

# Embedded Machine Learning Simple Workflow

Michael Stanley

<http://sensip.asu.edu>

[Mike.Stanley@ieee.org](mailto:Mike.Stanley@ieee.org)

Lecture materials can be accessed at  
[https://github.com/mstanley103/SenSIP\\_RET\\_2021](https://github.com/mstanley103/SenSIP_RET_2021)

# Today's Agenda

- Decision Trees and Ensemble Models
- “Fruit Color” ML Training Example. We will look (in detail) at:
  - Arduino data collection
  - Processing that data into a usable form in Python
  - Using Scikit Learn model generation
  - Evaluating results
  - Using TensorFlow model generation
  - Embedding those models into Arduino code for real-time inferencing
- Things to watch out for when you are creating ML models



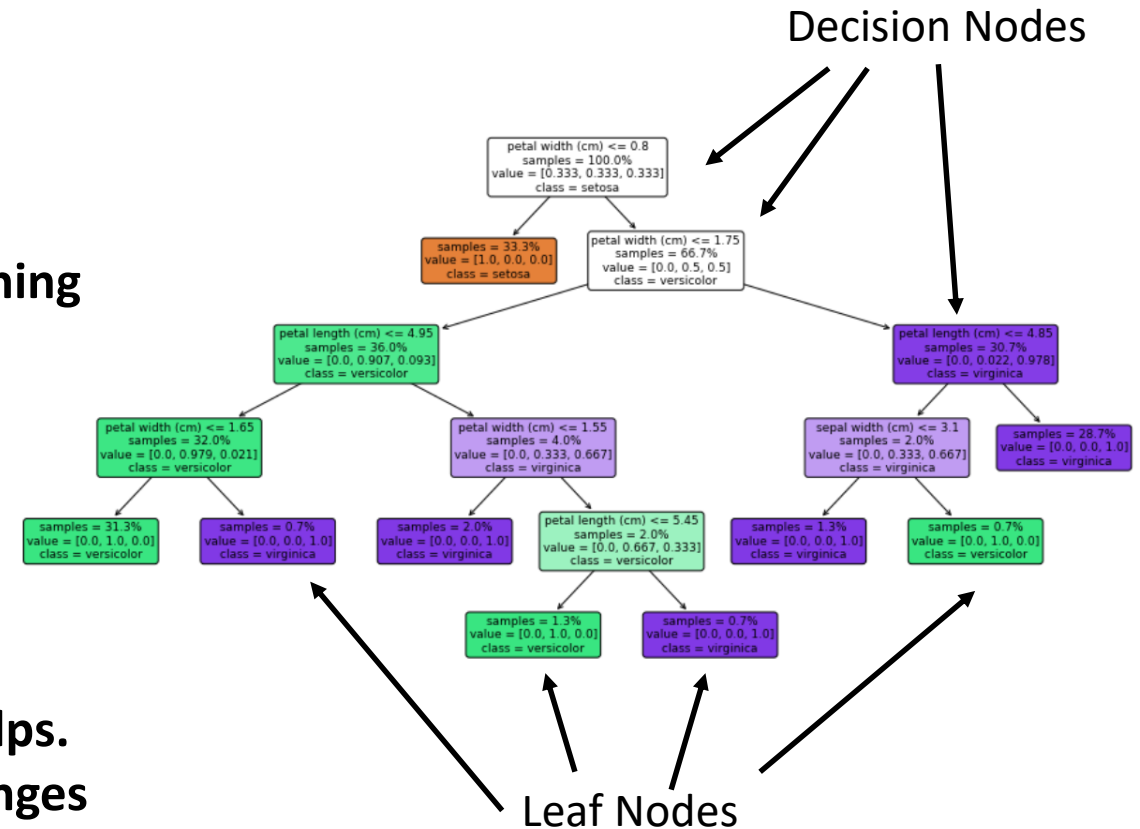
# Decision Trees

## Some Advantages:

- Simple and Intuitive
- Can be used for both classification and regression
- Does not require normalization
- Execution cost is logarithmic to the number of training samples
- Can handle both numerical and categorical data
- Can handle multiple output classes
- White box

## Some Disadvantages:

- Prone to overfitting. Setting a maximum depth helps.
- Small variations in input data can cause major changes in the tree
- Has problems with some relationships (ex: XOR)
- Subject to bias, make sure you balance your dataset!



# Fisher's Iris Data Set

Classes are:



# Iris setosa



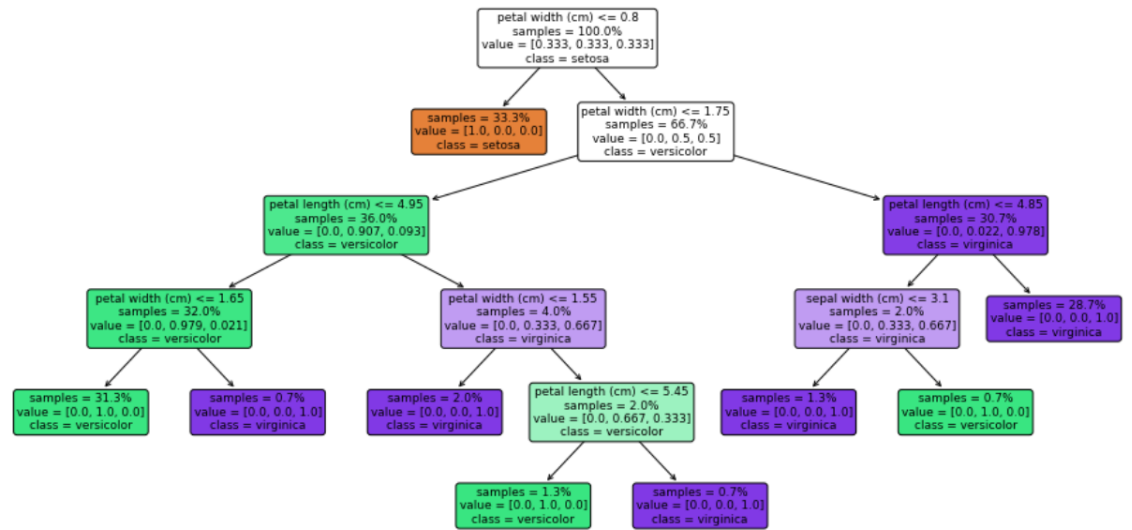
# Iris versicolor



# Iris virginica

Features are:

- sepal length
- sepalwidth
- petal length
- petal width



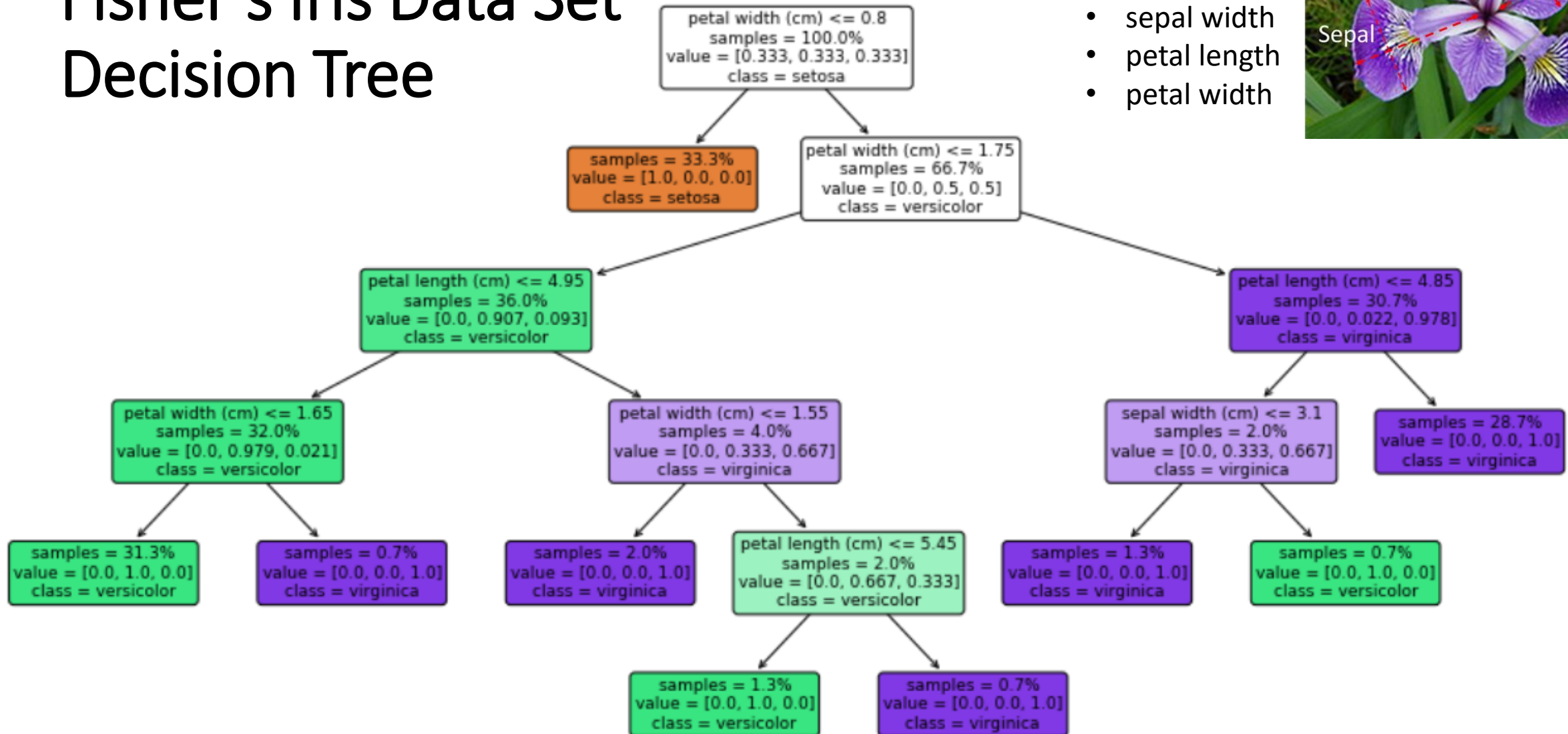
```
from sklearn.datasets import load_iris
from sklearn import tree
import matplotlib.pyplot as plt
```

```
clf = tree.DecisionTreeClassifier(random_state=0)
iris = load_iris()
```

```
clf = clf.fit(iris.data, iris.target)
plt.figure(figsize=(16,8))
annotations=tree.plot_tree(clf, filled=True,
rounded=True,
    feature_names=iris.feature_names,
    class_names=iris.target_names,
    impurity=False, proportion=True)
```

# Fisher's Iris Data Set Decision Tree

- Features are:
- sepal length
  - sepal width
  - petal length
  - petal width





# Ensembles

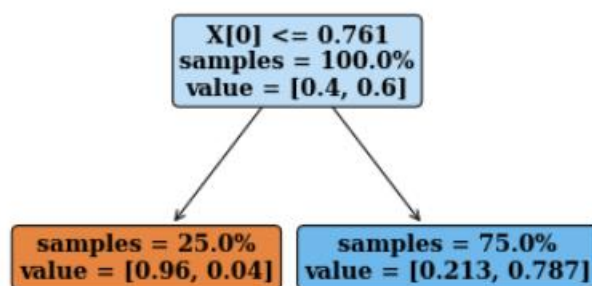
- Ensembles improve both generalization and accuracy by using a collection of simple models to collectively predict class membership.
- “**Bagging**” refers to using a set of lower level models which are each individually trained on a different subset of the training data, and then using a voting scheme to determine class membership. Each sub-model is independent of the other sub-models.
- “**Boosting**” techniques train sub-models sequentially. Errors in each sub-model are given higher weighting in the subsequent sub-model, with the process repeating. Predictions are combined via a weighted majority-vote scheme to produce the final result.



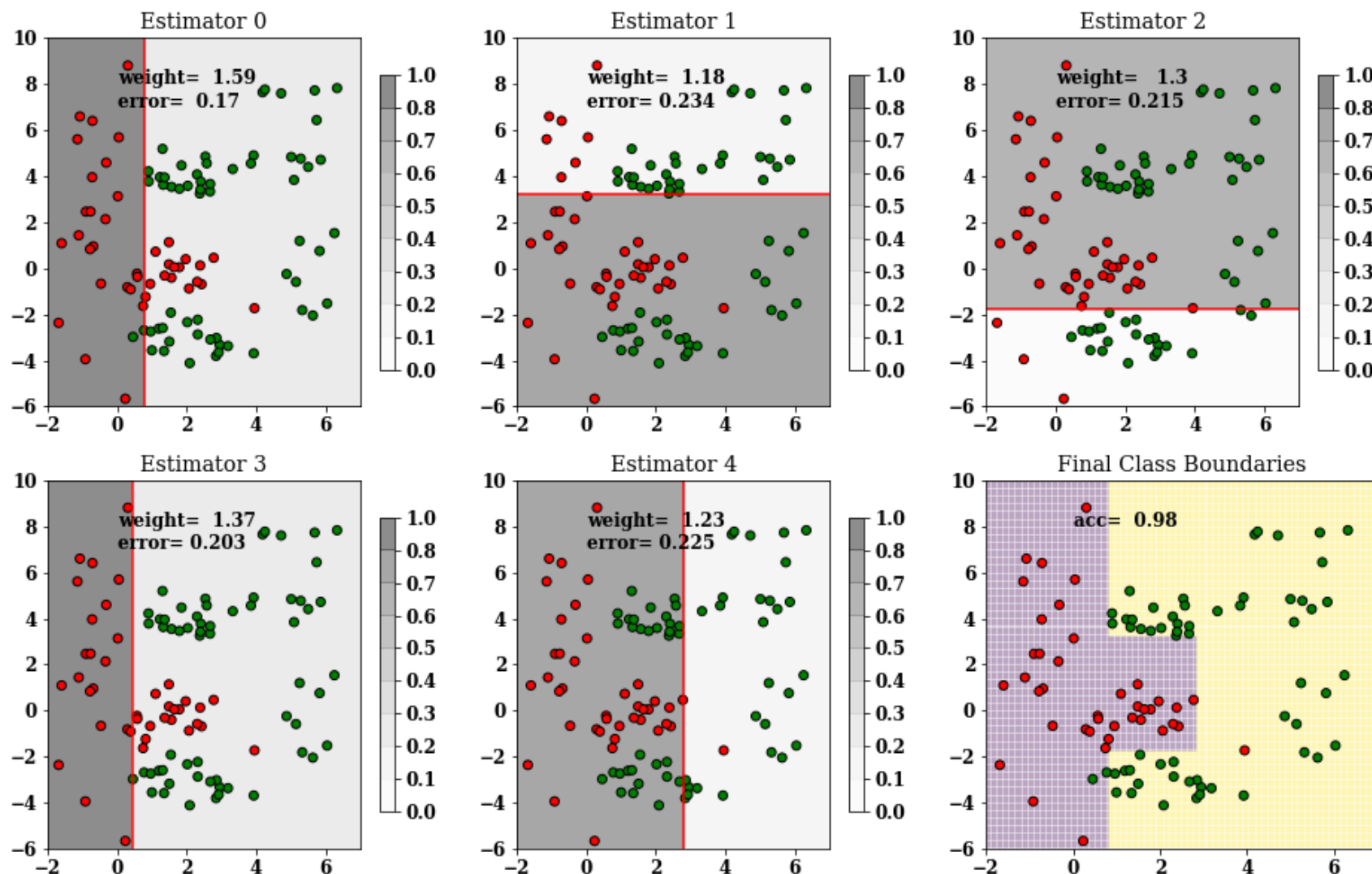
Random Forests are an example of bagging. Each individual tree is limited in depth to encourage generalization. Accuracy is improved by utilizing average results from the collective.

# AdaBoost

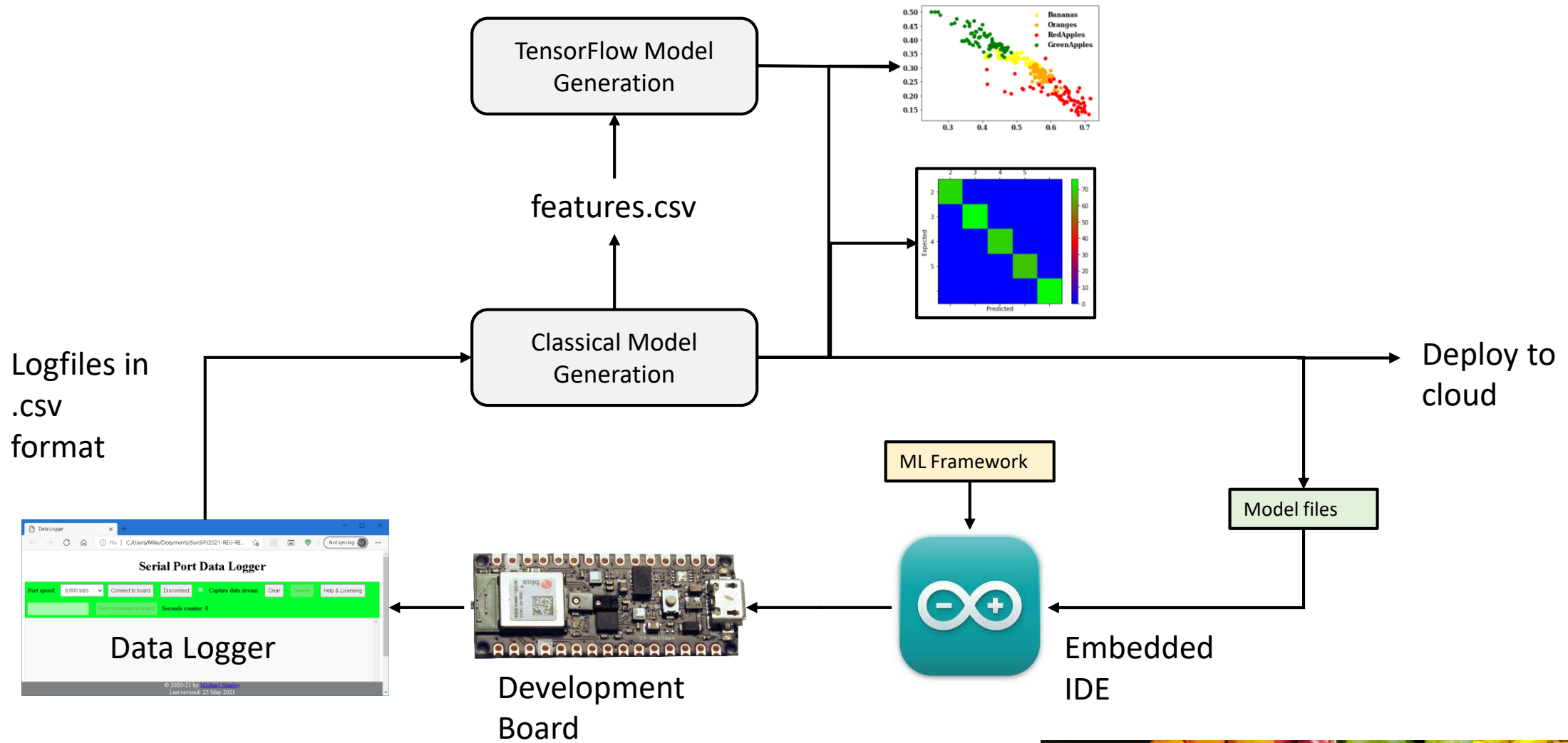
AdaBoost is an example of boosting. Here, we use a collection of crude “stub models” to create a final ensemble model with 98% accuracy on training data.



Example stub model



# ML Modeling Example 1 Workflow





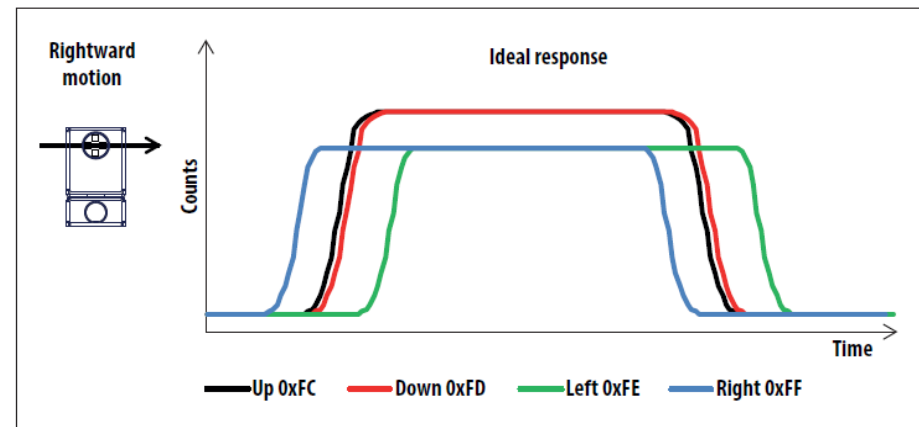
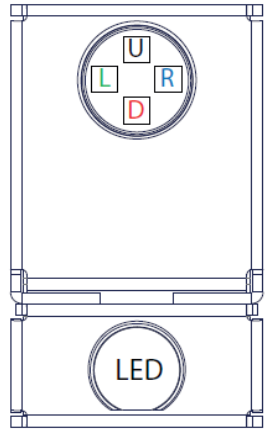
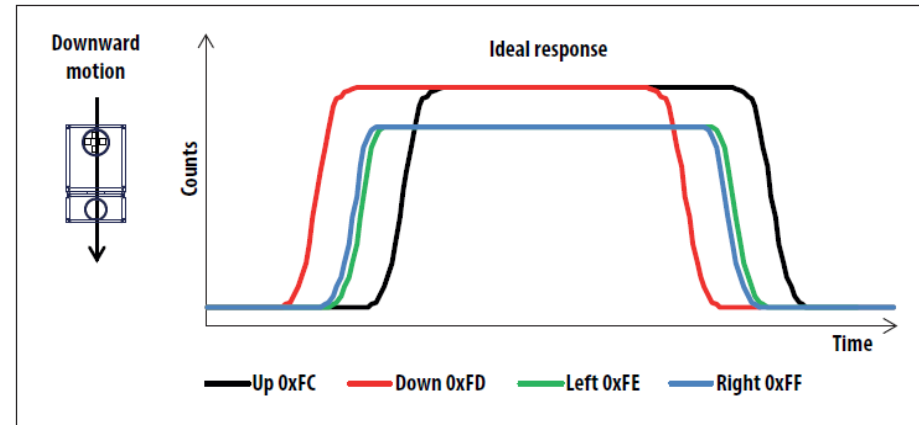
# Light Sensor APDS-9960



[Source: mouser.com](https://www.mouser.com)

Clever state machines coupled with on-chip LED and light sensitive diodes enable one device to provide sensor readings for:

- Proximity (distance)
- Ambient light
- RGB color mix
- Gesture detection



Source: APDS-9960 Datasheet

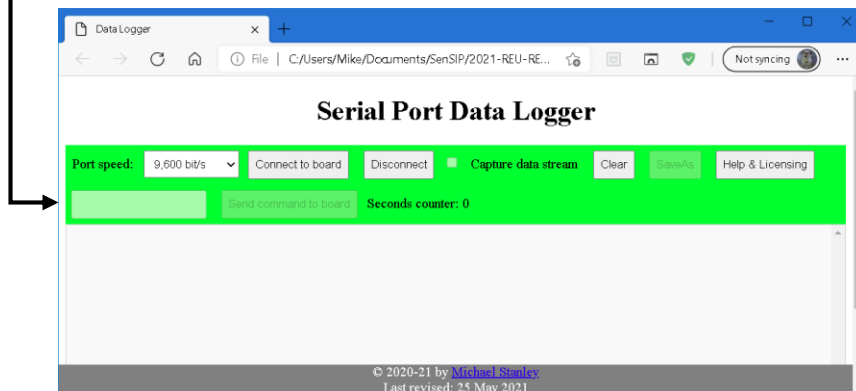
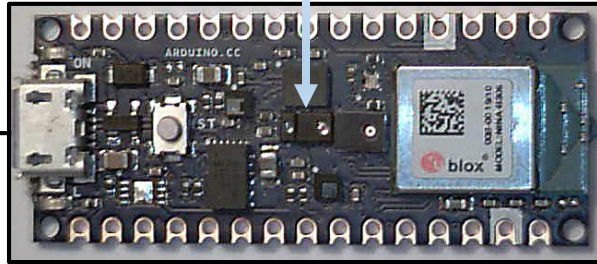
# Embedded Data Logger Code

```
#include <Wire.h>
#include <Arduino_APDS9960.h>
int sampleNum=0;
void setup() {
    Serial.begin(9600);
    while (!Serial);

    if (!APDS.begin()) {
        Serial.println("Error initializing APDS9960 sensor.");
    }
    // Turn on White LED light source
    digitalWrite(LED_R, LOW);
    digitalWrite(LED_G, LOW);
    digitalWrite(LED_B, LOW);
}
```

```
void loop() {
    sampleNum++;
    // check if a color reading is available
    while (!APDS.colorAvailable()) {
        delay(5);
    }
    int r, g, b, c;
    // read the color
    APDS.readColor(r, g, b, c);
    // print the values
    Serial.print(r);
    Serial.print(",");
    Serial.print(g);
    Serial.print(",");
    Serial.print(b);
    Serial.print(",");
    Serial.print(c);
    Serial.print(",");
    Serial.println(sampleNum);
    // wait a bit before reading again
    delay(500);
}
```

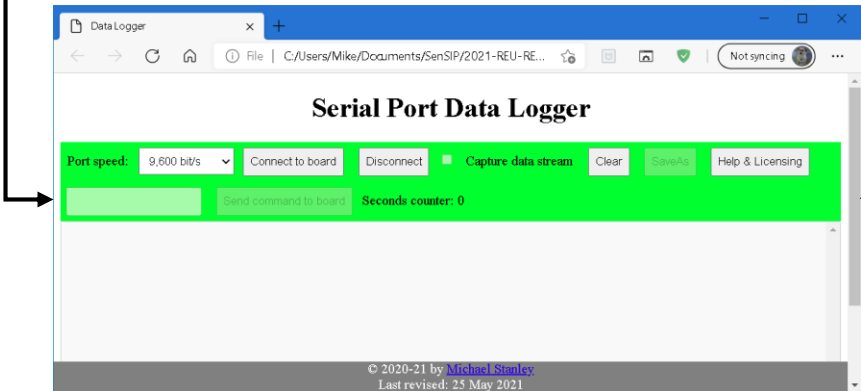
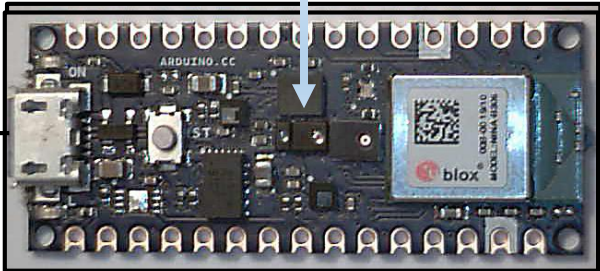
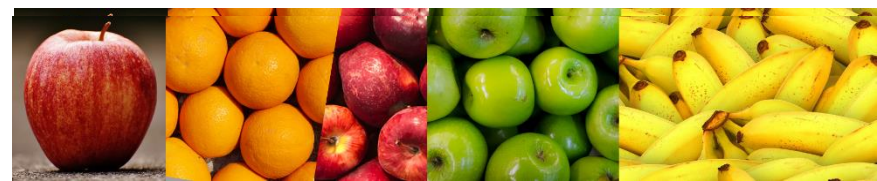
# Data Collection



loggerSample#, R, G, B, C, embeddedSample#

1	33	22	16	67	9926
2	33	22	17	69	9927
3	34	23	18	71	9928
4	37	25	18	75	9929
5	47	33	24	97	9930
6	47	33	24	98	9931
7	48	34	24	101	9932
8	50	36	26	106	9933
9	36	24	17	73	9934
10	41	28	20	83	9935
11	43	29	22	89	9936
12	45	31	23	93	9937

# Data Collection



RedApples.csv

```
1, 33,22,16,67,9926
2, 33,22,17,69,9927
3, 34,23,18,71,9928
4, 37,25,18,75,9929
5, 47,33,24,97,9930
6, 47,33,24,98,9931
7, 48,34,24,101,9932
8, 50,36,26,106,9933
9, 36,24,17,73,9934
10, 41,28,20,83,9935
11, 43,29,22,89,9936
12, 45,31,23,93,9937
```

GreenApples.csv

```
1, 33,22,16,67,9926
2, 33,22,17,69,9927
3, 34,23,18,71,9928
4, 37,25,18,75,9929
5, 47,33,24,97,9930
6, 47,33,24,98,9931
7, 48,34,24,101,9932
8, 50,36,26,106,9933
9, 36,24,17,73,9934
10, 41,28,20,83,9935
11, 43,29,22,89,9936
12, 45,31,23,93,9937
```

bananas.csv

```
1, 33,22,16,67,9926
2, 33,22,17,69,9927
3, 34,23,18,71,9928
4, 37,25,18,75,9929
5, 47,33,24,97,9930
6, 47,33,24,98,9931
7, 48,34,24,101,9932
8, 50,36,26,106,9933
9, 36,24,17,73,9934
10, 41,28,20,83,9935
11, 43,29,22,89,9936
12, 45,31,23,93,9937
```

oranges.csv

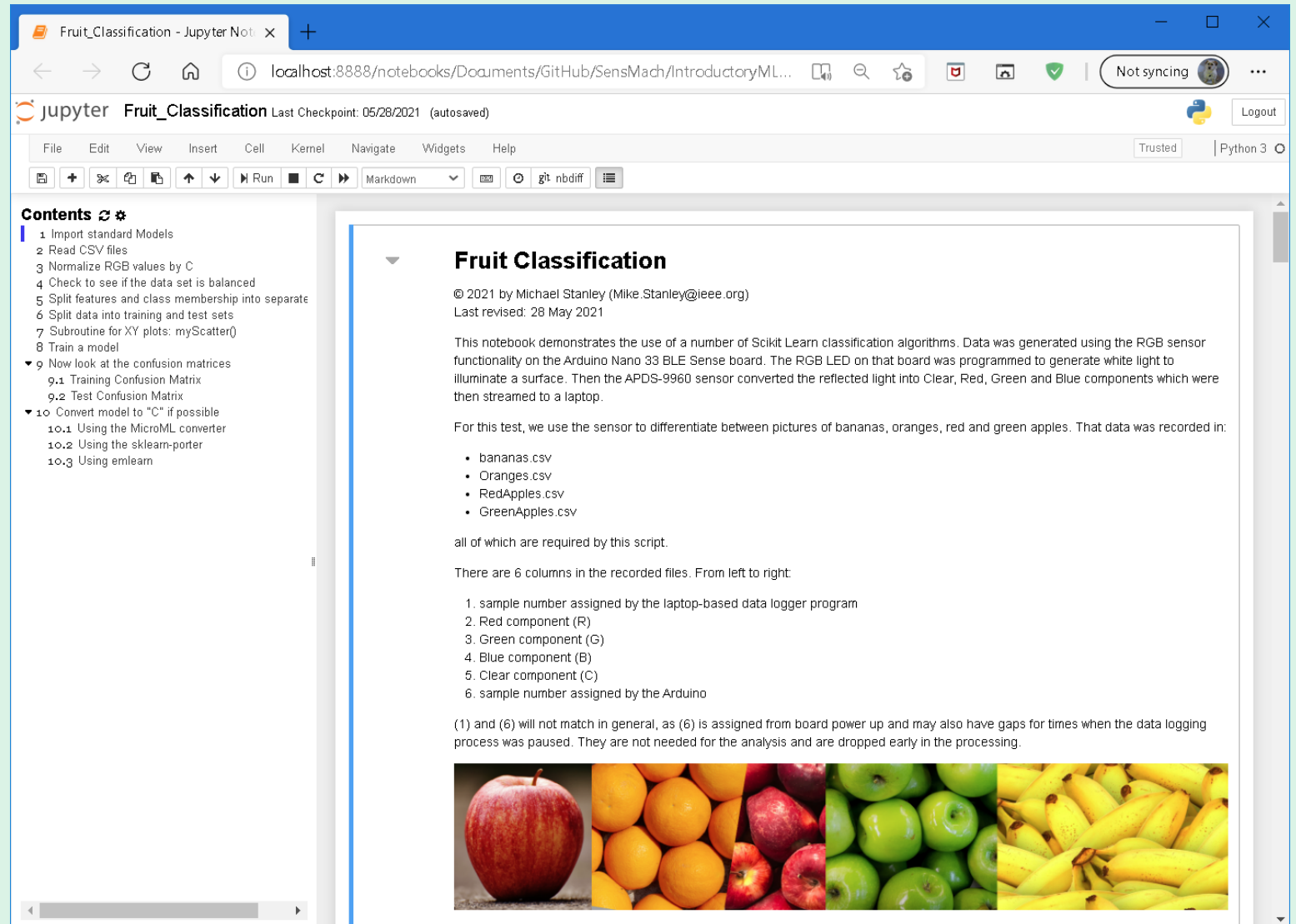
Python / Jupyter  
Notebook

Fruit\_  
Classification.ipynb



# Walkthroughs

- Jupyter Notebook for Fruit Classification using classical ML
- Embedded Implementation Code Review



Fruit\_Classification - Jupyter Notebook

localhost:8888/notebooks/Documents/GitHub/SensMach/IntroductoryML...

jupyter Fruit\_Classification Last Checkpoint: 05/28/2021 (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help

Contents

- 1 Import standard Models
- 2 Read CSV files
- 3 Normalize RGB values by C
- 4 Check to see if the data set is balanced
- 5 Split features and class membership into separate
- 6 Split data into training and test sets
- 7 Subroutine for XY plots: myScatter()
- 8 Train a model
- 9 Now look at the confusion matrices
  - 9.1 Training Confusion Matrix
  - 9.2 Test Confusion Matrix
- 10 Convert model to "C" if possible
  - 10.1 Using the MicroML converter
  - 10.2 Using the sklearn-porter
  - 10.3 Using emlearn

## Fruit Classification

© 2021 by Michael Stanley (Mike.Stanley@ieee.org)  
Last revised: 28 May 2021

This notebook demonstrates the use of a number of Scikit Learn classification algorithms. Data was generated using the RGB sensor functionality on the Arduino Nano 33 BLE Sense board. The RGB LED on that board was programmed to generate white light to illuminate a surface. Then the APDS-9960 sensor converted the reflected light into Clear, Red, Green and Blue components which were then streamed to a laptop.

For this test, we use the sensor to differentiate between pictures of bananas, oranges, red and green apples. That data was recorded in:


- bananas.csv
- Oranges.csv
- RedApples.csv
- GreenApples.csv

all of which are required by this script.

There are 6 columns in the recorded files. From left to right:

- sample number assigned by the laptop-based data logger program
- Red component (R)
- Green component (G)
- Blue component (B)
- Clear component (C)
- sample number assigned by the Arduino

(1) and (6) will not match in general, as (6) is assigned from board power up and may also have gaps for times when the data logging process was paused. They are not needed for the analysis and are dropped early in the processing.



# Embedded ML Application

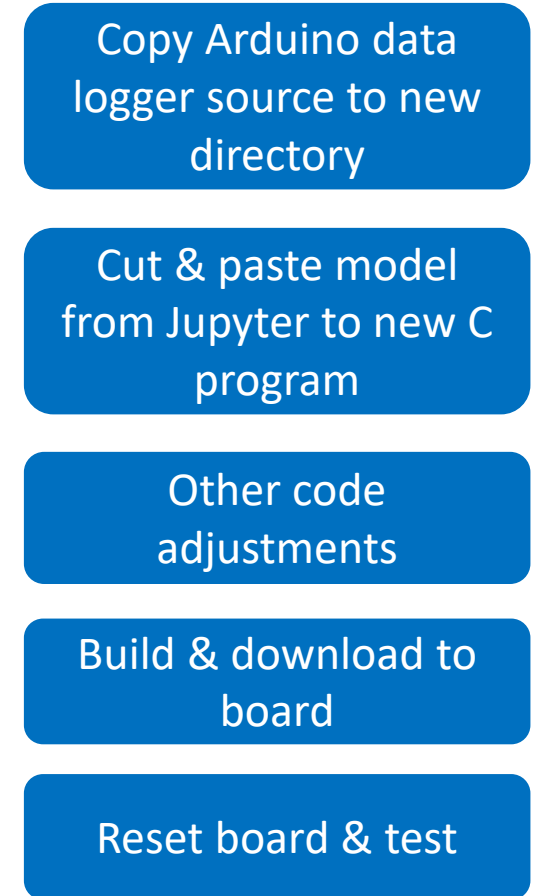
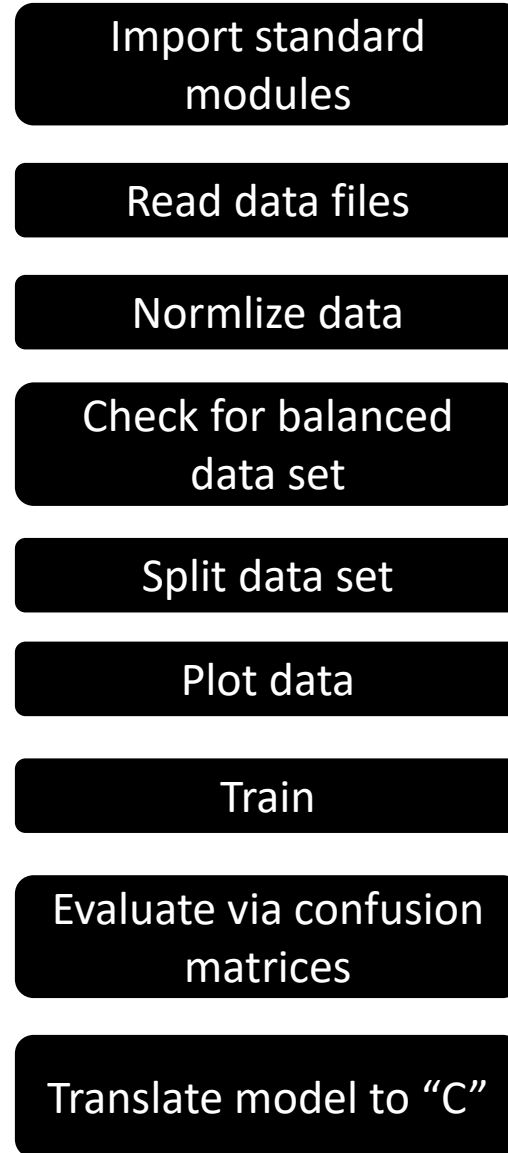
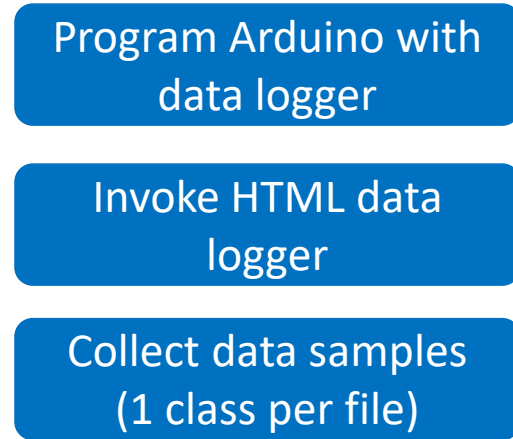
```
#include <Wire.h>
#include <math.h>
#include <Arduino_APDS9960.h>
int predict(float features[3]) {
    ... details omitted ...
}

void setup() {
    Serial.begin(9600);
    while (!Serial);

    if (!APDS.begin()) {
        Serial.println("Error initializing APDS9960 sensor.");
    }
    // Turn on White LED light source
    digitalWrite(LED_R, LOW);
    digitalWrite(LED_G, LOW);
    digitalWrite(LED_B, LOW);
}
```

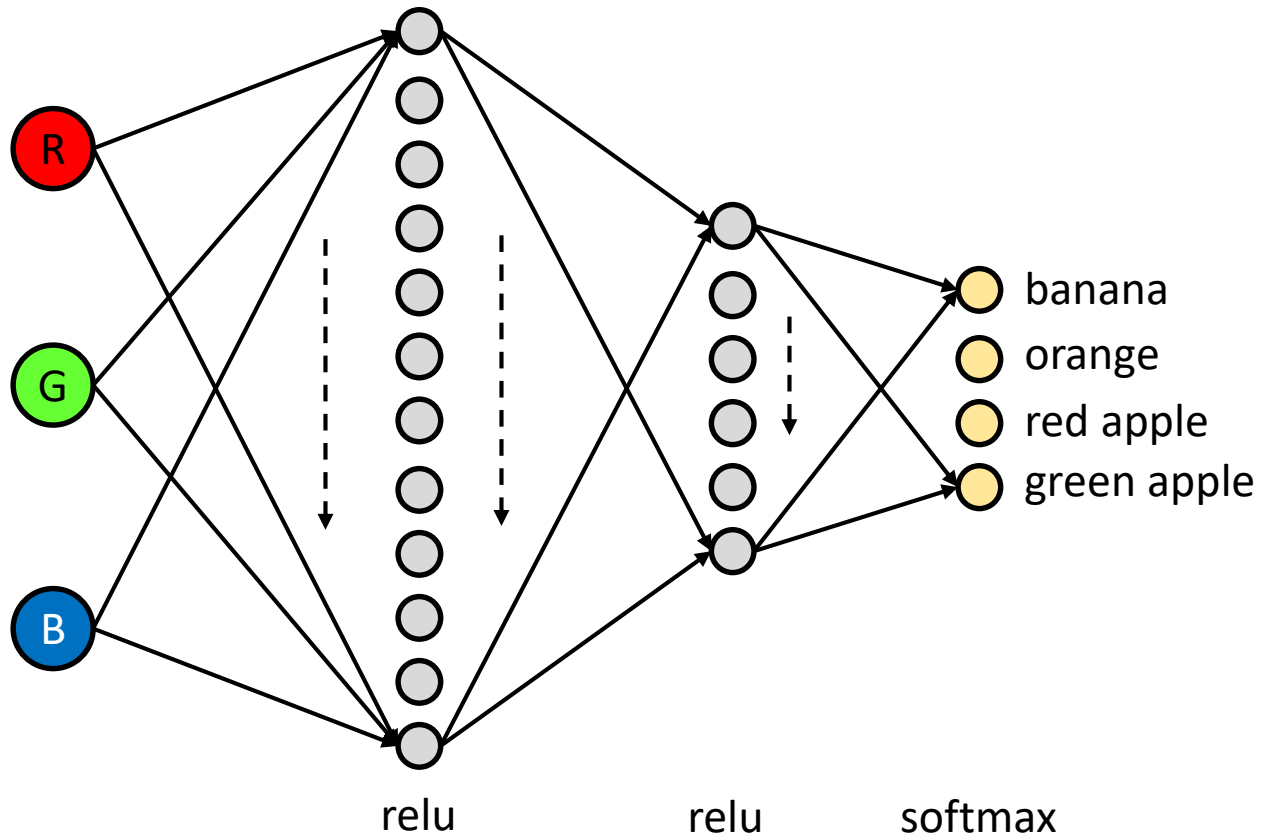
```
void loop() {
    int r, g, b, c;
    float features[3];
    int result;
    // check if a color reading is available
    while (!APDS.colorAvailable()) {
        delay(5);
    }
    // read the color
    APDS.readColor(r, g, b, c);
    features[0]=(float) r / (float) c;
    features[1]=(float) g / (float) c;
    features[2]=(float) b / (float) c;
    result = predict(features);
    switch (result) {
        case 0:
            Serial.println("Banana");
            break;
        case 1:
            ... details omitted ...
    }
    // wait a bit before reading again
    delay(500);
}
```

# What we did



Steps shown in black were completed in Jupyter

# Neural Network to be generated via Keras



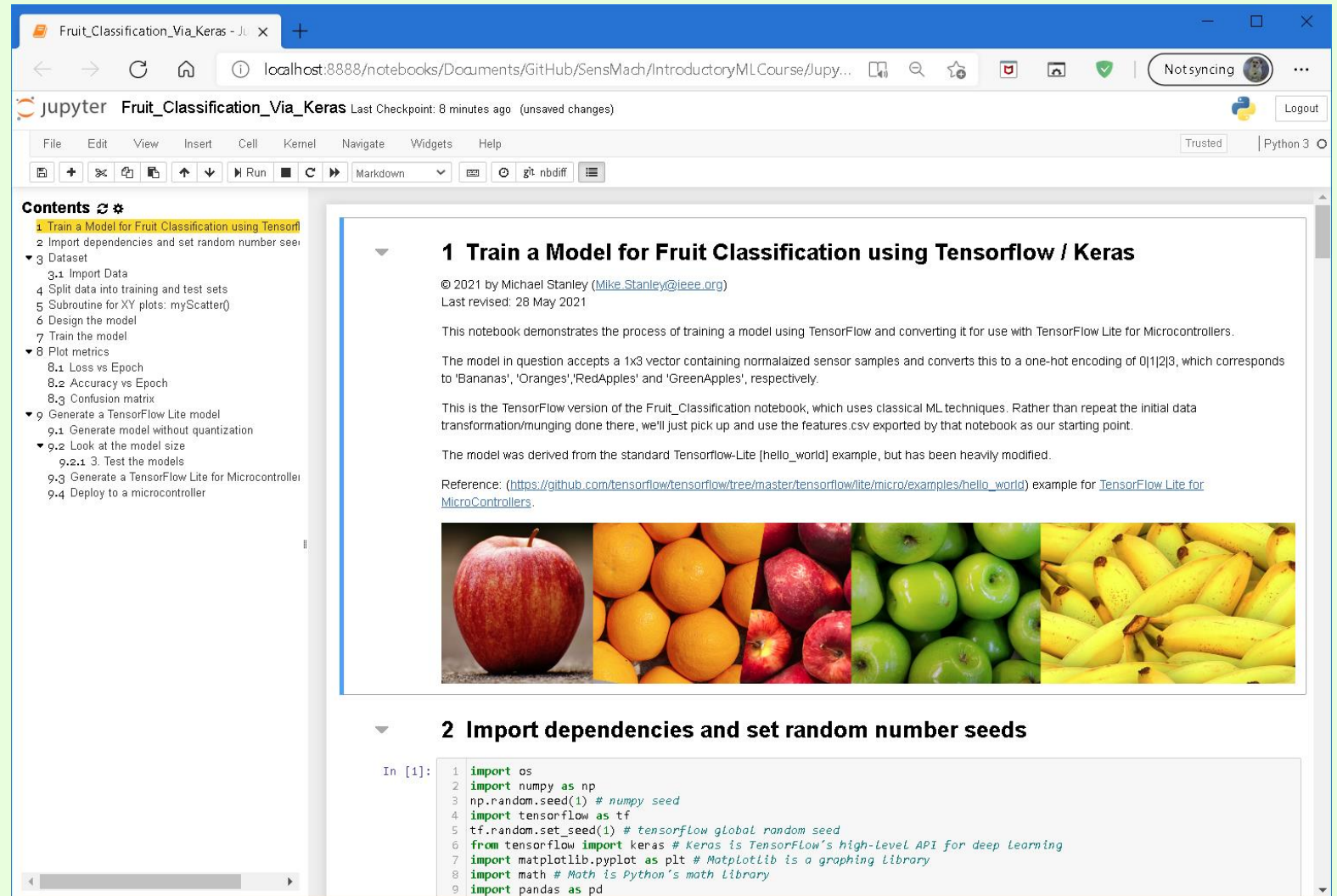
- Why this configuration? Because it worked. We tried several.
- All layers are fully connected
- The categorical final layer needed to be represented by 4 softmax outputs (differs from scikit-learn)

[Reference: The Sequential model \(keras.io\)](https://keras.io)



# Walkthroughs

- Jupyter Notebook for Fruit Classification using Keras
- Embedded Implementation Code Review



The screenshot displays a Jupyter Notebook interface in a web browser. The browser's address bar shows the URL `localhost:8888/notebooks/Documents/GitHub/SensMach/IntroductoryMLCourse/jupy...`. The notebook's title bar reads "Fruit\_Classification\_Via\_Keras" with a status of "Last Checkpoint: 8 minutes ago (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help) and a toolbar with icons for file operations, running cells, and markdown editing.

**Contents**

- 1 Train a Model for Fruit Classification using Tensorflow / Keras
- 2 Import dependencies and set random number seeds
- 3 Dataset
  - 3.1 Import Data
- 4 Split data into training and test sets
- 5 Subroutine for XY plots: myScatter()
- 6 Design the model
- 7 Train the model
- 8 Plot metrics
  - 8.1 Loss vs Epoch
  - 8.2 Accuracy vs Epoch
  - 8.3 Confusion matrix
- 9 Generate a TensorFlow Lite model
  - 9.1 Generate model without quantization
  - 9.2 Look at the model size
    - 9.2.1 3. Test the models
  - 9.3 Generate a TensorFlow Lite for Microcontroller
  - 9.4 Deploy to a microcontroller

**1 Train a Model for Fruit Classification using Tensorflow / Keras**

© 2021 by Michael Stanley ([Mike.Stanley@ieee.org](mailto:Mike.Stanley@ieee.org))  
Last revised: 28 May 2021


This notebook demonstrates the process of training a model using TensorFlow and converting it for use with TensorFlow Lite for Microcontrollers.

The model in question accepts a 1x3 vector containing normalized sensor samples and converts this to a one-hot encoding of 0|1|2|3, which corresponds to 'Bananas', 'Oranges', 'RedApples' and 'GreenApples', respectively.

This is the TensorFlow version of the Fruit\_Classification notebook, which uses classical ML techniques. Rather than repeat the initial data transformation/munging done there, we'll just pick up and use the features.csv exported by that notebook as our starting point.

The model was derived from the standard TensorFlow-Lite [hello\_world] example, but has been heavily modified.

Reference: ([https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/hello\\_world](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/micro/examples/hello_world)) example for TensorFlow Lite for MicroControllers.



**2 Import dependencies and set random number seeds**

```
In [1]: 1 import os
2 import numpy as np
3 np.random.seed(1) # numpy seed
4 import tensorflow as tf
5 tf.random.set_seed(1) # tensorflow global random seed
6 from tensorflow import keras # Keras is TensorFlow's high-level API for deep learning
7 import matplotlib.pyplot as plt # Matplotlib is a graphing library
8 import math # Math is Python's math library
9 import pandas as pd
```

# Files used in this reference

Filename(s)	Description
loggerV4.html / helpV3.html	Simple HTML/Javascript based application to record data sent by Arduino
ColorSensor.ino	Embedded code for Arduino data logger
ColorSensorInferencing.ino	Embedded code using decision tree model
ColorSensorKerasInferencing/*.*	Embedded code using Tensorflow neural net
Fruit_Classification.ipynb	Jupyter notebook for DT model generation
Fruit_Classification_Via_Keras.ipynb	Jupyter notebook for Tensorflow model generation
FruitFiles/*.csv	Raw data files captured by Mike



**If time allows...**

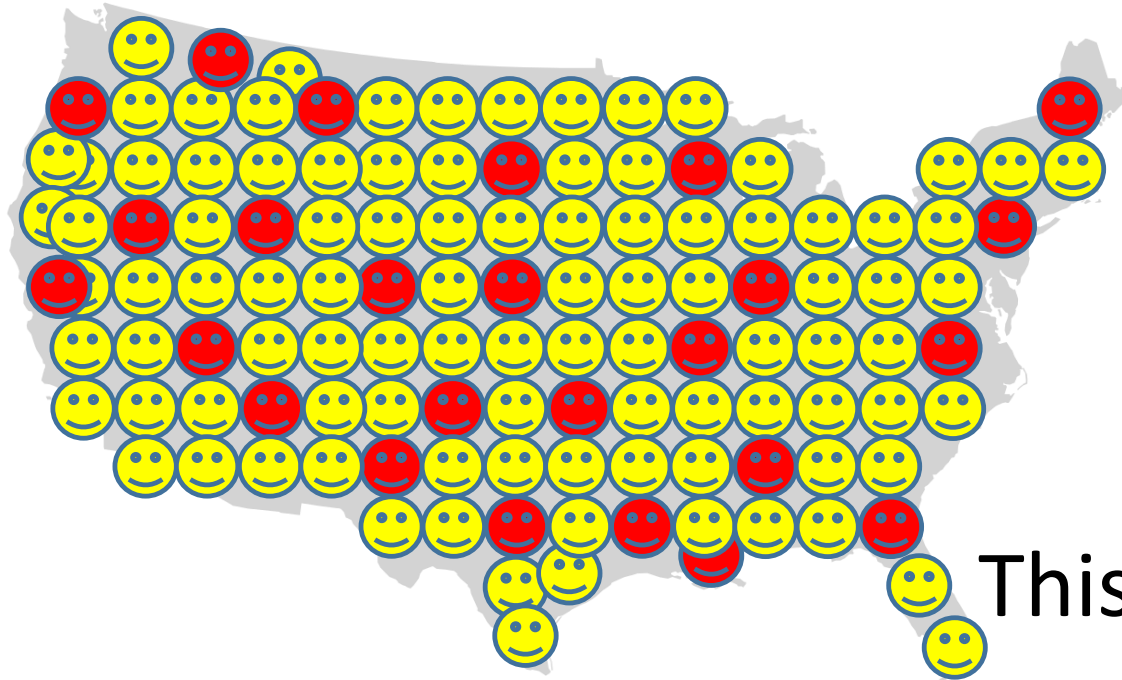
# Some Terminology

## Out-of-Sample

refers to the entire population – which may be infinite or uncountable

## In-Sample

refers to the a finite sample drawn from the larger population



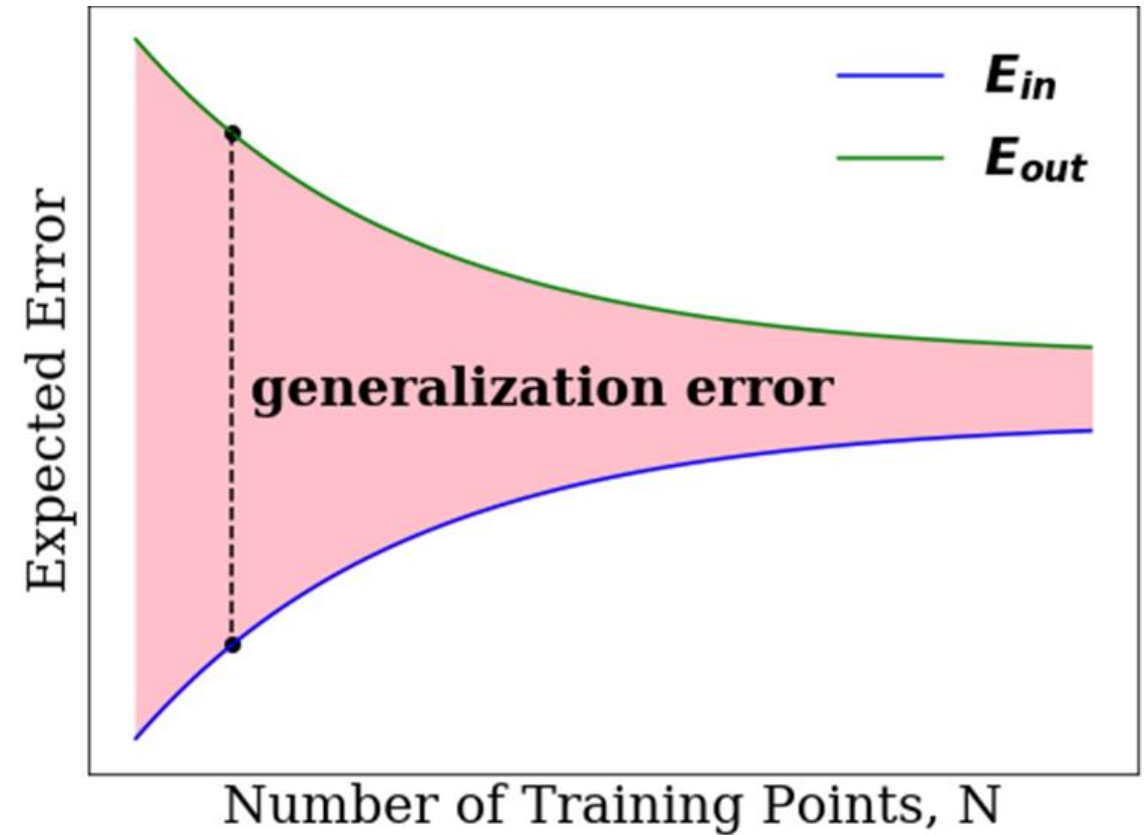
This you know

This you can only approximate



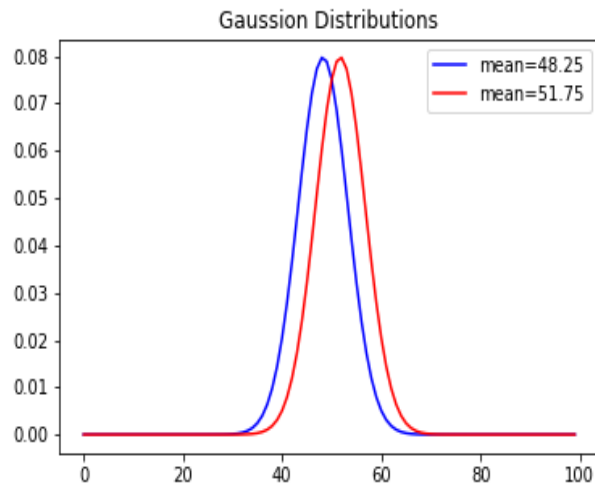
# ML Goals

- ML problems can be thought of as generating a model for a population that obeys an unknown probability distribution.
- The goal is to generate a model that operates as well on Out-of-Sample (new) data as it does on In-Sample (training) data.
- If the expected error of the model on out-of-sample (testing) data is approximately the same as the error found when using training data, then we say that the model “**generalizes well**”
- Adding additional samples to your training is always helpful.



# Sampling Bias

Your training data must be representative of the general population. Otherwise any model generated from it will include the same sampling bias.



OCT. 18, 2018, AT 11:54 AM

## Clinton-Trump Probably Won't Be The Next 'Dewey Defeats Truman'

By [Harry Enten](#)

Filed under [2018 Election](#)



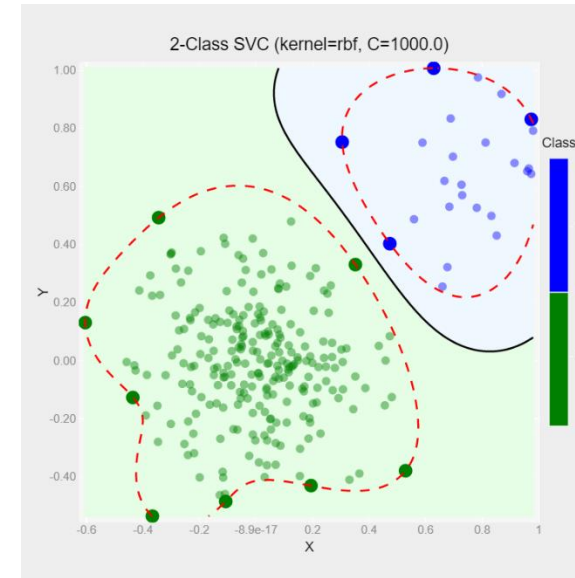
President Harry S. Truman gleefully displays an early edition of the Chicago Daily Tribune from his train in St. Louis after his defeat of Thomas E. Dewey in the 1948 presidential election. GETTY IMAGES

Source: <https://fivethirtyeight.com/features/clinton-trump-probably-wont-be-the-next-dewey-defeats-truman/>

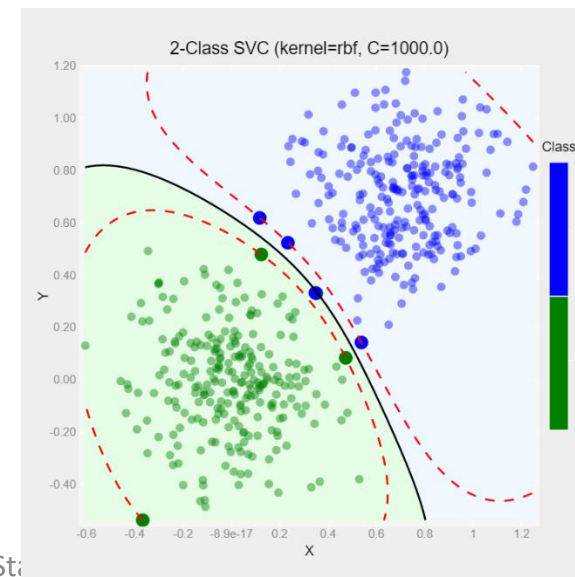
# Effects of Unbalanced Training Sets

- It is generally a good idea to have a “balanced” training set where the number of samples for each class is the same.
- Some machine learning algorithms have the ability to apply weights to help offset the effect of an unbalanced training set.
- “One class” algorithms only require data for the primary class for training.

The examples to the right both use identical probability distributions for the input data, but the first example is unbalanced.



250 class 0  
samples, 25 class  
1 samples



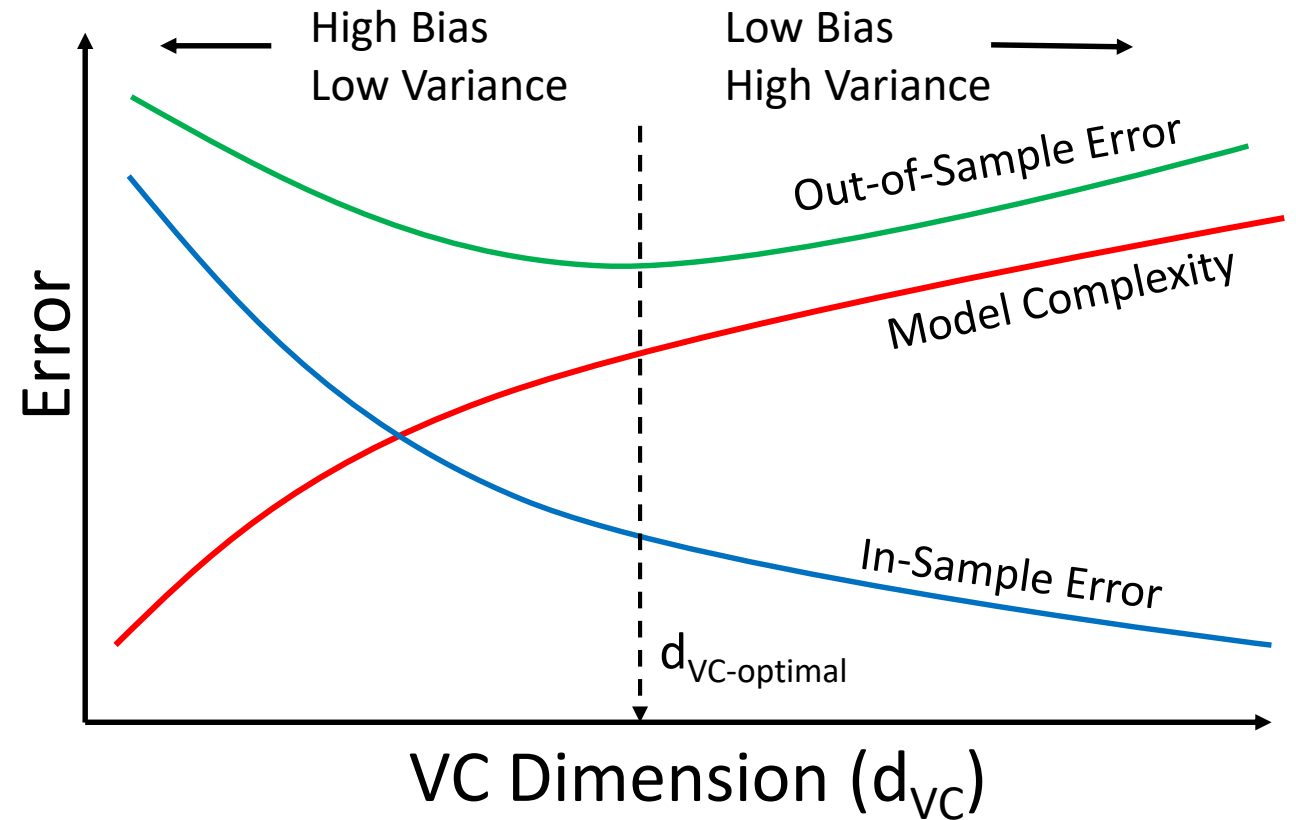
250 samples of  
both classes

# The Curse of Dimensionality

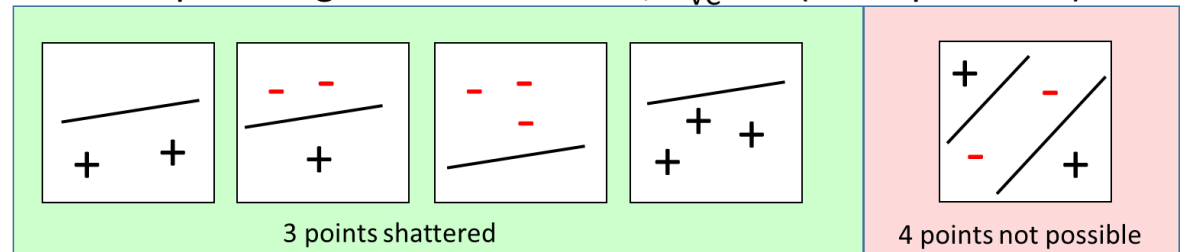
**Rule of thumb:**  
**use  $N \geq 10 d_{VC}$**

where  $N$ =number of samples required for training.

Without going into details...  $d_{VC}$  refers to the maximum number of points which can be properly shattered (classified) by a machine learning hypothesis



An example using a linear classifier,  $d_{VC} = 3$  (breakpoint = 4)





# So what's the point?

- More complex models (with higher  $d_{VC}$ ) require more data to train.
  - If models become too complex, they may not generalize well.
  - Your training data must accurately reflect the same probability distribution as the overall population, or your model will exhibit bias.
  - Make sure you have a balanced training set or take steps to account for an unbalanced one.
  - More training samples is always better
- 
- Basically, it's easy to mis-use machine learning techniques. Take the time to learn a bit about some of the issues raised here.

For  
further  
study

CALIFORNIA INSTITUTE OF TECHNOLOGY



# LEARNING FROM DATA

Machine Learning course - recorded at a live broadcast from Caltech

## HIGHLIGHTS

**NEW: SECOND TERM OF THE COURSE PREDICTS COVID-19 TRAJECTORY.**

A real Caltech course, not a watered-down version

**7 Million Views**  
ON YOUTUBE & ITUNES

**Article about the course in**



- **Free**, introductory *Machine Learning* online course (MOOC)
- Taught by Caltech Professor Yaser Abu-Mostafa [[article](#)]
- [Lectures](#) recorded from a live broadcast, including Q&A
- Prerequisites: [Basic](#) probability, matrices, and calculus
- 8 homework sets and a final exam

[[Home](#)]

[The lectures](#)

[Homework](#)

[Textbook](#)

[Forum](#)

[The instructor](#)

[Contact](#)

 **Tweet**

 **Follow**

[terms and conditions](#)

