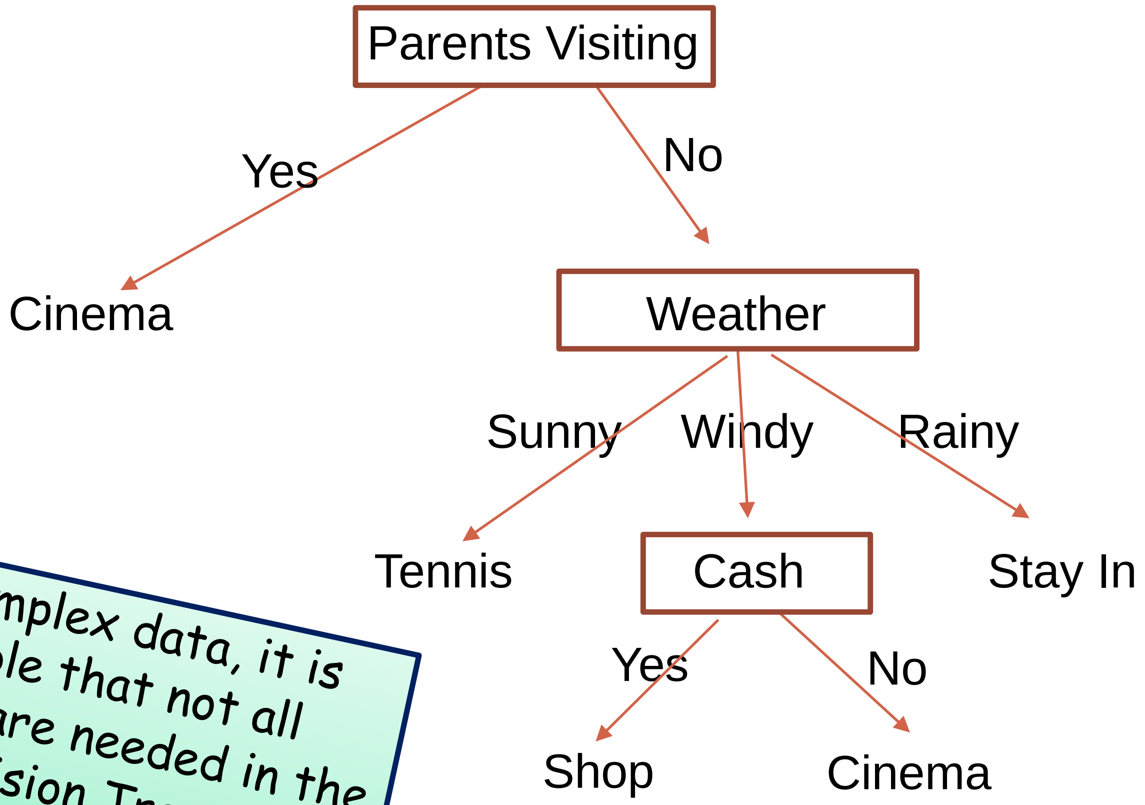
Weather	Parents Visiting	Have extra cash	Weekend Activity
Sunny	Yes	Yes	Cinema
Sunny	No	Yes	Tennis
Windy	Yes	Yes	Cinema
Rainy	Yes	No	Cinema
Rainy	No	Yes	Stay In
Rainy	Yes	No	Cinema
Windy	No	No	Cinema
Windy	No	Yes	Shopping
Windy	Yes	Yes	Cinema
Sunny	No	Yes	Tennis

What to do this weekend?



Can the tree be more efficient?

What to do this weekend?



For complex data, it is possible that not all features are needed in the Decision Tree.

Decision Tree Algorithms

- There are many existing Decision Tree algorithms.

- If written correctly, an algorithm will determine the best decisions for the tree.

How do we know how accurate our decision tree is?

Decision Tree Evaluation

- A confusion matrix is often used to show how well the model matched the actual classifications.
  - The matrix is not confusing – it simply illustrates how “confused” the model is!
- It is generated based on test data.

		Predicted Classification			
		Cinema	Shop	Stay In	Tennis
Actual Classification	Cinema	95	20	2	3
	Shop	3	7	1	1
	Stay In	2	0	5	0
	Tennis	36	8	4	73

CODING A DECISION TREE

The Data

For our first example, we will be using a set of measurements taken on various red wines.

The data set is from

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

The data are located at

https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv Load the decision tree packages

There are 12 measurements, taken on 1599 different red wines.

Attribute Summary

```
Data columns (total 12 columns):
fixed acidity          1599 non-null float64
volatile acidity       1599 non-null float64
citric acid            1599 non-null float64
residual sugar         1599 non-null float64
chlorides              1599 non-null float64
free sulfur dioxide      1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                1599 non-null float64
pH                     1599 non-null float64
sulphates              1599 non-null float64
alcohol                1599 non-null float64
quality                1599 non-null int64

dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Question: Can we predict the quality of the wine from the other attributes?

Coding Decision Trees: General Steps

1. Load the decision tree packages
2. Read in the data
3. Identify the target feature
4. Divide the data into a training set and a test set.
5. Fit the decision tree model
6. Apply the model to the test data
7. Display the confusion matrix

1. Load Decision Tree Package

Python

```
from sklearn import tree
```

2. Read in the data

Python

```
import pandas as pd
data_url = "https://archive.ics.uci.edu/ml/machine-
learning-databases/wine-quality/winequality-red.csv"

wine = pd.read_csv(data_url, delimiter=';')

print(wine.info())
```

3. Identify the target feature

Python

```
#Split the quality column out of the data
wine_target = wine['quality']
wine_data = wine.drop('quality', axis=1)
```

For the functions that we will be using, the target values (e.g., quality) must be a separate object.

4. Divide the Data

Python

```
N = len(wine_target)

#We'll use 70% for training
sample_size = int(0.70 * N)
from sklearn import model_selection
test_size = 0.10
seed = 7
train_data, test_data, train_target, test_target = 
model_selection.train_test_split(wine_data,
        wine_target, test_size=test_size,
        random_state=seed)
```

5. Fit the Decision Tree Model

Python

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(train_data, train_target)
```

6. Apply the Model to the Test Data

Python

```
prediction = clf.predict(test_data)
```

7. Display Confusion Matrix

Python

```
row_name ="Quality"
cm = pd.crosstab(test_target, prediction,     \
                 rownames=[row_name], colnames=[''])
print(' '*(len(row_name)+3),"Predicted ", row_name)
print(cm)
```

Activity #2: Decision Tree Program

- Make sure that you can run the decisionTree code:

Python
`Py_ex2_DecisionTree.ipynb`

RANDOM FOREST

Random Forest

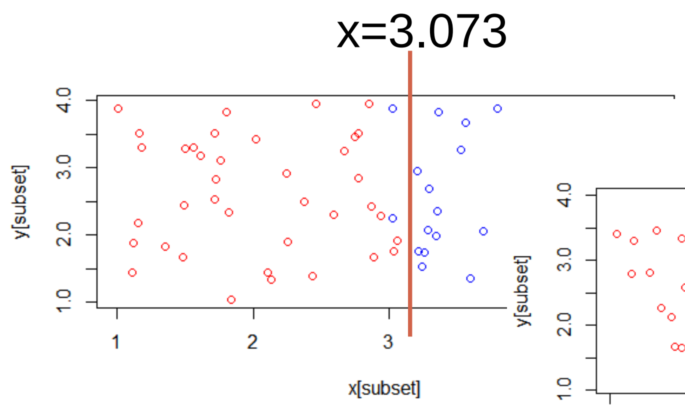
An Ensemble Technique

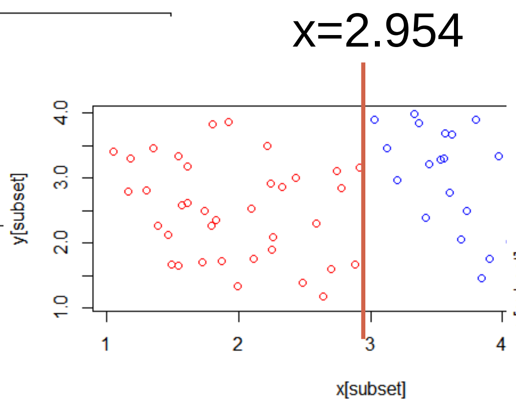
- Ensemble techniques combine a group of “weaker” learning techniques to build a stronger technique.

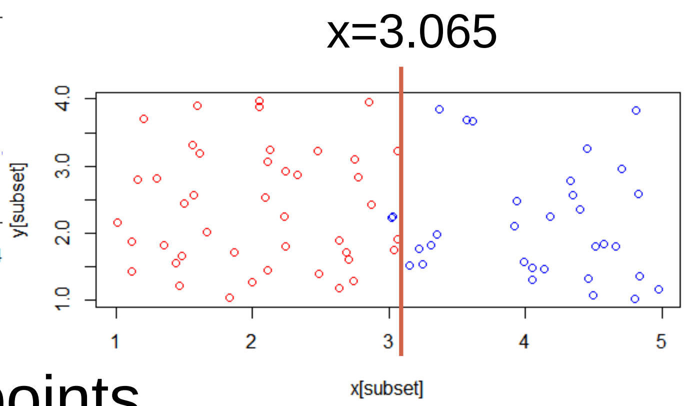
Random Forest combines the results of multiple decision trees to create a more robust result.

Random Forest: How does it work?

Samples a subset of the data and determine features that provide the best decision points.







Takes an average of the decision points.

x_ave = 3.031

Random Forest: How does it work?

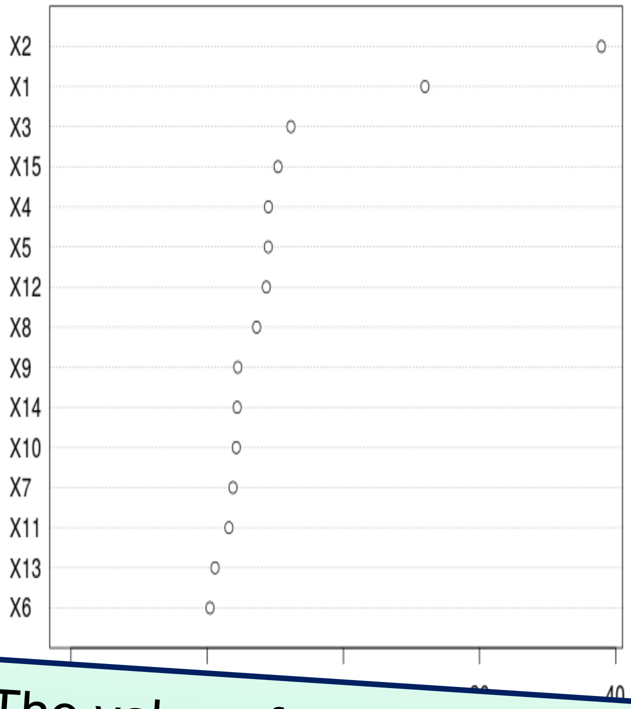
Random Forests can use different techniques for selecting features for computing each decision value.

This can lead to the choice of different features.

Random Forest: Feature Importance

- We would like to know the “importance” of the features (e.g., which features are the most important for making decisions).

- Different algorithms use various metrics to determine the importance of the features.



The value of the measurements are not as important as the order of the features.

CODING A RANDOM FOREST

The Data

For the Random Forest example, we will reuse the winequality_red data set.

Coding Random Forest: General Steps

1. Load the random forest packages
2. Read in the data
3. Identify the target feature
4. Divide the data into a training set and a test set.
    a. Choose the sample size
    b. Randomly select rows
    c. Separate the data
5. Fit the random forest model
6. Apply the model to the test data
7. Display the feature importance

1. Load Random Forest Package

Python

```
from sklearn.ensemble import RandomForestClassifier
```

Steps 2 - 4

Python

```
import pandas as pd
. . .
test_target = wine_target.drop( ndx, axis = 0)
```

Repeat steps 2 – 4 from the Decision Tree example.

5. Fit the Random Forest Model

Python

```
clf = RandomForestClassifier(n_jobs=1)
clf.fit(train_data, train_target)
```

6. Apply the Model to the Test Data

Python

```
forest_results = clf.predict(test_data)
```

7. Compute Feature Importance

Python

```
list(zip(train_data, clf.feature_importances_))
importances = clf.feature_importances_
```

8. Display Feature Importance

Python

```
Import numpy as np

indices = np.argsort(importances)[::-1]
print("Feature ranking:")
col_names = list(train_data.columns.values)
for f in range(len(indices)):
    feature = col_names[indices[f]]
    space = ' '*(20 - len(feature))
    print("%d.\t %s %s (%f)" % \
   (f + 1, feature, space, importances[indices[f]]))
```

Activity #3: Random Forest Tree Program

- Make sure that you can run the Random Forest code:

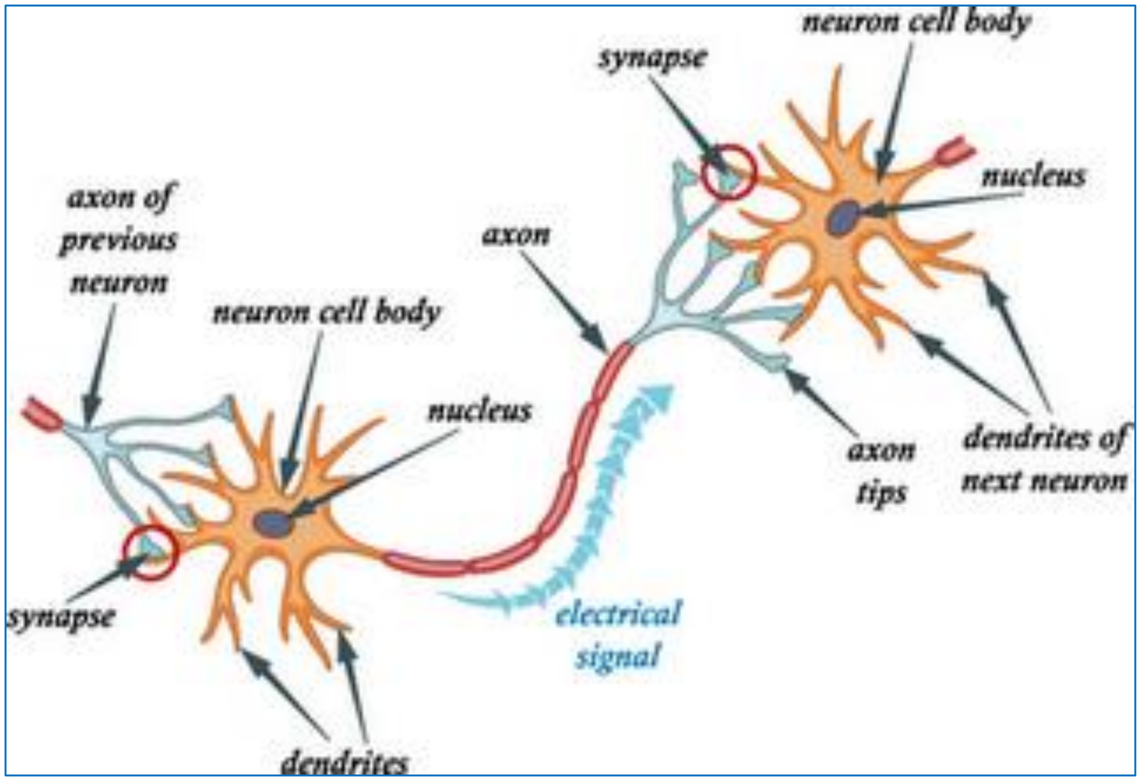
Python
`Py_ex3_RandForest.ipynb`

NEURAL NETWORKS

Neural Network

A computational model used in machine learning which is based on the biology of the human brain.

Neurons in the Brain

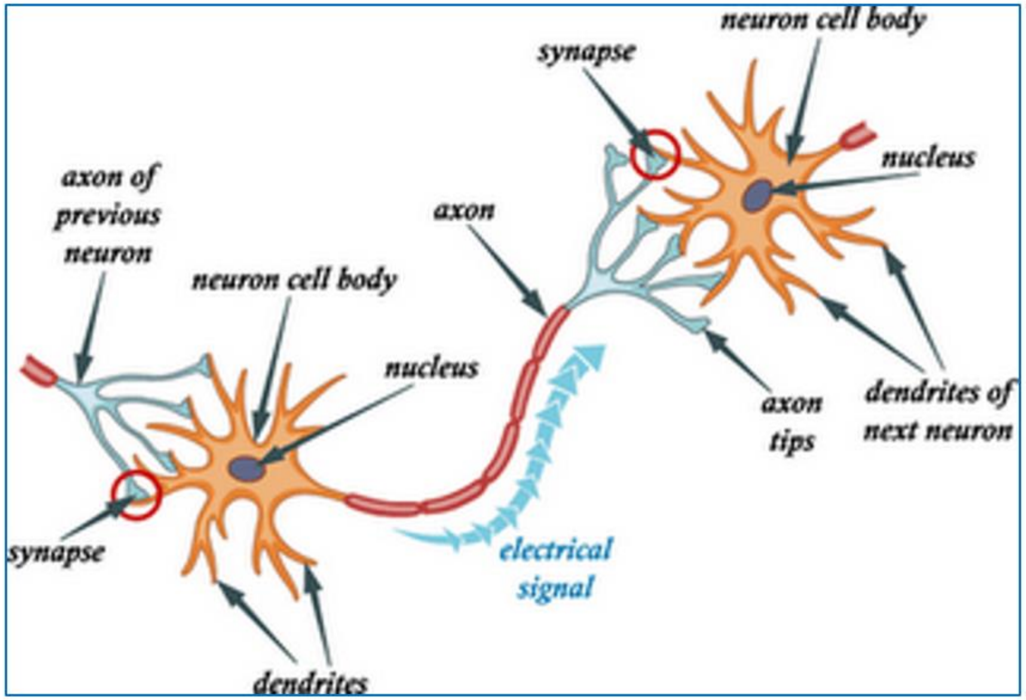


Neurons continuously receive signals, process the information, and fires out another signal.

The human brain has about 86 billion neurons, according to Dr. Suzana Herculano-Houzel

Diagram borrowed from http://study.com/academy/lesson/synaptic-cleft-definition-function.html

Neurons in the Brain

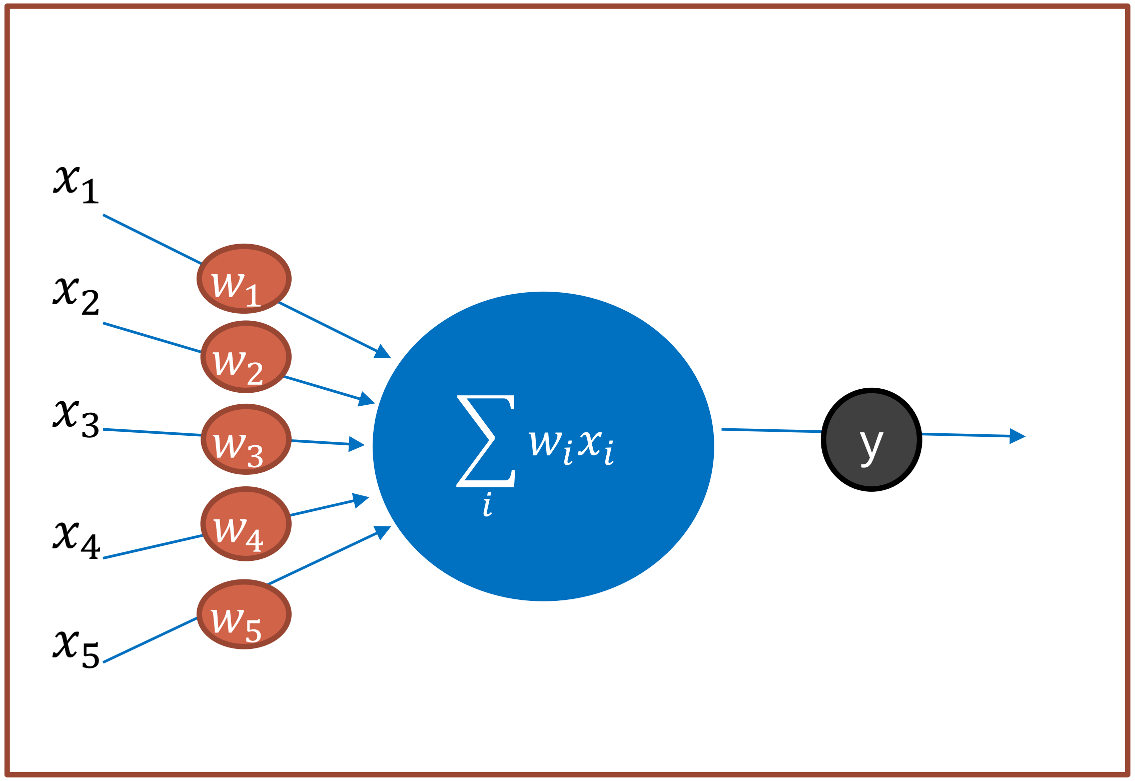


Neurons continuously receive signals, process the information, and fires out another signal.

The human brain has about 86 billion neurons, according to Dr. Suzana Herculano-Houzel

Diagram borrowed from http://study.com/academy/lesson/synaptic-cleft-definition-function.html

Simulation of a Neuron

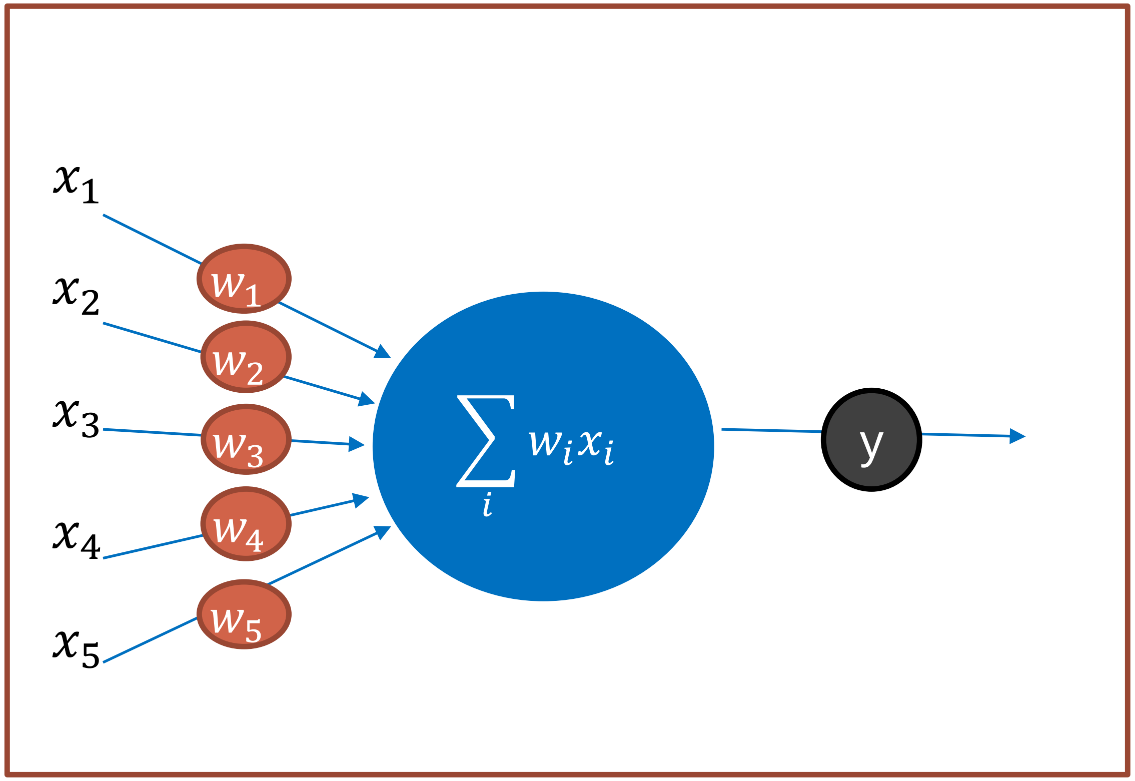


The “incoming signals” could be values from a data set(s).

A simple computation (like a weighted sum) is performed by the “nucleus”.

The result, y, is “fired out”.

Simulation of a Neuron



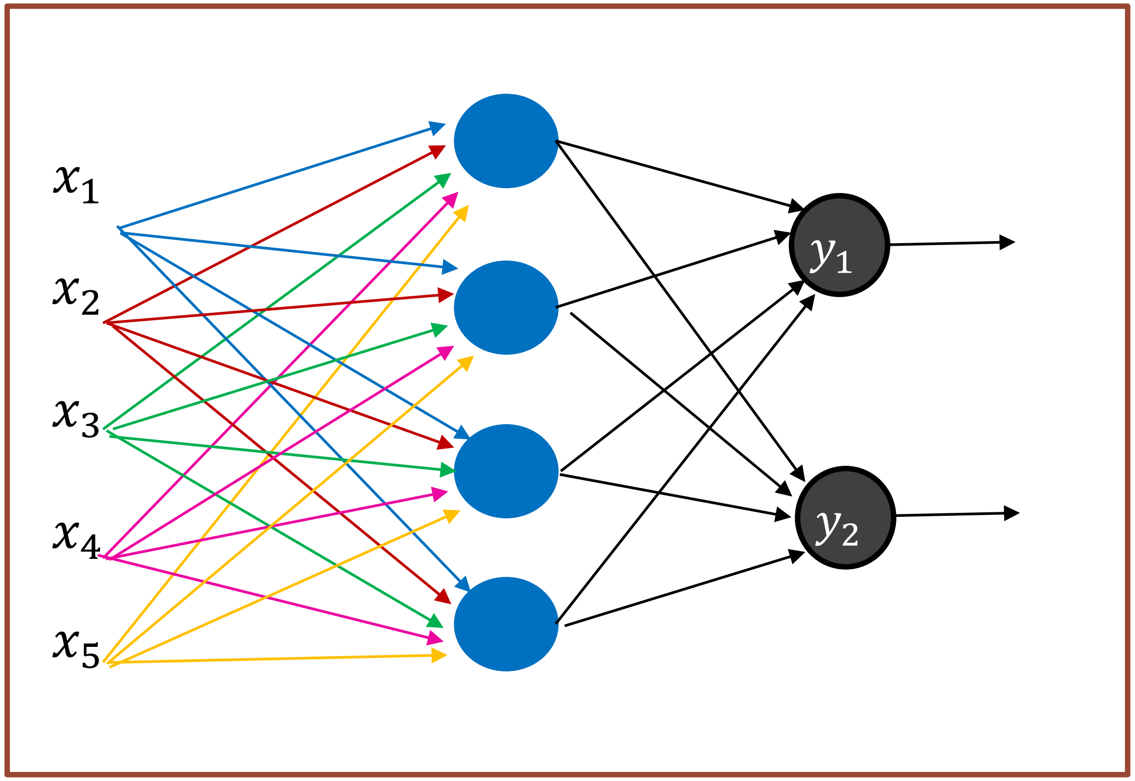
The weights, w_i, are not known.

During training, the "best" set of weights are determined that will generate a value close to y given a collection of inputs x_i.

Simulation of a Neuron

A single neuron does not provide much information (often times, a 0/1 value)

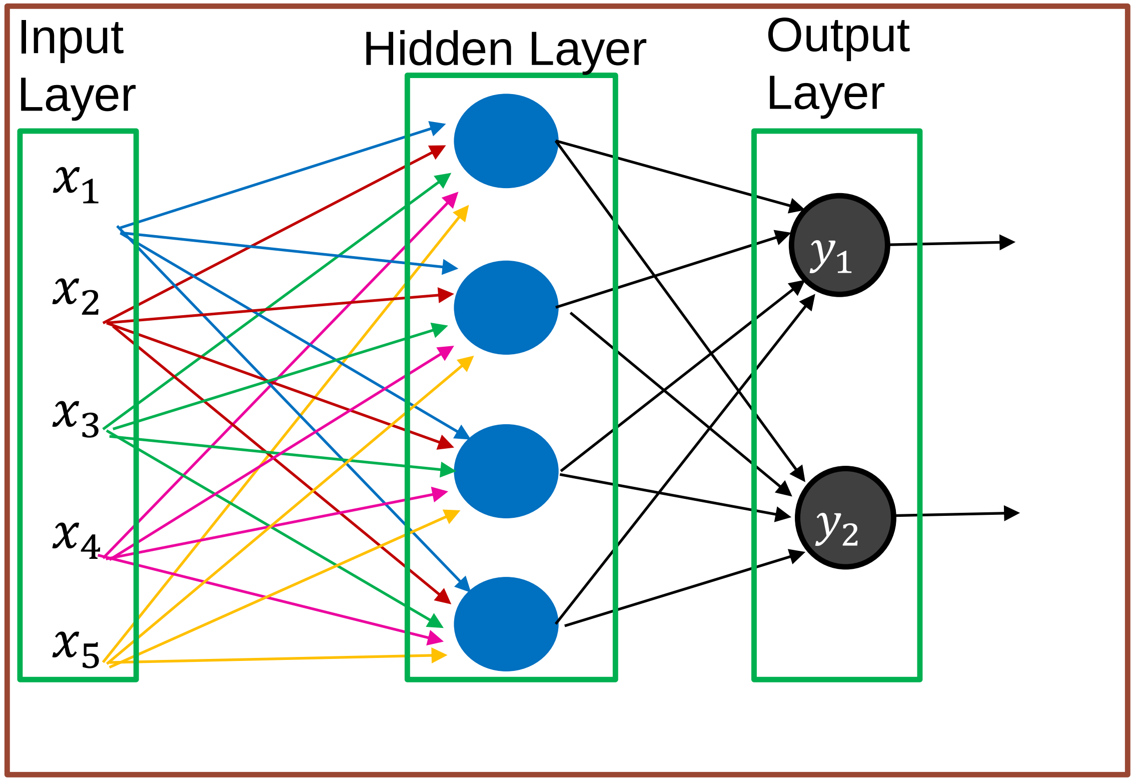
A Network of Neurons



Different computations with different weights can be performed to produce different outputs.

This is called a feedforward network because all values progress from the input to the output.

A Network of Neuron



A neural network has a single hidden layer

A network with two or more hidden layers is called a “deep neural network”.

CODING A NEURAL NETWORK

Example: Breast Cancer Data

- The Cancer data set originally was captured at UCI Repository (https://archive.ics.uci.edu/ml/datasets.html)
- Look at the data, so that you understand what is in each file.

Filename	Brief Description
cancer_data.csv	The table of measurements cancer_DESCR.csv – an overview of the data
cancer_feature_names.csv	The names of the columns in cancer_data.csv
cancer_target.csv	The classification (0 or 1) of each row in cancer_data.csv
cancer_target_names.csv	The names of the classifications (malignant or benign)

Coding a Neural Network: General Steps

1. Load the neural network packages
2. Read in the data
3. Divide the data into a training set and a test set.
4. Preprocess the data
5. Train the neural network
6. Apply the model to the test data
7. Display the confusion matrix

1. Load Neural Networks Package

Python

```
from sklearn.neural_network import MLPClassifier
#MLP means muli-layer perceptron
```

2. Read in the Data

Python

```
import numpy as np
data_file = 'cancer_data.csv'
target_file = 'cancer_target.csv'
cancer_data=np.loadtxt(data_file,dtype=float,delimiter=',')
cancer_target=np.loadtxt(target_file, dtype=float,
delimiter=',')
```

3. Divide Data

Python

```
from sklearn import model_selection
test_size = 0.30
seed = 7
train_data, test_data, train_target, test_target =
model_selection.train_test_split(cancer_data,
        cancer_target, test_size=test_size,
        random_state=seed)
```

4. Preprocess the Data

Python

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(train_data)
train_data = scaler.transform(train_data)
test_data = scaler.transform(test_data)
```

5. Train the Model

Python

```
num_features=test_data.shape[1]
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5,
hidden_layer_sizes=(num_features, ))
mlp.fit(train_data,train_target)
```

6. Apply the Model to the Test Data

Python

```
predictions = mlp.predict(test_data)
```

7. Display Confusion Matrix

Python

```
from sklearn.metrics import        \
             classification_report,confusion_matrix
print(confusion_matrix(test_target, predictions))

print(classification_report(test_target,predictions))
```

Activity: Neural Network Program

- Make sure that you can run the Neural Network codes:

Python
`Py_ex4_NeuralNets.ipynb`

TENSOR FLOW

What is TensorFlow?

An example of deep learning; a neural network that has many layers.

A software library, developed by the Google Brain Team

Deep Learning Neural Network

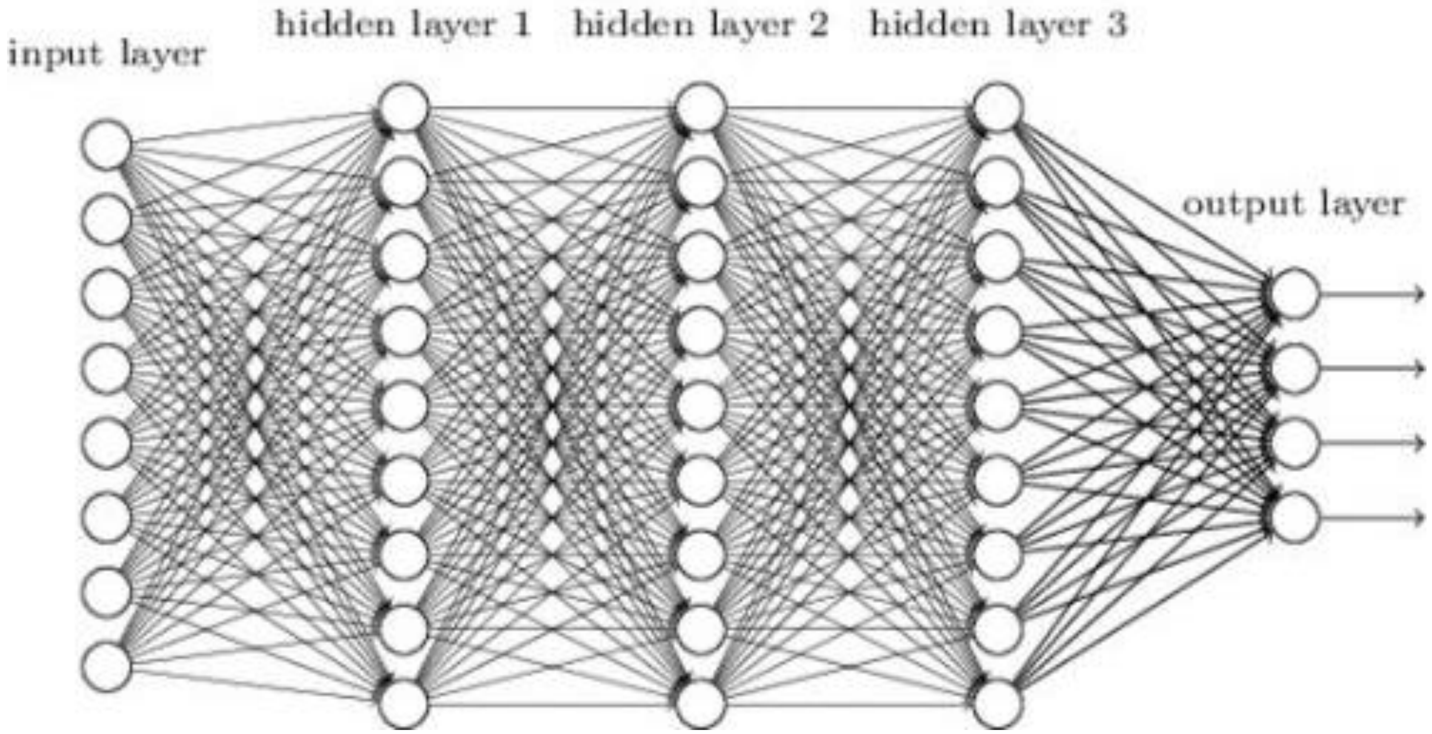


Image borrowed from:
http://www.kdnuggets.com/2017/05/deep-learning-big-deal.html

Terminology: Tensors

Tensor: A multi-dimensional array

Example: A sequence of images can be represented as a 4-D array: [image_num, row, col, color_channel]

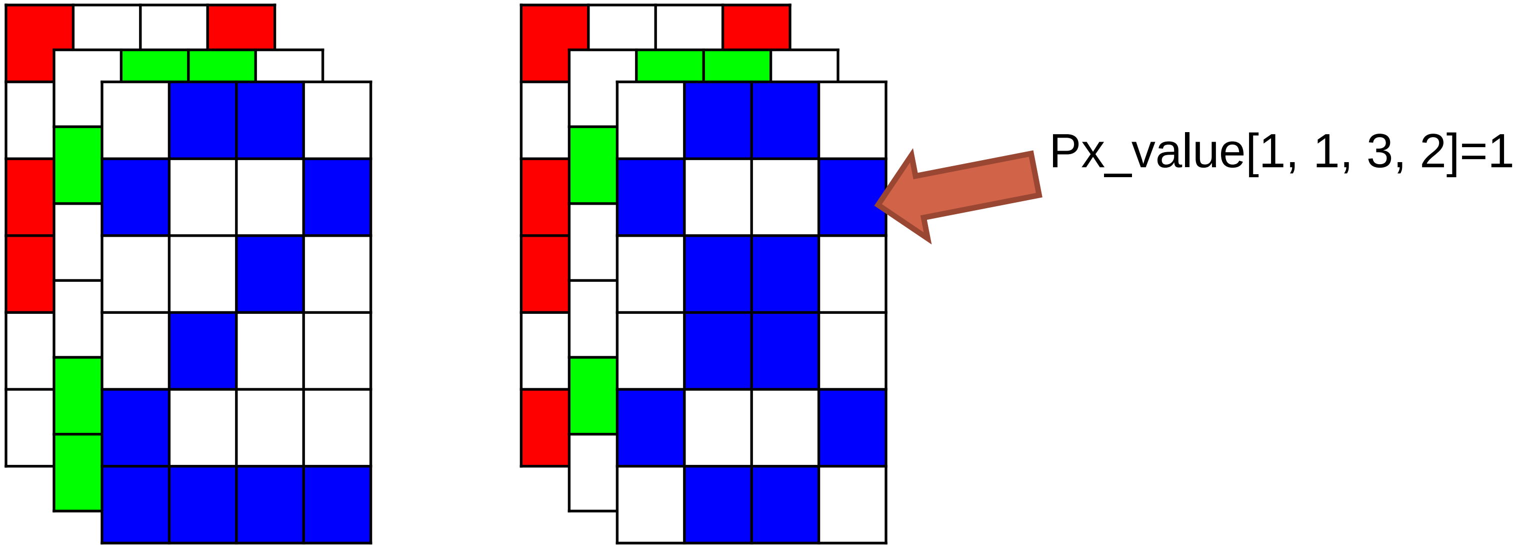
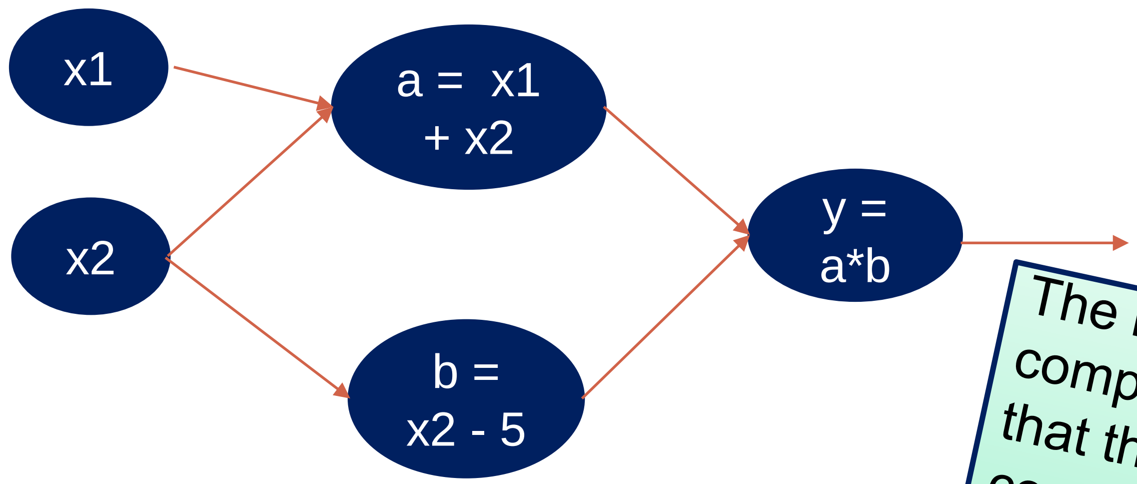


Image #0

Image #1

Terminology: Computational Graphs

- Computational graphs help to break down computations.
  - For example, the graph for y=(x1+x2)*(x2 - 5) is



The beauty of computational graphs is that they show where computations can be done in parallel.

TensorFlow: Overview

- TensorFlow programs are organized into two sections:
  - *A construction phase*
    - Tensor variables are set up
    - The computational graph is defined
    - A “loss” or “cost” function is defined
  - *An execution phase*
    - The session is created
    - The operations defined in the graph are performed

TensorFlow: Sessions

No computation is done until a session is launched.

The session assigns the operations to devices (e.g., GPU/CPUs).

CODING A TENSOR FLOW

Coding Tensor Flow: General Steps

1. Load the tensor flow/keras packages
2. Read in the data
3. Split the data
4. Pre-process the data
5. Define the model
6. Configure the Learning Process

7. Fit the Model to the Training Data
8. Apply the model to the test data
9. Evaluate the results

1. Load Keras Packages

Python

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.utils import to_categorical
from keras.optimizers import SGD
```

2. Read in the Data

Python

```
import numpy as np
data_file = 'cancer_data.csv'
target_file = 'cancer_target.csv'
cancer_data=np.loadtxt(data_file,dtype=float,delimiter=
',')
cancer_target=np.loadtxt(target_file, dtype=float,
delimiter=',')
```

3. Split the Data

Python

```
from sklearn import model_selection
test_size = 0.30
seed = 7
train_data, test_data, train_target, test_target = 
model_selection.train_test_split(cancer_data,
        cancer_target, test_size=test_size,
        random_state=seed)
```

4. Pre-process the Data

Python

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler.fit(train_data)

# Now apply the transformations to the data:
x_train = scaler.transform(train_data)
x_test = scaler.transform(test_data)

# Convert the classes to ‘one-hot’ vector
y_train = to_categorical(train_target, num_classes=2)
y_test = to_categorical(test_target, num_classes=2)
```

5. Define the Model

Python

```
model = Sequential()
# in the first layer, you must specify the expected
#input data shape
# here, 30-dimensional vectors.
model.add(Dense(30, activation='relu', input_dim=30))
model.add(Dropout(0.5))
model.add(Dense(60, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
print(model.summary())
```

6. Configure the Learning Processs

Python

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
nesterov=True)
model.compile(loss='categorical_crossentropy',
optimizer=sgd, metrics=['accuracy'])
```

7. Fit the Model

Python

```
b_size = int(.8*x_train.shape[0])

model.fit(x_train, y_train, epochs=300,
batch_size=b_size)
```

8. Apply the Model to Test Data

Python

```
predictions = model.predict_classes(x_test)
```

9. Evaluate the Results

Python

```
score = model.evaluate(x_test, y_test,
batch_size=b_size)
print('\nAccuracy:  %.3f' % score[1])
from sklearn.metrics import confusion_matrix
print(confusion_matrix(test_target, predictions))
```

Activity: TensorFlow Program

- Make sure that you can run the TensorFlow code:

Python
`Py_ex5_TensorFlow.ipynb`

CONVOLUTIONAL NEURAL NETWORKS

What are Convolutional Neural Networks?

Originally, convolutional neural networks (CNNs) were a technique for analyzing images.

CNNs apply multiple neural networks to subsets of a whole image in order to identify parts of the image.

Applications have expanded to include analysis of text, video, and audio.

The Idea behind CNN

Recall the old joke about the blind-folded scientists trying to identify an elephant.

A CNN works in a similar way. It breaks an image down into smaller parts and tests whether these parts match known parts.

It also needs to check if specific parts are within certain proximities.

For example, the tusks are near the trunk and not near the tail.

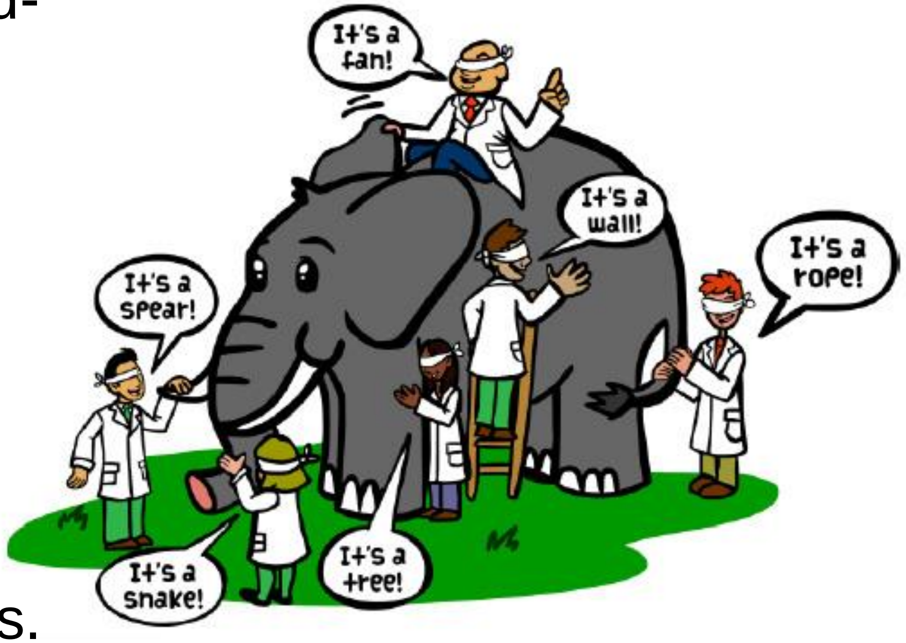
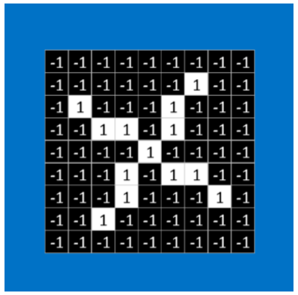
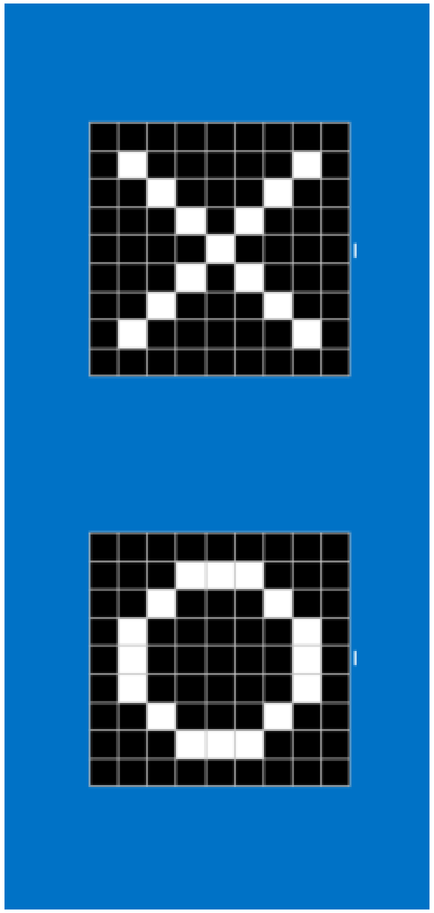


Image borrowed from https://tekrighter.wordpress.com/2014/03/13/metabolomics-elephants-and-blind-men/

Is the image on the left most like an X or an O?

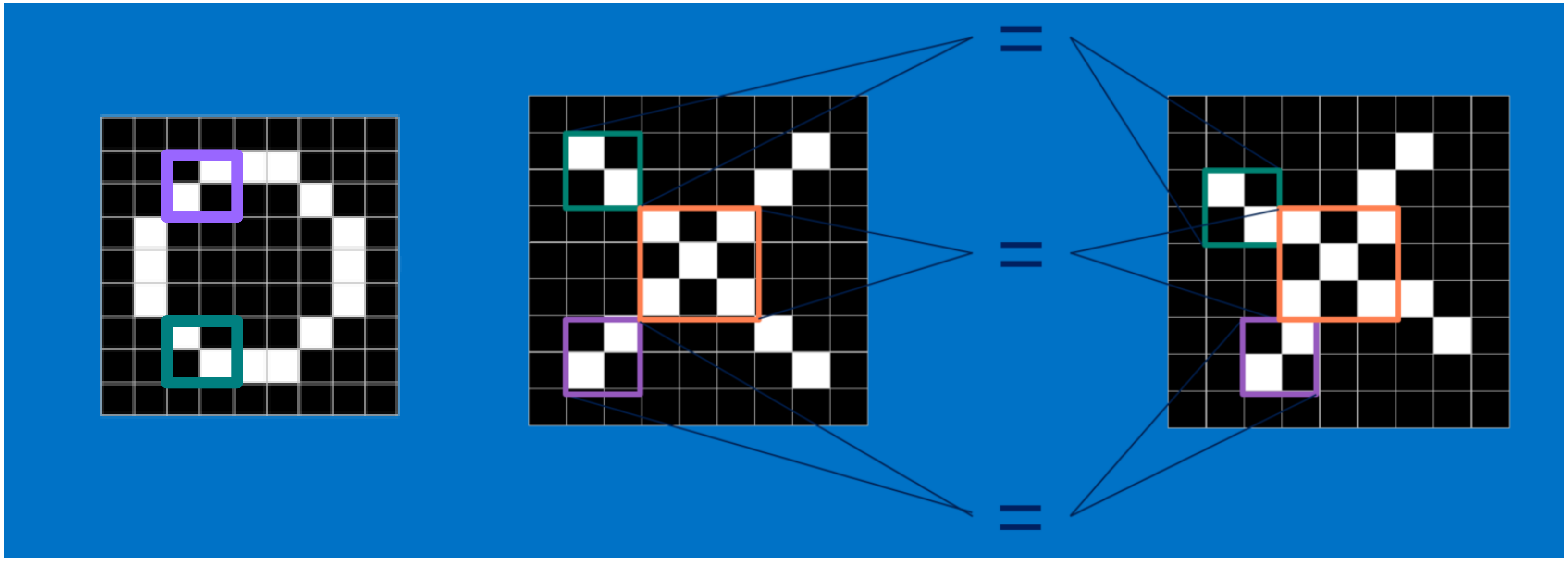


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1



Images borrowed from
http://brohrer.github.io/how_convolutional_neural_networks_work.html

What features are in common?

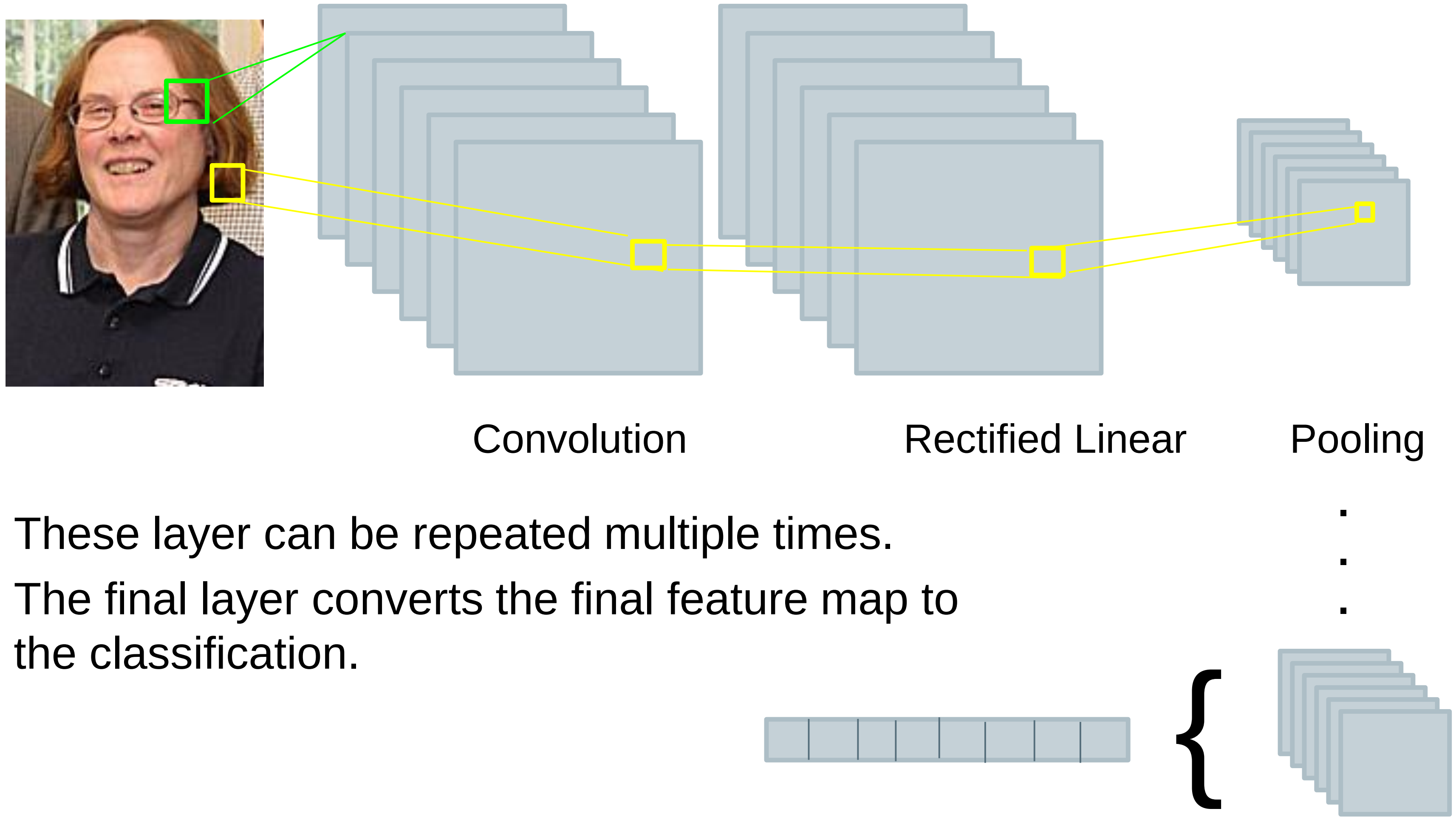


Building Blocks of CNN

- CNN performs a combination of layers
  - Convolution Layer
    - Compares a feature with all subsets of the image
    - Creates a map showing where the comparable features occur
  - Rectified Linear Units (ReLU) Layer
    - Goes through the features maps and replaces negative values with 0
  - Pooling Layer
    - Reduces the size of the rectified feature maps by taking the maximum value of a subset

- And, ends with a final layer
  - Classification (Fully-connected layer) layer
    - Combines the specific features to determine the classification of the image

Steps



- These layer can be repeated multiple times.
- The final layer converts the final feature map to the classification.

Example: MNIST Data

- The MNIST data set is a collection of hand-written digits (e.g., 0 – 9).
- Each digit is captured as an image with 28x28 pixels.
- The data set is already partitioned into a training set (60,000 images) and a test set (10,000 images).
- The tensorflow packages have tools for reading in the MNIST datasets.
- More details on the data are available at http://yann.lecun.com/exdb/mnist/

Image borrowed from *Getting Started with TensorFlow* by Giancarlo Zaccone

Coding CNN: General Steps

1. Load the tensor flow/keras packages
2. Read in the data
3. Pre-process the data
4. Define the model
5. Configure the Learning Process

6. Fit the model to the training data
7. Apply the model to test data & evaluate results

1. Load Keras Packages

Python

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation,
Convolution2D, MaxPooling2D, Flatten, Dropout
from keras.utils import np_utils
```

2. Read in the Data

Python

```
(X_train, Y_train), (X_test, Y_test) =
mnist.load_data()
```

3a. Pre-process the Data: Set up Constants

Python

```
numTrain = X_train.shape[0]
numTest = X_test.shape[0]
numRows = X_train.shape[1]
numCols = X_train.shape[2]
labels = set(Y_train)
input_size = numRows * numCols
numLabels = len(labels)
batch_size = 100
hidden_neurons = 200
epochs = 8
np.random.seed(1234)  #for reproducibility
```

3b. Pre-process the Data: Reshape

Python

```
X_train = X_train.reshape(numTrain, numRows, numCols,
1)
X_test = X_test.reshape(numTest, numRows, numCols, 1)

#  Scale values
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

#  Convert labels to ‘one-hot’ vectors
Y_train = np_utils.to_categorical(Y_train, numLabels)
Y_test = np_utils.to_categorical(Y_test, numLabels)
```

4a. Define the Model: Initial Layer

Python

```
#  Set up model for tensorflow
subfield = (3,3)
pooling_size = (2,2)
num_color_channels = 1
numNodes = 32

model = Sequential()
#Initial Input layer
model.add(Convolution2D(numNodes, subfield,
input_shape=(numRows, numCols, num_color_channels)))
model.add(Activation('relu'))
```

4b. Define the Model: Hidden Layers

Python

```
#  First Convolution Layer
model.add(Convolution2D(numNodes, subfield))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=pooling_size))
model.add(Dropout(0.25))
model.add(Flatten())

# Fully Connected Layer
model.add(Dense(hidden_neurons))
model.add(Activation('relu'))
```

4c. Define the Model: Output Layer

Python

```
#  Output Layer
model.add(Dense(numLabels))
model.add(Activation('softmax'))
```

5. Configure the Learning Process

Python

```
model.compile(loss='categorical_crossentropy',
              metrics=['accuracy'], optimizer='adam')
```

6. Fit the Model to Training Data

Python

```
model.fit(X_train, Y_train,
batch_size=batch_size,epochs=epochs, validation_split =
0.1, verbose=1)
score = model.evaluate(X_train, Y_train, verbose=1)
print('\nTrain accuracy:', score[1])
```

7. Apply Model to Test Data

Python

```
score = model.evaluate(X_test, Y_test, verbose=1)
print('\nTest accuracy:', score[1])
```

Activity: CNN Program

- Make sure that you can run the CNN code:

Python
`Py_ex6_CNN.ipynb`

LONG SHORT-TERM MEMORY

What is Long Short-Term Memory?

An example of a recurrent neural network that can handle sequences of data.

The sequences can have dependencies based on time or distance.

These networks can be used to analyze text, videos, and time-series data.

The Idea behind LSTMs

Often times information builds upon previous information.

Example: What would be the next word in the sentence

“The clouds are in the _____”

The Idea behind LSTMs

The more distant the relevant information, the less likely a recurrent neural network will determine the relationship.

Example: What would be the next word in the sentence

> “I grew up in Mexico. It was a wonderful childhood, and I am fluent in ___________”

The Idea behind LSTMs

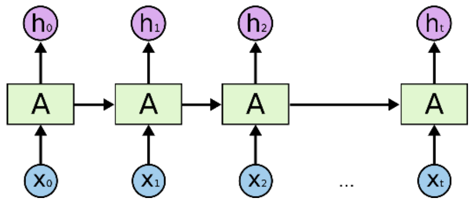
Perhaps a more relevant example:

To fully understand *The Avengers: Endgame*, you would need to watch several of the previous Avenger movies.

Images borrowed from:
https://en.wikipedia.org/wiki/Thor_(Marvel_Cinematic_Universe)
https://alteregocomics.com/avengers-infinity-war-thor-1-6-scale-figure/
https://nytimespost.com/avengers-endgame-chris-hemsworth-reveals-the-worst-part-about-fat-thor/

The Idea behind LSTMs

A Recurrent Neural Network (RNN) passes on information



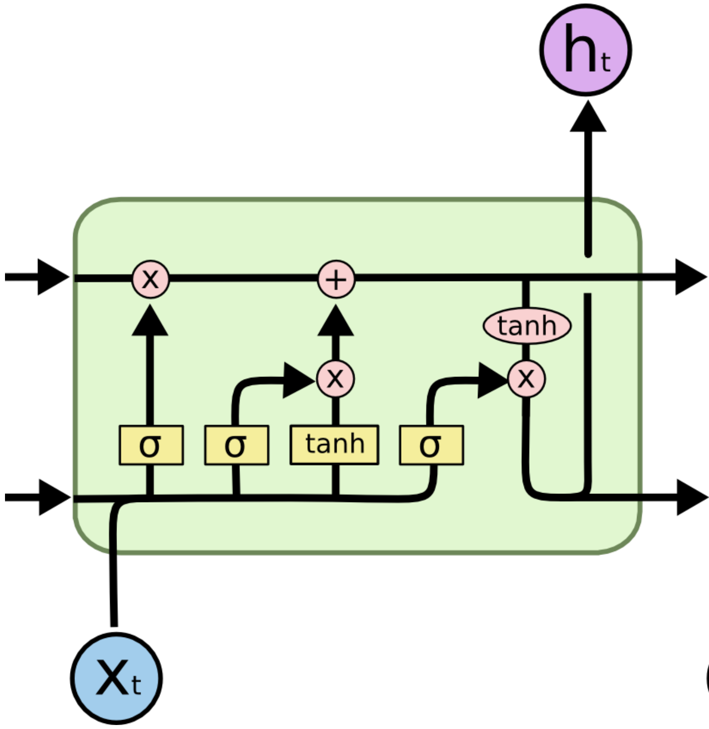
A drawback for RNNs is that the more distant the information, the more likely the information will be forgotten.

LSTMs must have a mechanism to maintain a history of what has gone before each object.

They do this by adding more detailed structures within each node..

Building Blocks of LSTMs

- The nodes, called cells, process the inputs in 4 ways.
  1. Determine which current information should be thrown away (forget gate)
  2. Determine what new information should be kept (input gate)
  3. Update the current state of the cell
  4. Determine what information will be passed on (output gate)



Images borrowed from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Coding LSTM: General Steps

1. Load the tensor flow/keras packages
2. Read in the data
3. Pre-process the data
   a. Simplify to lower case and no punctuation
   b. Tokenize into words and convert to sequences
   c. Split into training data & testing data
4. Define the model
5. Configure the Learning Process

6. Fit the model to the training data
7. Apply the model to test data & evaluate results

1. Load Keras Packages

Python

```
import numpy as np
import pandas as pd

from sklearn.feature_extraction.text import
CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM,
SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
```

2. Read in the Data

Python

```
data = pd.read_csv('Sentiment.csv')

# Keep only the neccessary columns
data = data[['text','sentiment']]
print(data.head())
```

3a. Pre-process Data: Simplify

Python

```
# Remove tweets that are neutral
data = data[data.sentiment != "Neutral"]

# Convert text to lower case without punctuation
data['text'] = data['text'].apply(lambda x: x.lower())
data['text'] = data['text'].apply((lambda x: 
re.sub('[^a-zA-z0-9\s]','',x)))

# Display number of positive & negative tweets
print(data[ data['sentiment'] == 'Positive'].size)
print(data[ data['sentiment'] == 'Negative'].size)
```

3b. Pre-process Data: Sequences

Python

```
for idx,row in data.iterrows():
    row[0] = row[0].replace('rt',' ')

max_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, split=' ')
tokenizer.fit_on_texts(data['text'].values)
X = tokenizer.texts_to_sequences(data['text'].values)
X = pad_sequences(X)
```

3c. Pre-process Data: Split

Python

```
Y = pd.get_dummies(data['sentiment']).values
X_train, X_test, Y_train, Y_test =
train_test_split(X,Y, test_size = 0.33, random_state =
42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

4. Define the Model

Python

```
embed_dim = 128
lstm_out = 196

model = Sequential()
model.add(Embedding(max_fatures, embed_dim,input_length
= X.shape[1]))
model.add(SpatialDropout1D(0.4))
model.add(LSTM(lstm_out, dropout=0.2,
recurrent_dropout=0.2))
model.add(Dense(2,activation='softmax'))
```

5. Configure the Learning Process

Python

```
model.compile(loss = 'categorical_crossentropy',
optimizer='adam',metrics = ['accuracy'])
print(model.summary())
```

6. Fit the Model

Python

```
batch_size = 32
model.fit(X_train, Y_train, epochs = 7,
batch_size=batch_size, verbose = 1)
```

7. Apply the Model to Test Data

Python

```
validation_size = 1500

X_validate = X_test[-validation_size:]
Y_validate = Y_test[-validation_size:]
X_test = X_test[:-validation_size]
Y_test = Y_test[:-validation_size]
```

8. Evaluate the Results

Python

```
# Final evaluation of the model
score,acc = model.evaluate(X_test, Y_test, verbose = 2,
batch_size = batch_size)

print("score: %.2f" % (score))
print("acc: %.2f" % (acc))
```

Bonus Step: Apply model to new item

Python

```
twt = ['Meetings: Because none of us is as dumb as all
of us.']

# Pre-process tweet
twt = re.sub('[^a-zA-z0-9\s]','',twt[0].lower())
twt = tokenizer.texts_to_sequences(twt)
twt = pad_sequences(twt, maxlen=28, dtype='int32',
value=0)
```

Bonus Step: Apply model to new item

Python

```
# Run through the model
sentiment = model.predict(twt,batch_size=1,verbose =
2)[0]

#State result
if(np.argmax(sentiment) == 0):
    print("**The tweet is negative.**")
elif (np.argmax(sentiment) == 1):
    print("**The tweet is positive,**")
```

Activity #7: LSTM Program

- Make sure that you can run the CNN code:

Python
`Py_ex7_LSTM.ipynb`

NEED MORE HELP?

Office Hours via Zoom

Tuesdays:	3 pm - 5 pm
Wednesdays:	3 pm – 5 pm
Thursdays:	10 am - noon

Zoom Links are available at https://www.rc.virginia.edu/support/#office-hours

Website:
https://rc.virginia.edu

Email:
hpc-support@virginia.edu

QUESTIONS?