Embedded Machine Learning Simple Workflow

Michael Stanley

http://sensip.asu.edu

Mike.Stanley@ieee.org

Lecture materials can be accessed at
https://github.com/mstanley103/SenSIP_RET_2021

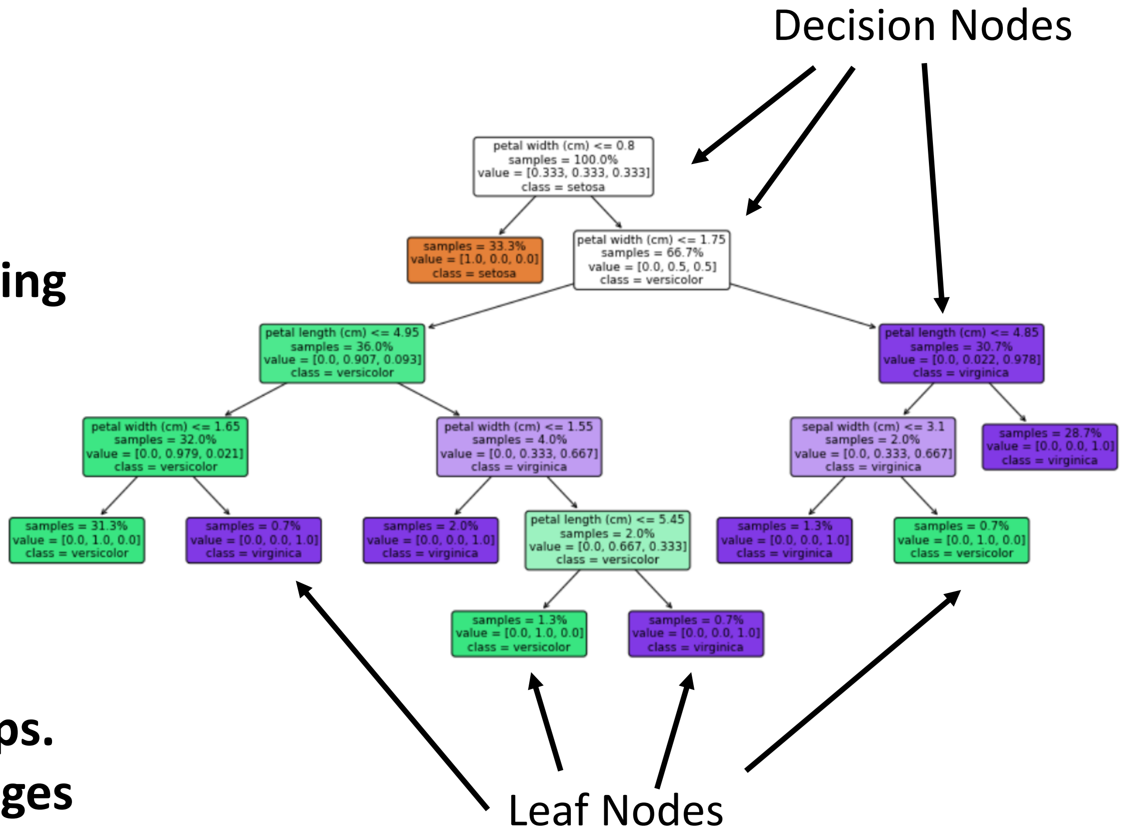
Today’s Agenda

- 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 outfor when you are creating ML models

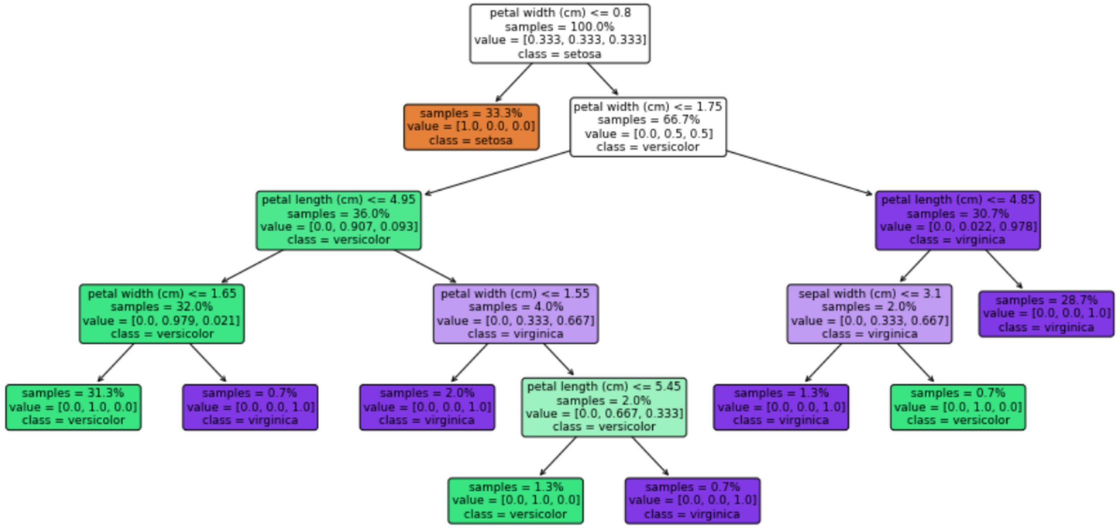
Decision Trees

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:

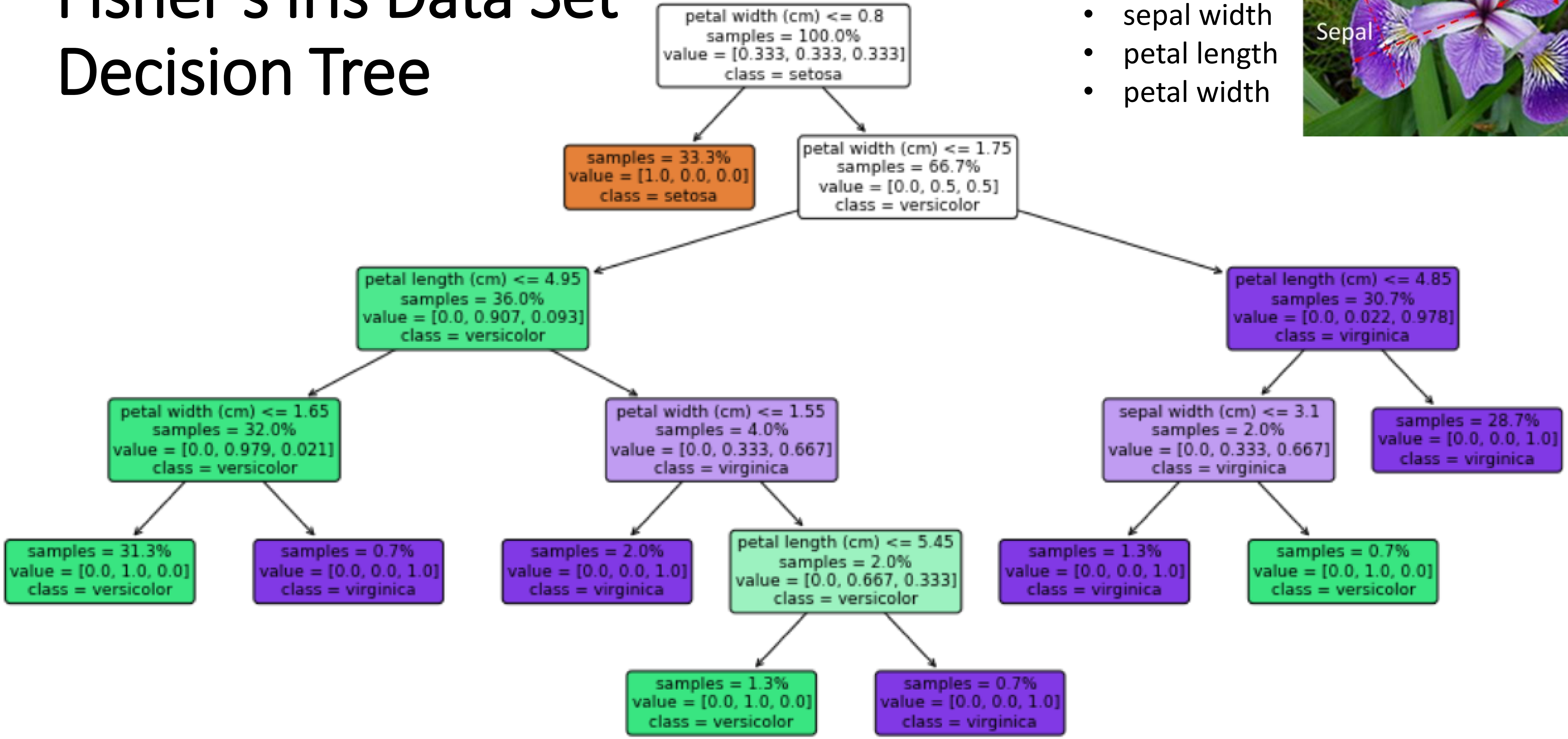
- sepal length
- sepalwidth
- petal length
- petal width


```
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

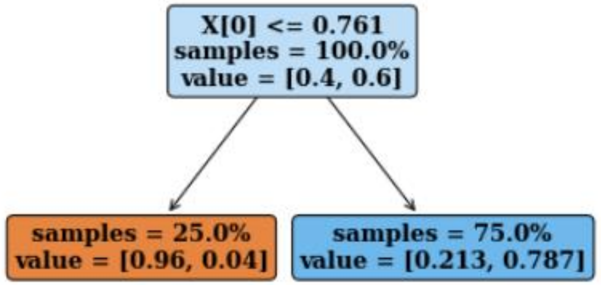
Ensembles

- Ensembles improve both generalization and accuracy by using the 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.

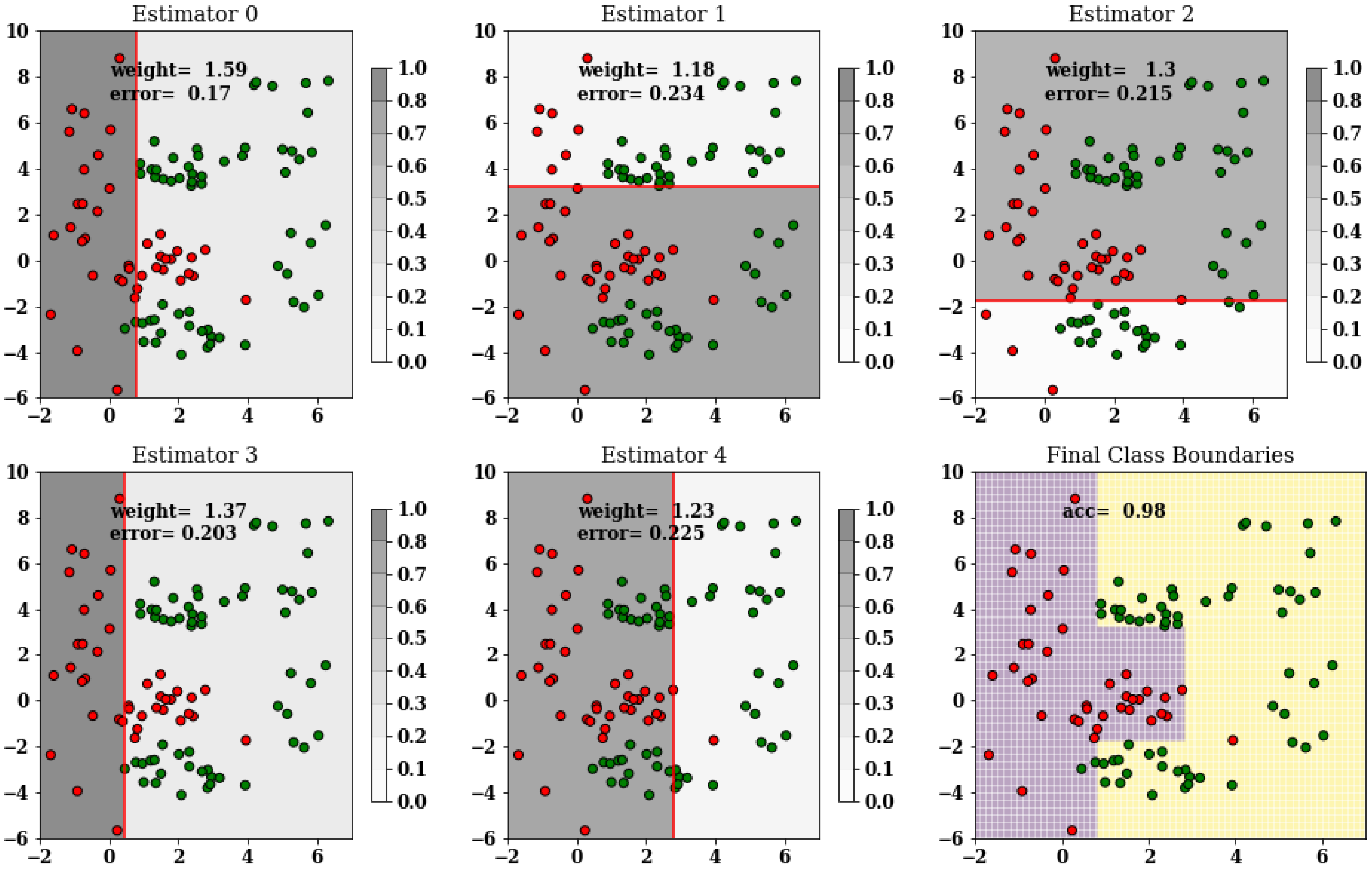
Random Forests are an example of bagging. Each individual tree is limited in depth to encourage generalization. Accuracy is improved by utilizing average results from the collective.

AdaBoost

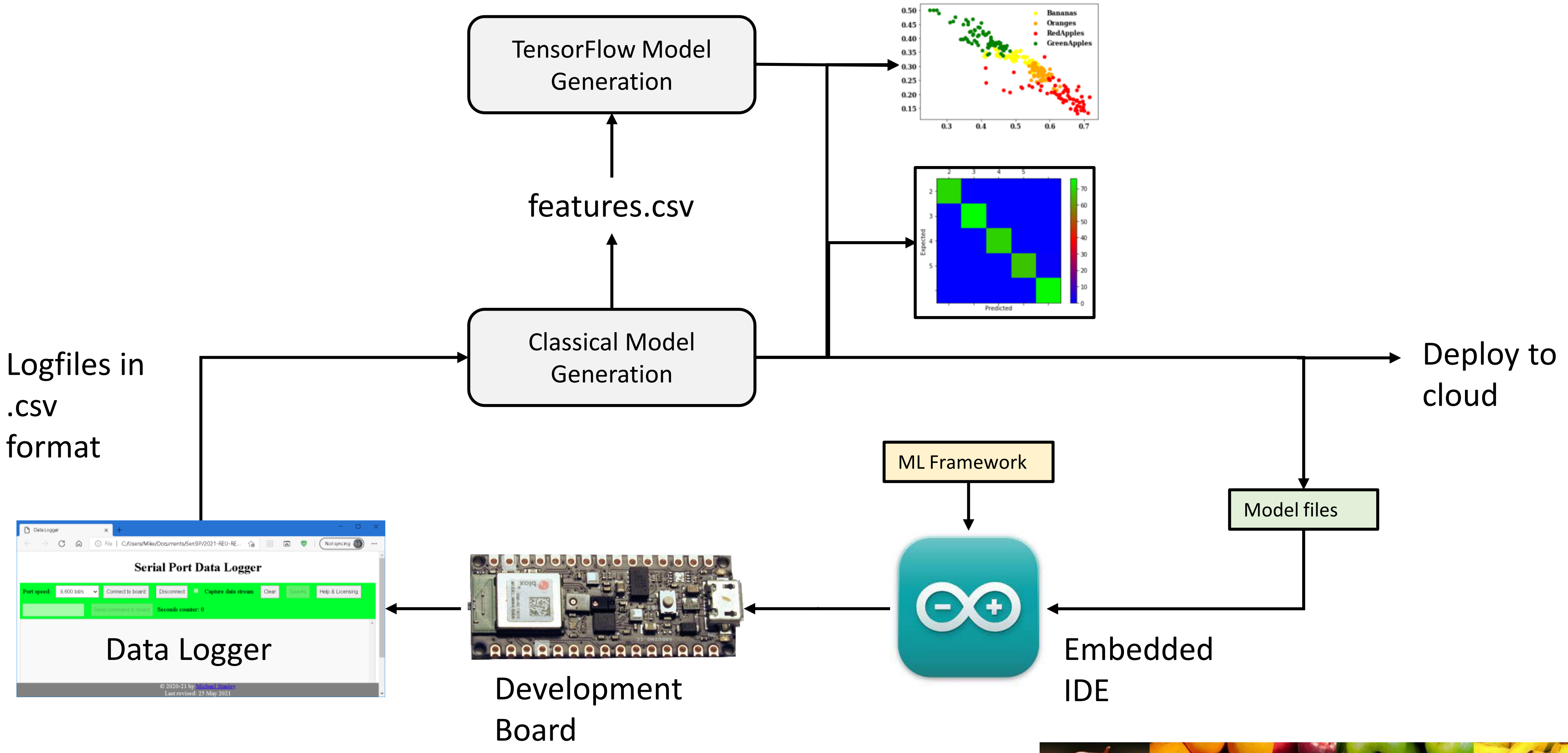
AdaBoost is an example of boosting. Here, we use a collection of crude “stub models” to create a final ensemble model with 98% accuracy on training data.



Example stub model



ML Modeling Example 1 Workflow

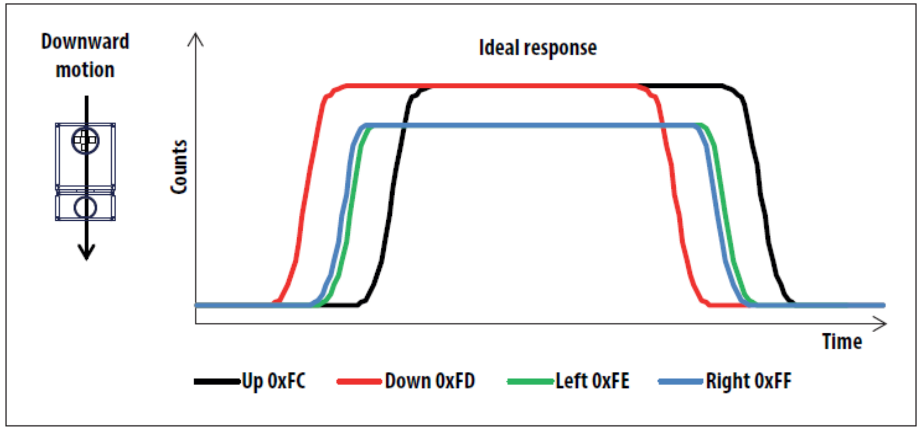


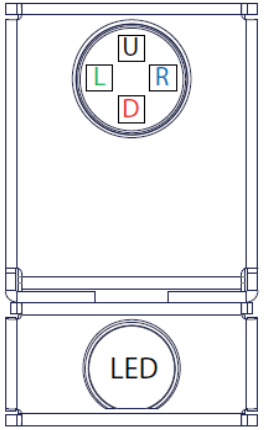
Light Sensor APDS-9960

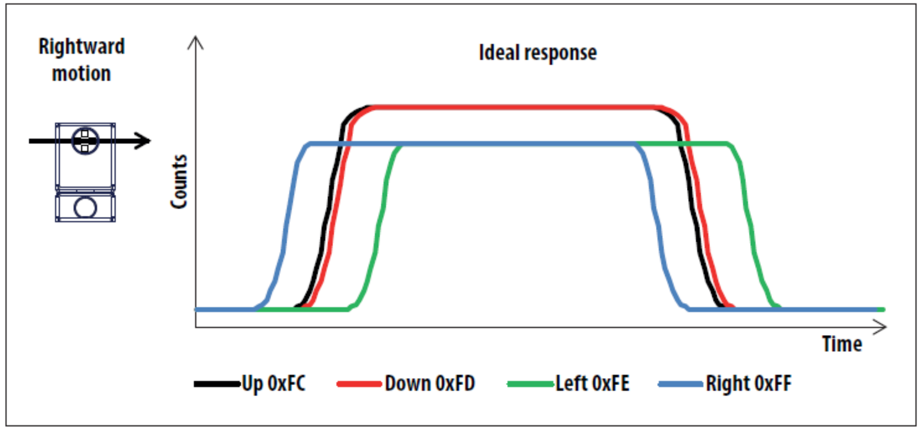
[Source: mouser.com](https://www.mouser.com)

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

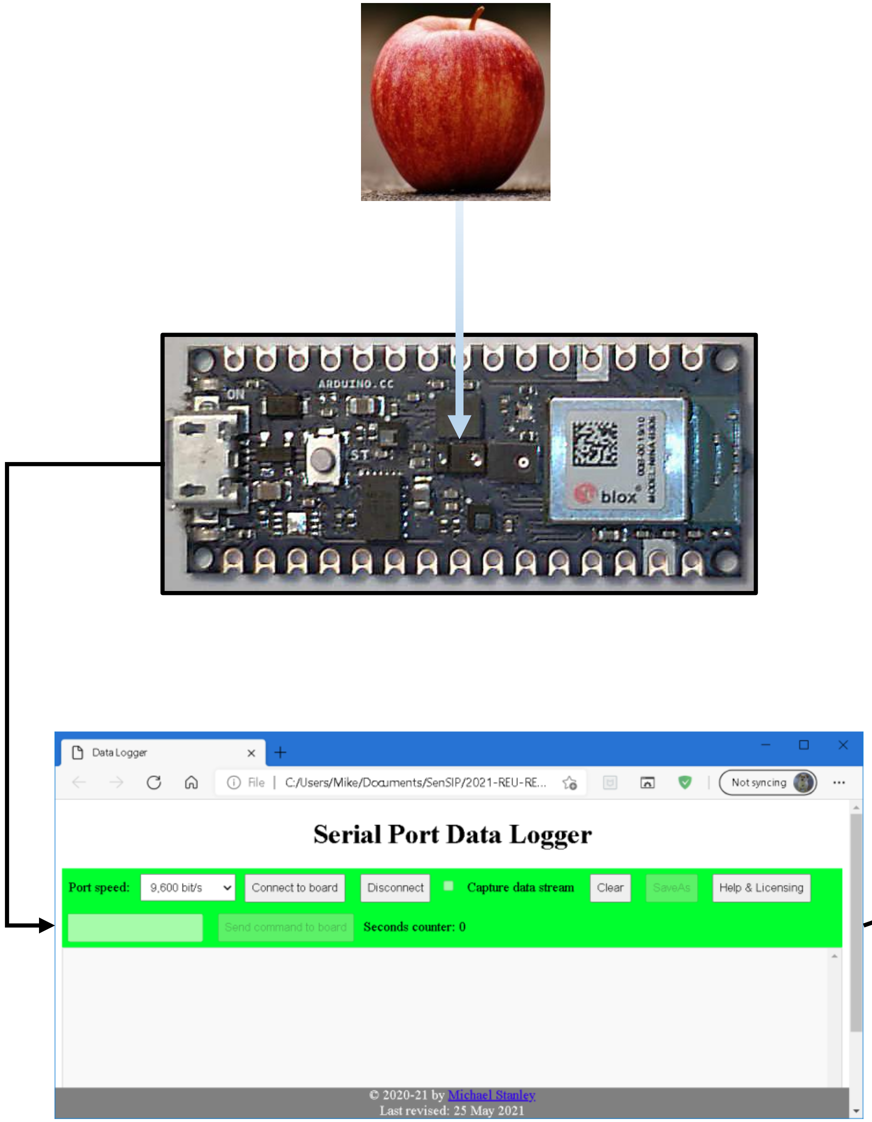
Embedded Data Logger Code

```
#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(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, 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);
}
```

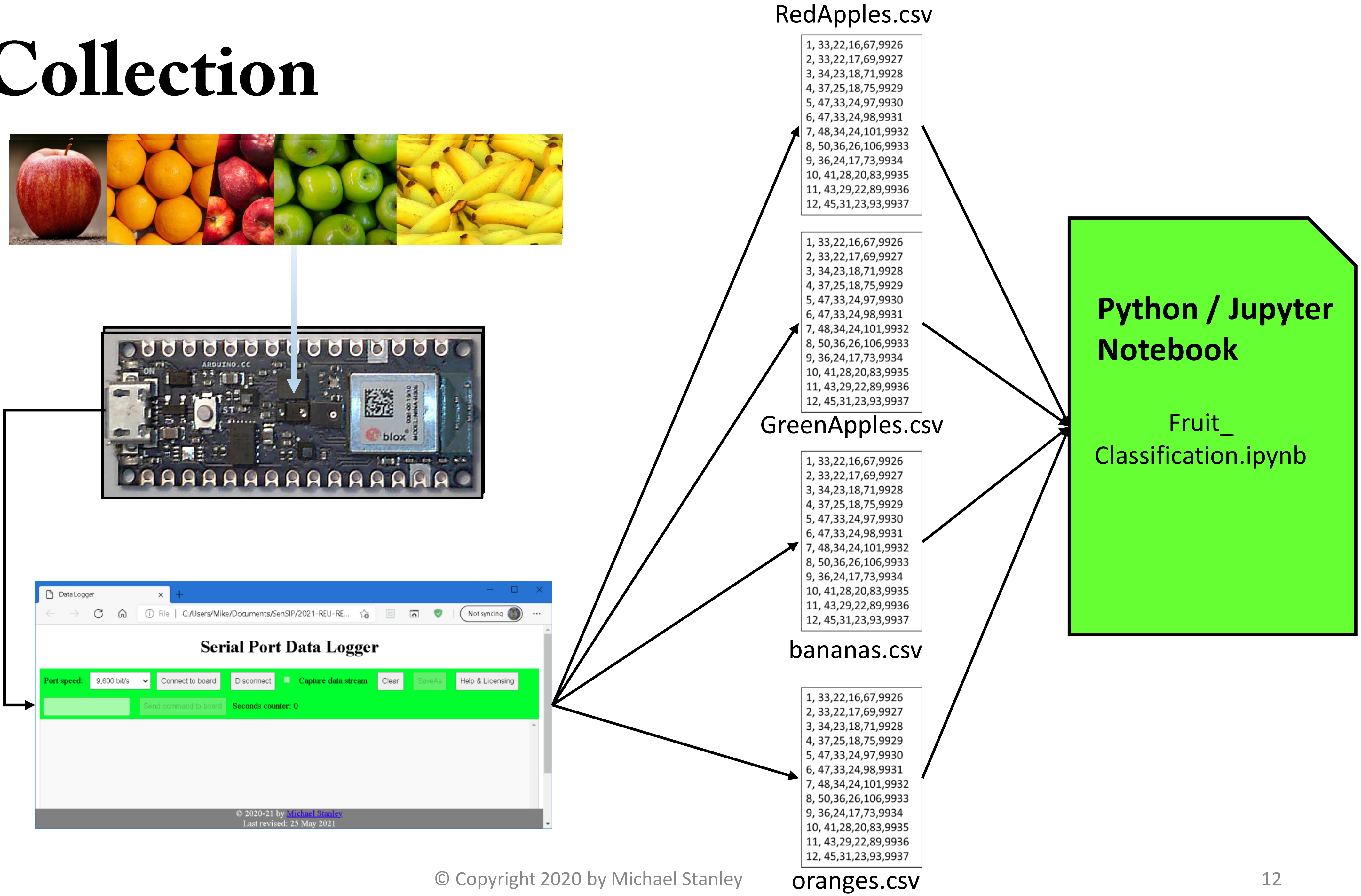
Data Collection



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

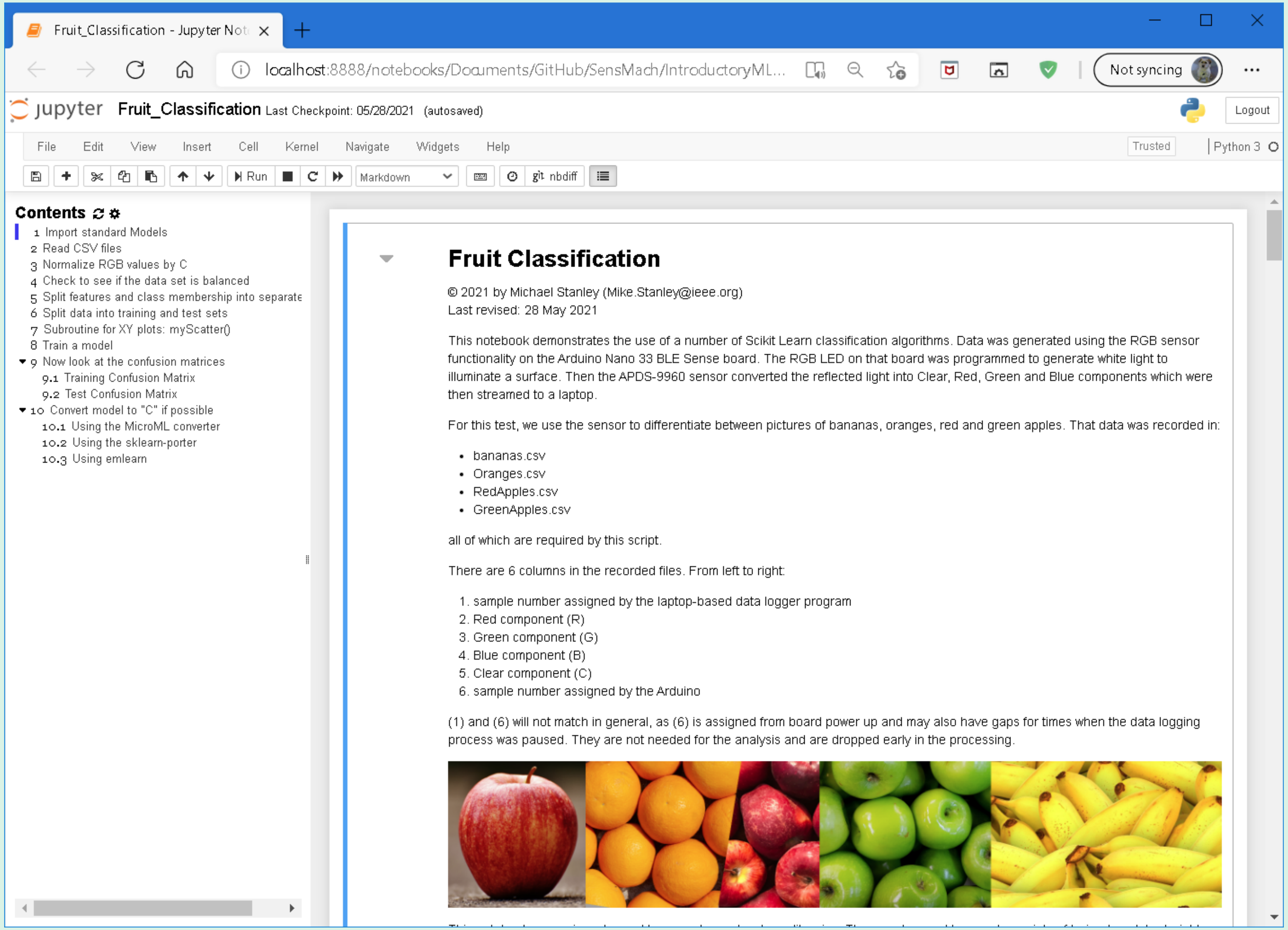
1, 33,22,16,67,9926
2, 33,22,17,69,9927
3, 34,23,18,71,9928
4, 37,25,18,75,9929
5, 47,33,24,97,9930
6, 47,33,24,98,9931
7, 48,34,24,101,9932
8, 50,36,26,106,9933
9, 36,24,17,73,9934
10, 41,28,20,83,9935
11, 43,29,22,89,9936
12, 45,31,23,93,9937

Data Collection



Walkthroughs

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



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(LEDR, LOW);
  digitalWrite(LEDG, LOW);
  digitalWrite(LEDB, 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

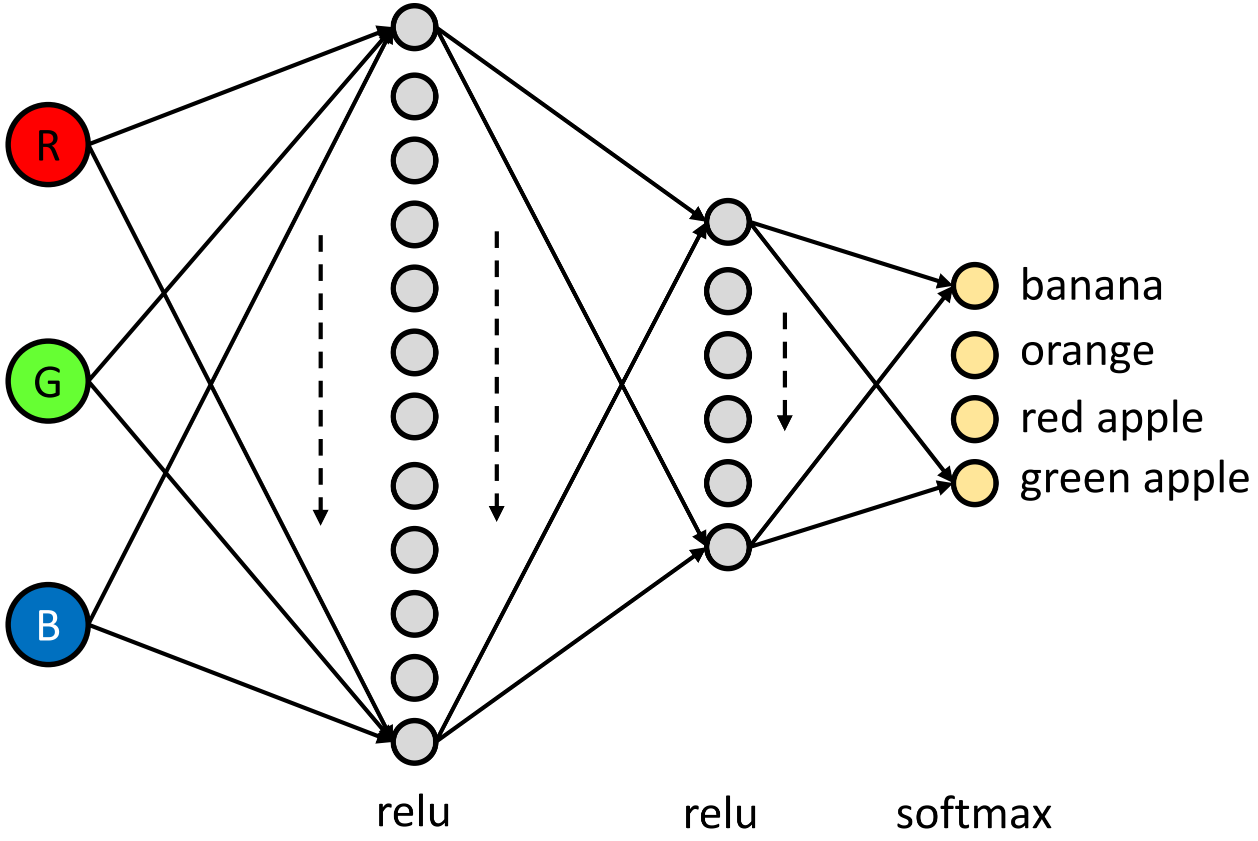
- Program Arduino with data logger
- Invoke HTML data logger
- Collect data samples (1 class per file)

- Import standard modules
- Read data files
- Normlize data
- Check for balanced data set
- Split data set
- Plot data
- Train
- Evaluate via confusion matrices
- Translate model to “C”

- Copy Arduino data logger source to new directory
- Cut & paste model from Jupyter to new C program
- Other code adjustments
- Build & download to board
- Reset board & test

Steps shown in black were completed in Jupyter

Neural Network to be generated via Keras

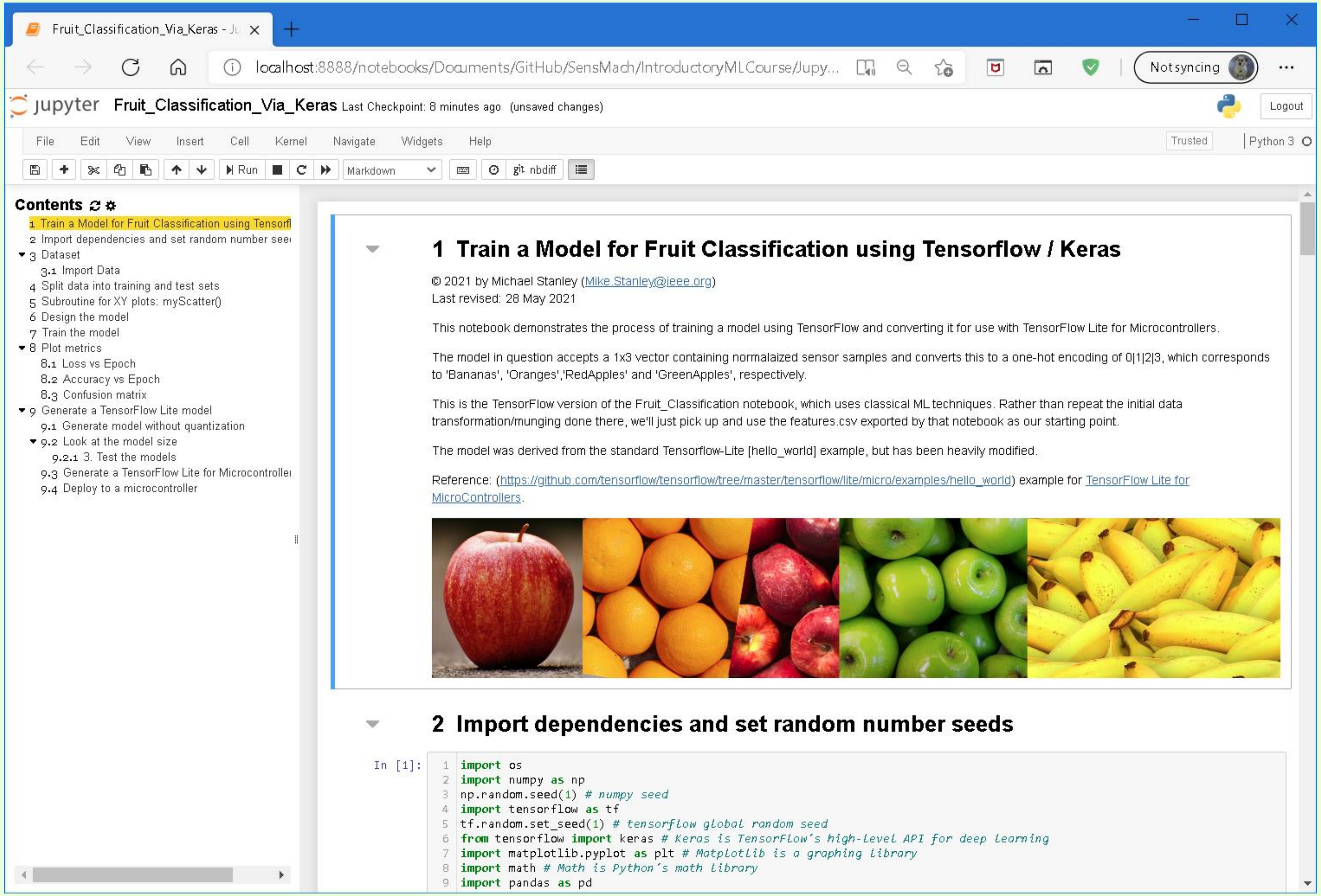


- Why this configuration? Because it worked. We tried several.
- All layers are fully connected
- The categorical final layer needed to be represented by 4 softmax outputs (differs from scikit-learn)

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

Walkthroughs

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



Files used in this reference

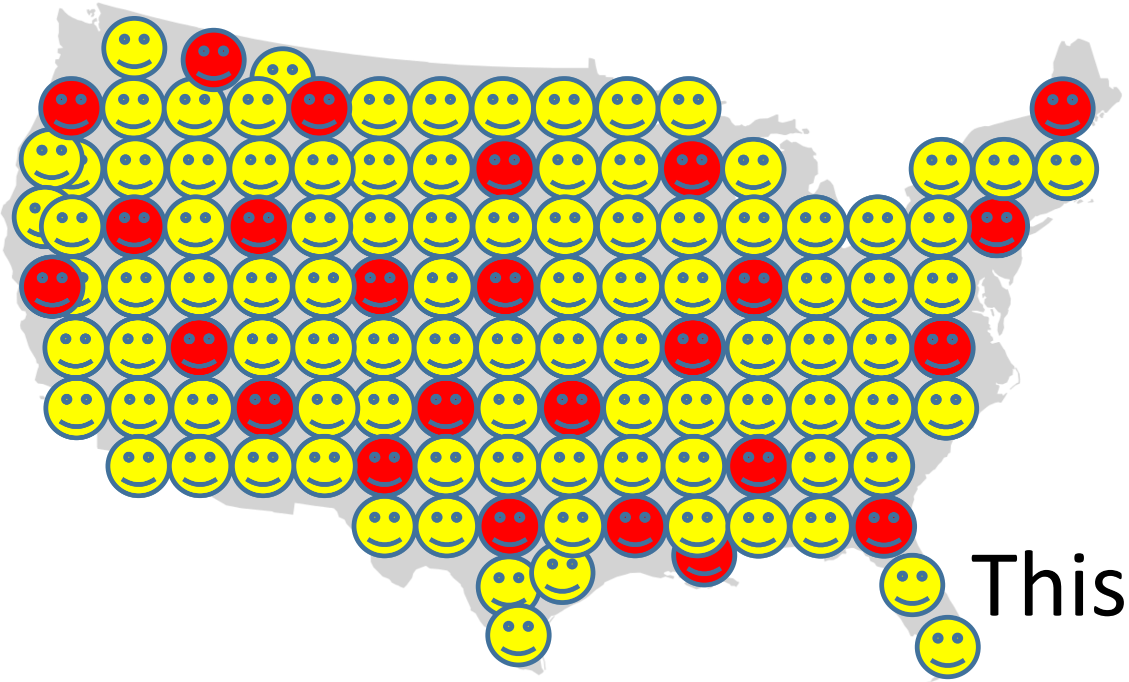
Filename(s)	Description
loggerV4.html / helpV3.html	Simple HTML/Javascript based application to record data sent by Arduino
ColorSensor.ino	Embedded code for Arduino data logger
ColorSensorInferencing.ino	Embedded code using decision tree model
ColorSensorKerasInferencing/*.*	Embedded code using Tensorflow 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

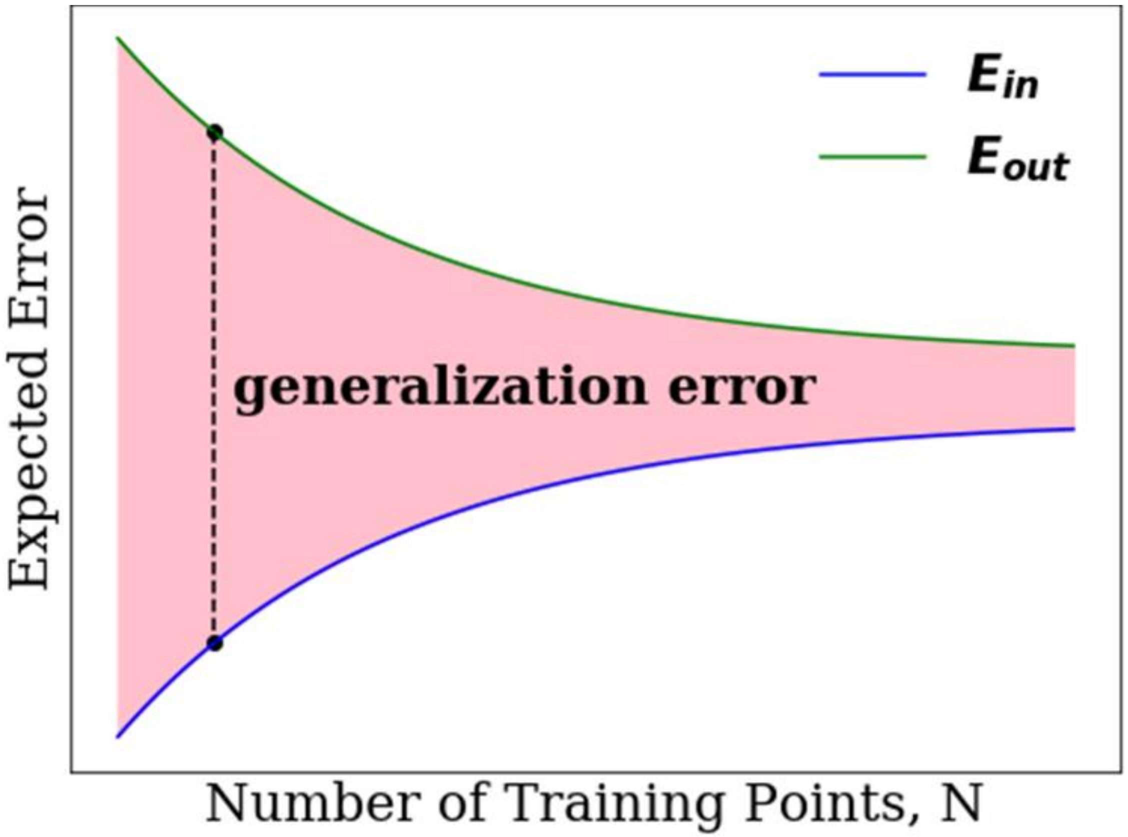
This you know

This you can only approximate

6/14/2021 20

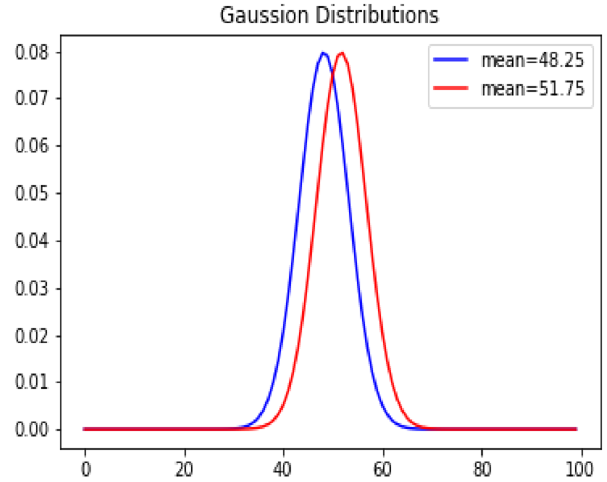
ML Goals

- 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.




FiveThirtyEight

Politics Sports Science & Health Economics Culture

OCT. 18, 2016, AT 11:54 AM

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

By Harry Enten

Filed under 2016 Election

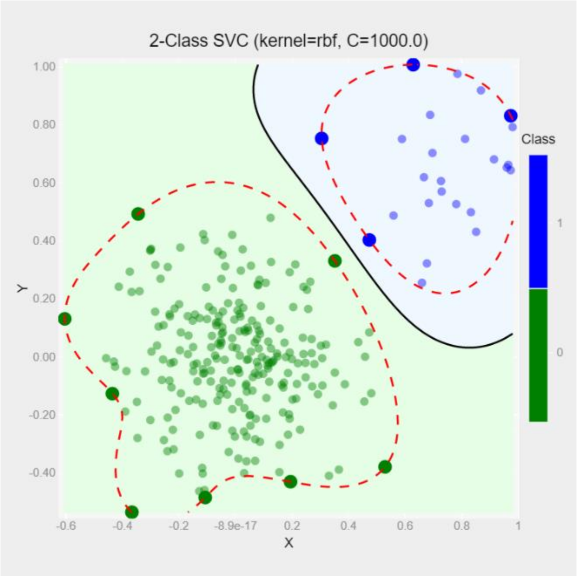
President Harry S. Truman gleefully displays an early edition of the Chicago Daily Tribune from his train in St. Louis after his defeat of Thomas E. Dewey in the 1948 presidential election. GETTY IMAGES

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

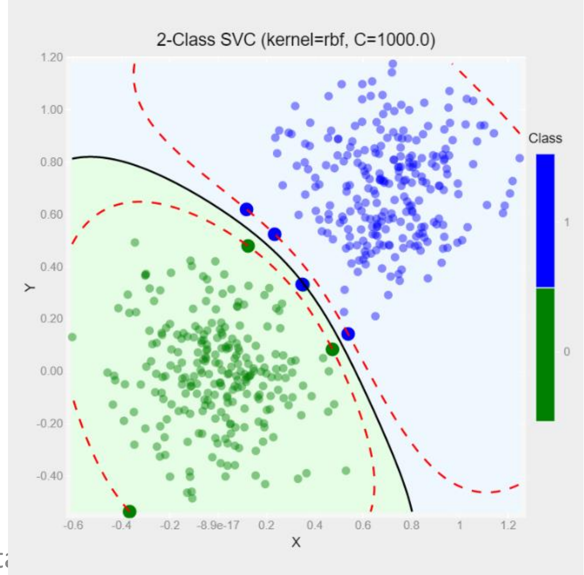
Effects of Unbalanced Training Sets

- 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

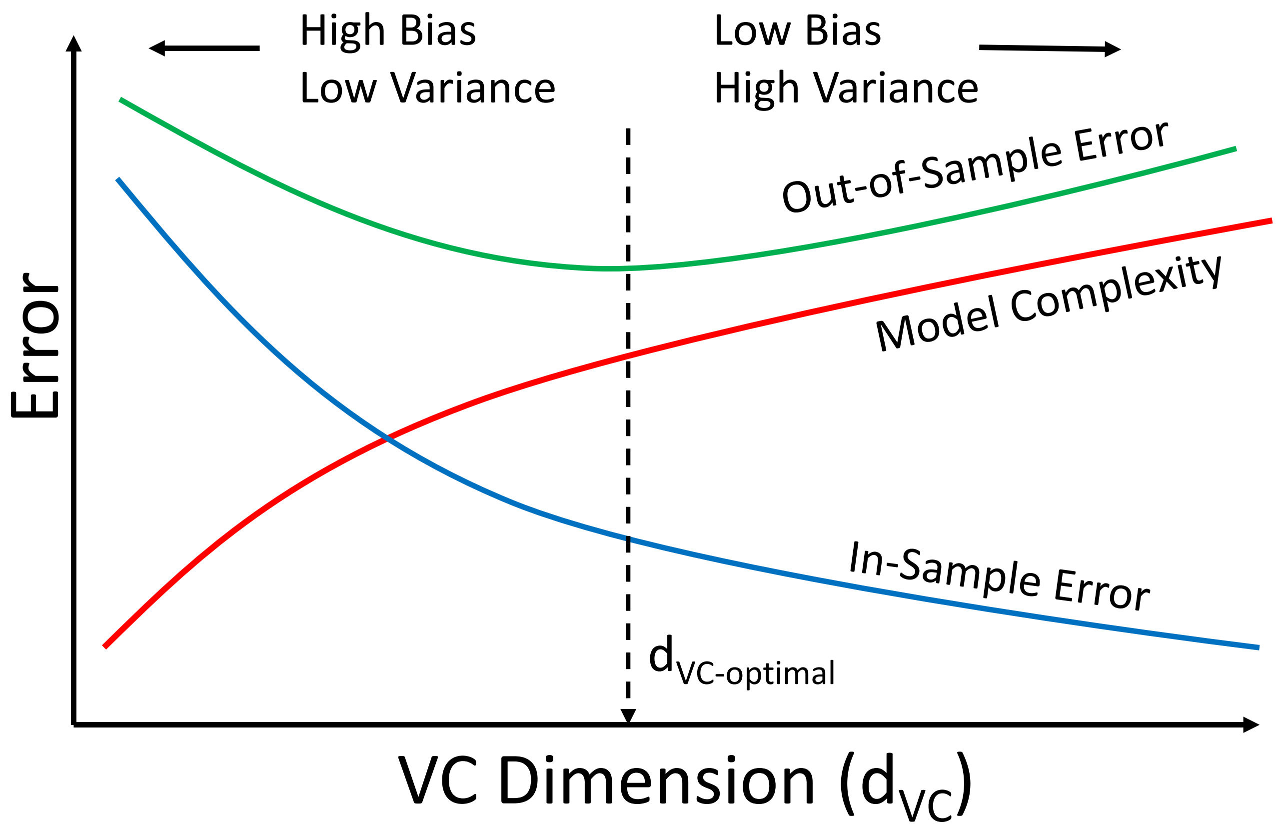
The Curse of Dimensionality

Rule of thumb:

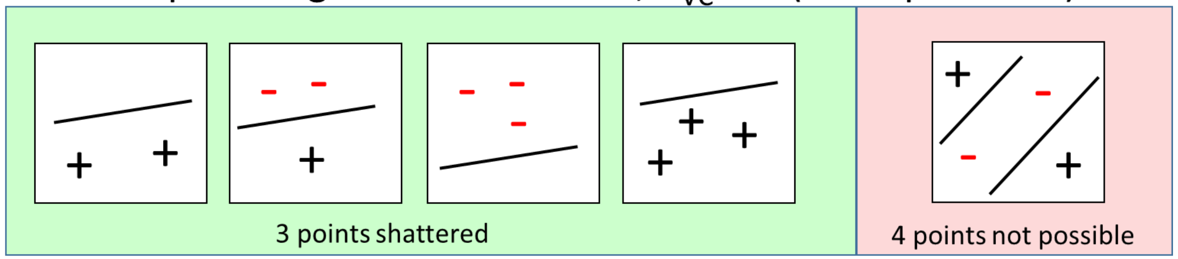
use $N >= 10\ d_{VC}$

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

https://work.caltech.edu/telecourse