Decision Trees

Thomas Schwarz, SJ

Decision Trees

- One of many machine learning methods
  - Used to learn categories
- Example:
  - The Iris Data Set
    - Four measurements of flowers
    - Learn how to predict species from measurement

Iris Data Set

Iris Setosa

Iris Virginica

Iris Versicolor

Iris Data Set

- Data in a .csv file
  - Collected by Fisher
  - One of the most famous datasets
    - Look it up on Kaggle or at UC Irvine Machine Learning Repository

Measuring Purity

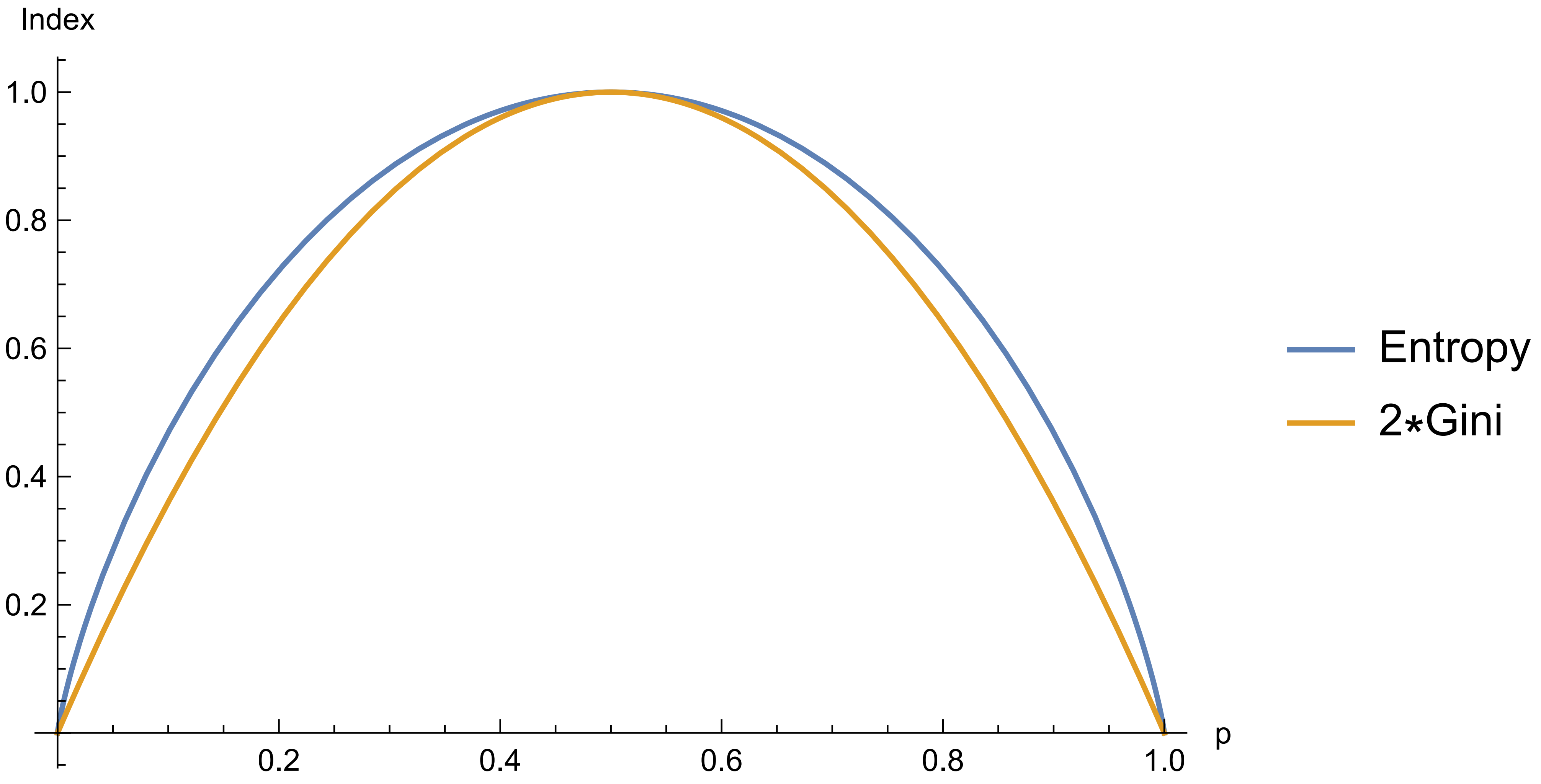
- Entropy
  - n categories with proportions p_i = (nr in Cat i)/(total nr)
  - $$\text{Entropy}(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} \log_2(p_i) p_i$$
    - Unless one of the proportions is zero, in which case the entropy is zero.
  - High entropy means low purity, low entropy means high purity

Measuring Purity

- Gini index
  - Gini$(p_1, p_2, \ldots, p_n) = \sum_{k=1}^{n} p_i(1 - p_i)$
  - Best calculated as
    - $\sum_{k=1}^{n} p_i(1 - p_i) = \sum_{k=1}^{n} p_i - \sum_{k=1}^{n} p_i^2 = 1 - \sum_{k=1}^{n} p_i^2$

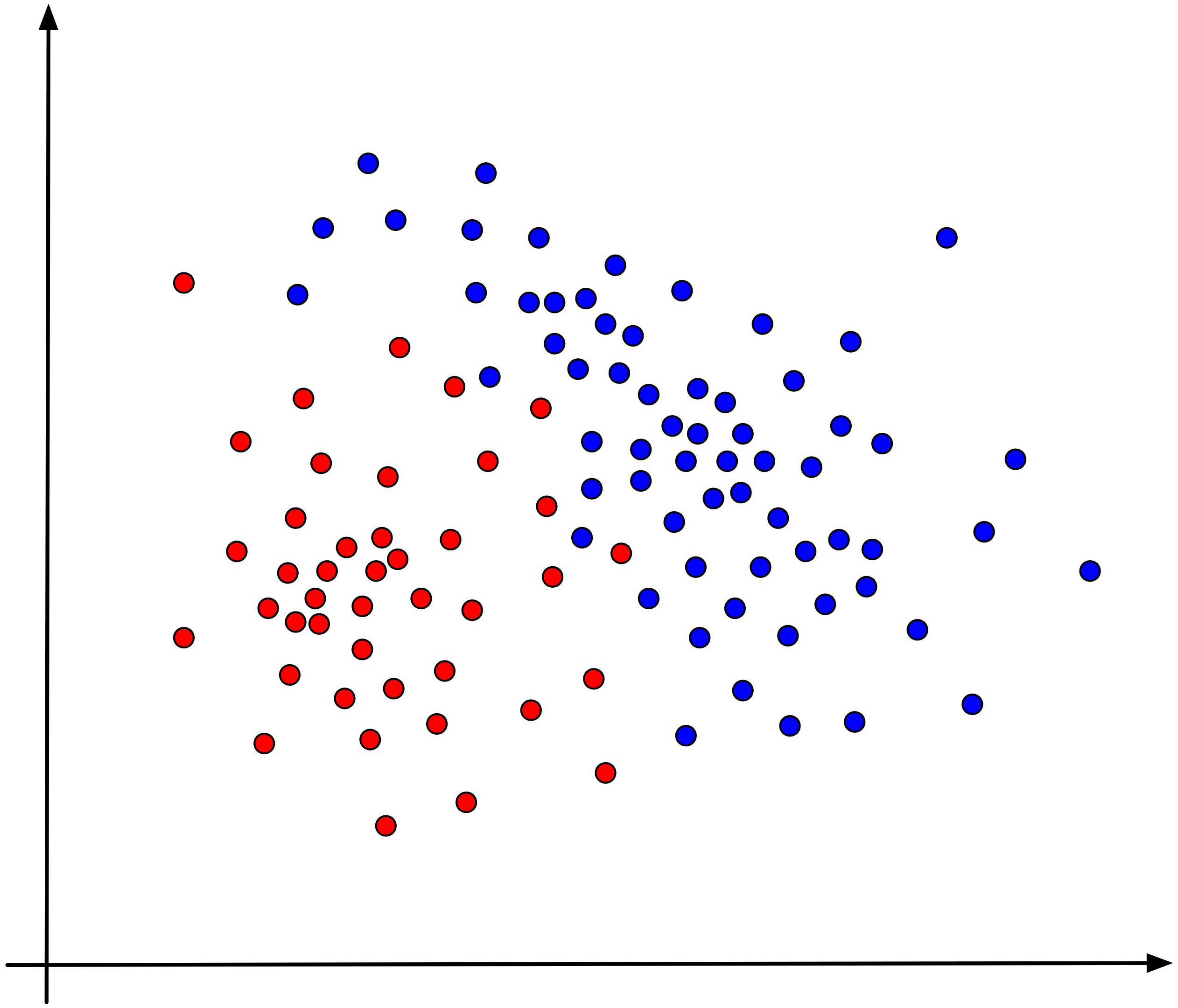
Measuring Purity

- Assume two categories with proportions p and q



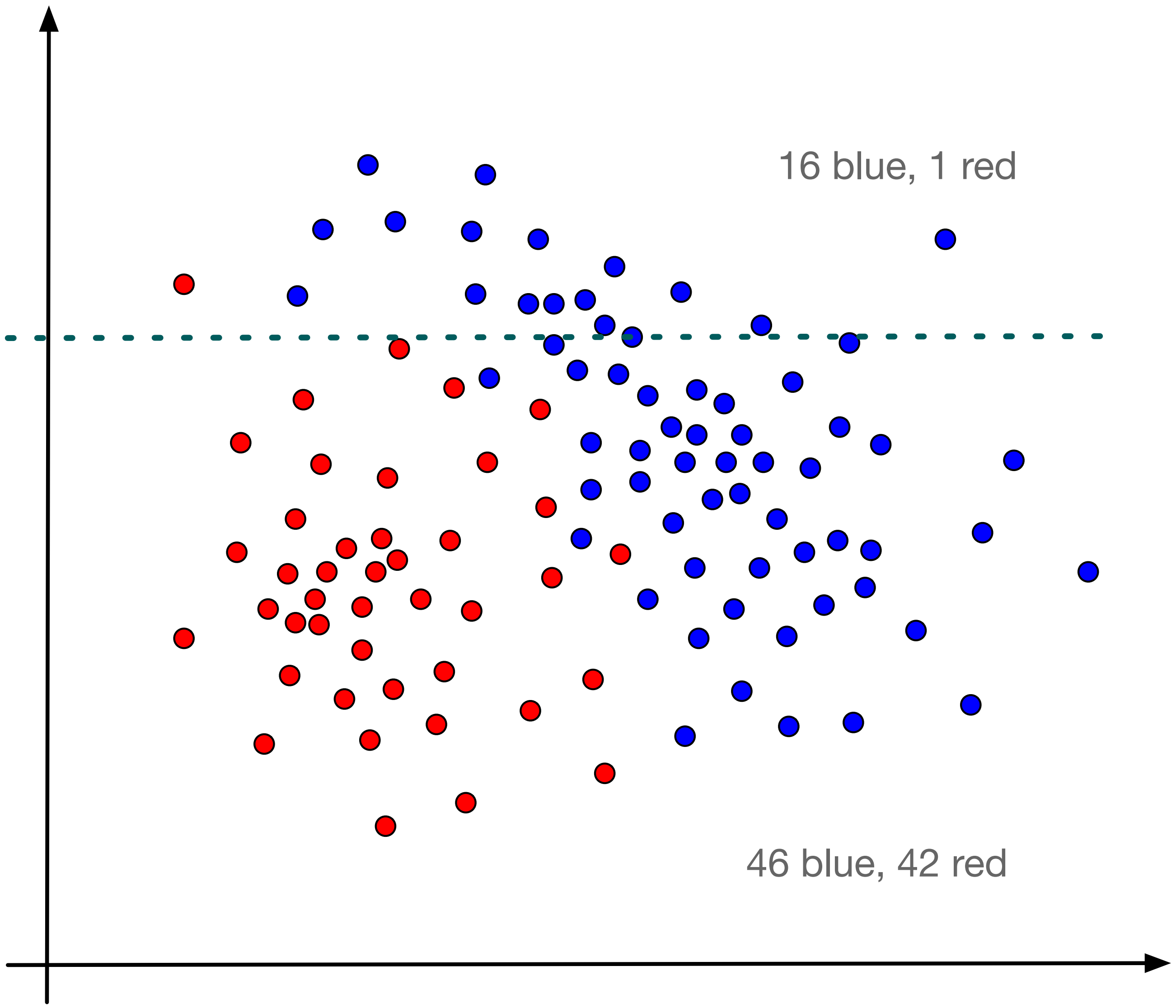
Building a Decision Tree

- A decision tree
  - Can we predict the category (red vs blue) of the data from its coordinates?



Building a Decision Tree

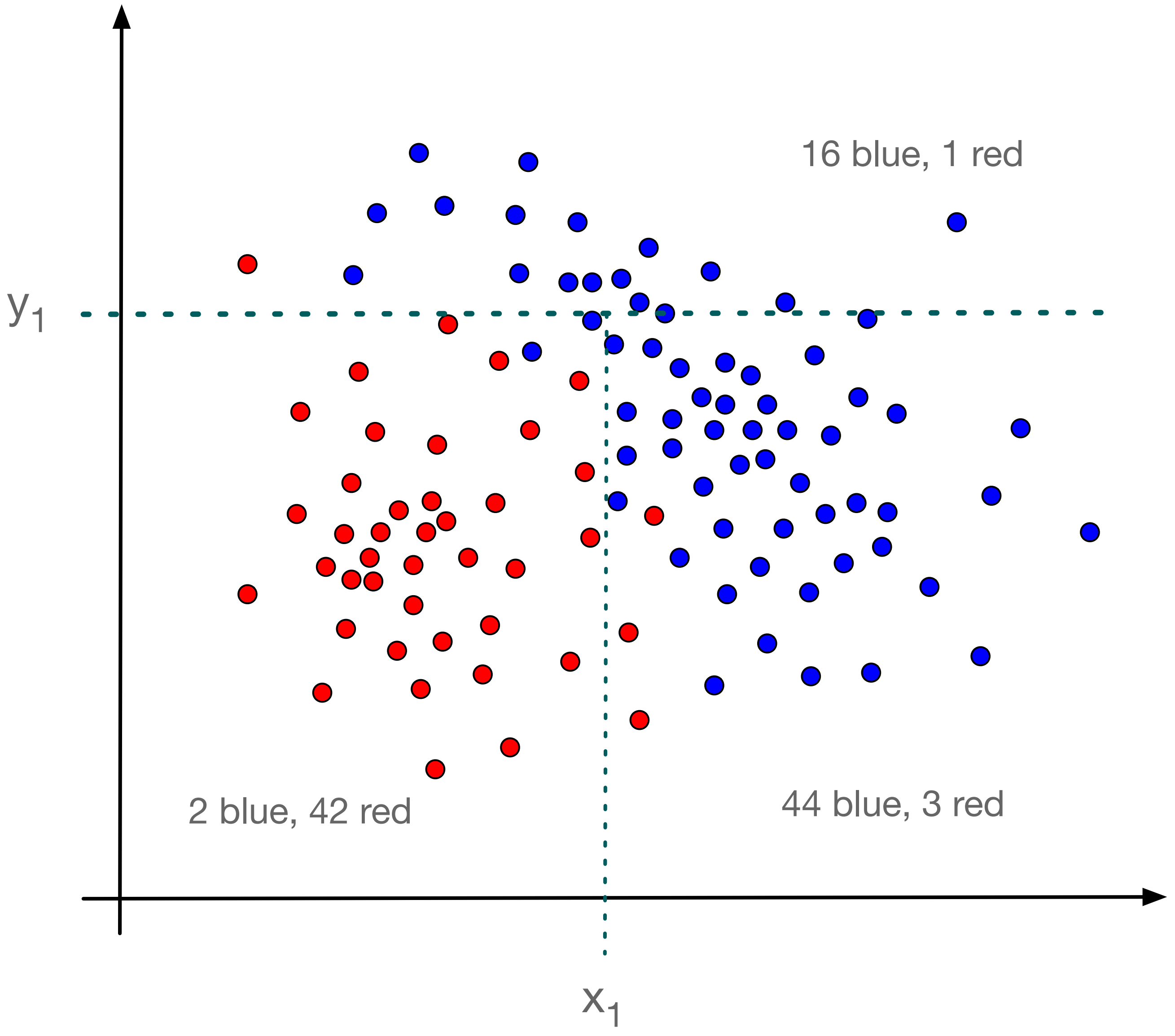
- Introduce a single boundary



`Almost all points above the line are blue`

Building a Decision Tree

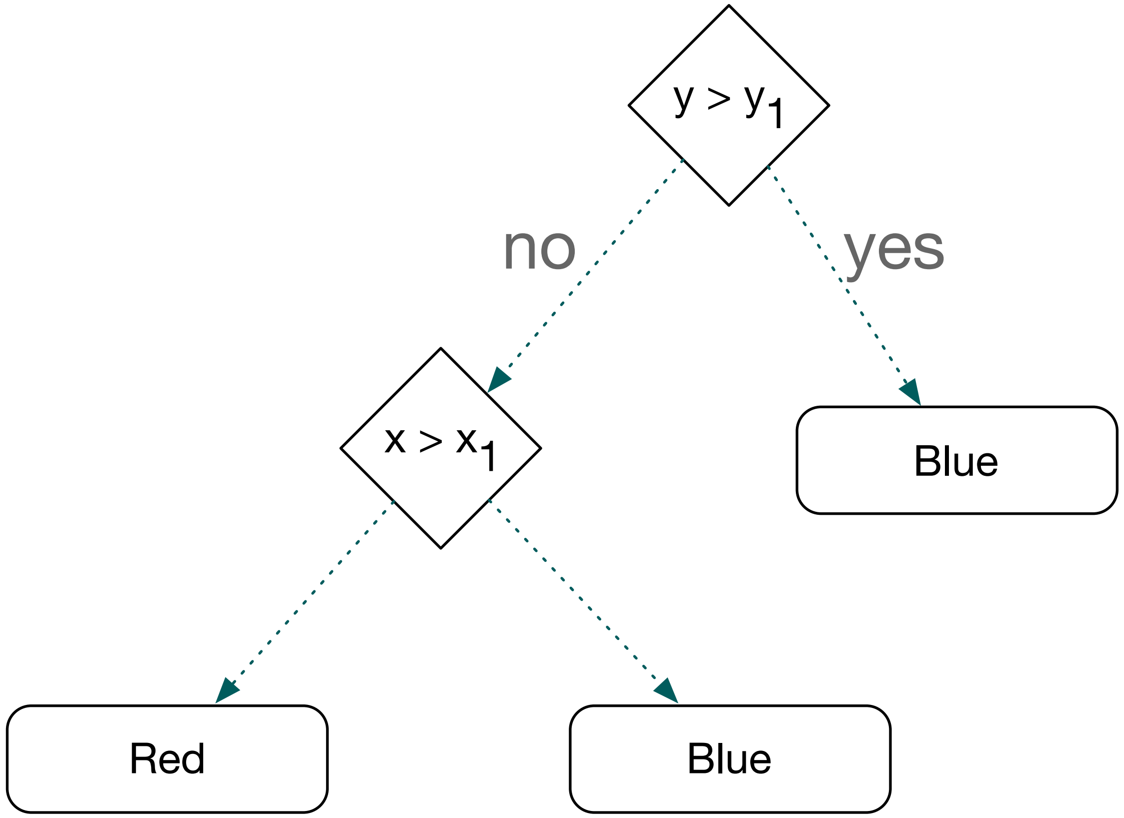
- Subdivide the area below the line



`Defines three almost homogeneous regions`

Building a Decision Tree

- Express as a decision tree



$y > y_1$

no

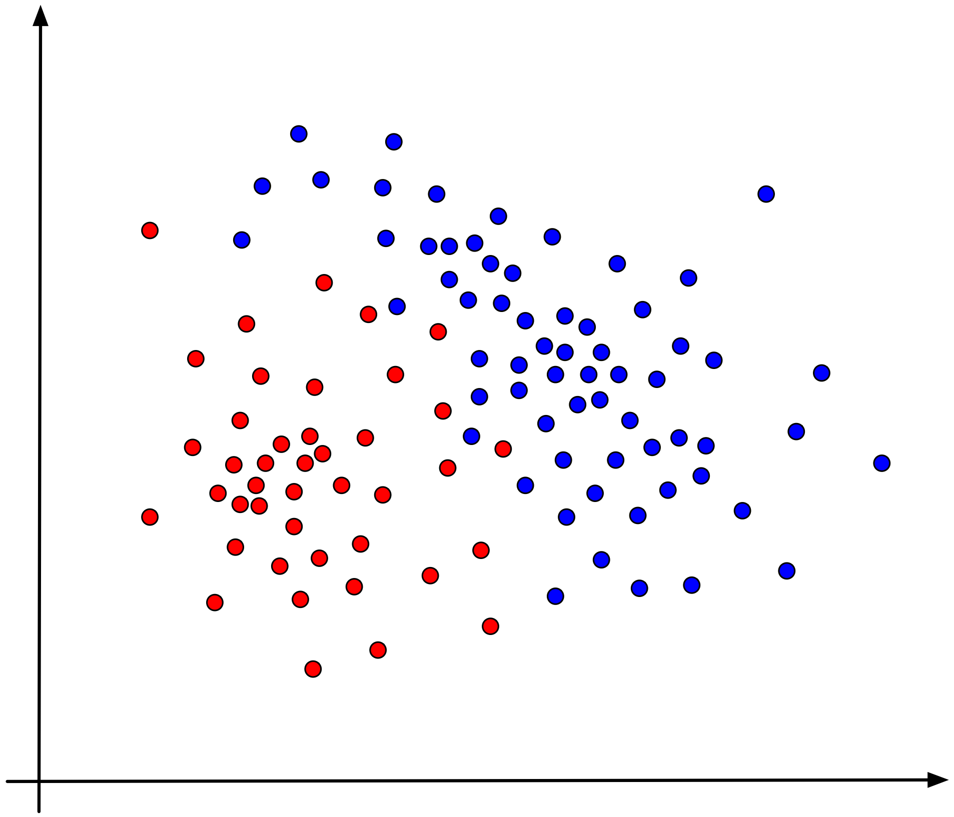
yes

$x > x_1$

Blue

Red

Blue



Building a Decision Tree

- Decision trees are easy to explain
- Might more closely mirror human decision making
- Can be displayed graphically and are easily interpreted by a non-expert
- Can easily extend to non-numeric variables

- Tend do not be as accurate as other simple methods
- Non-robust: Small changes in data sets give rise to completely different final trees

Building a Decision Tree

- If a new point with coordinates (x, y) is considered
  - Use the decision tree to predict the color of the point
- Decision tree is not always correct even on the points used to develop it
  - But it is mostly right
- If new points behave like the old ones
  - Expect the rules to be mostly correct

Building a Decision Tree

- How do we build decision trees
  - Many algorithms were tried out and compared
  - First rule: Decisions should be simple, involving only one coordinate
  - Second rule: If decision rules are complex they are likely to not generalize
    - E.g.: the lone red point in the upper region is probably an outlier and not indicative of general behavior

Building a Decision Tree

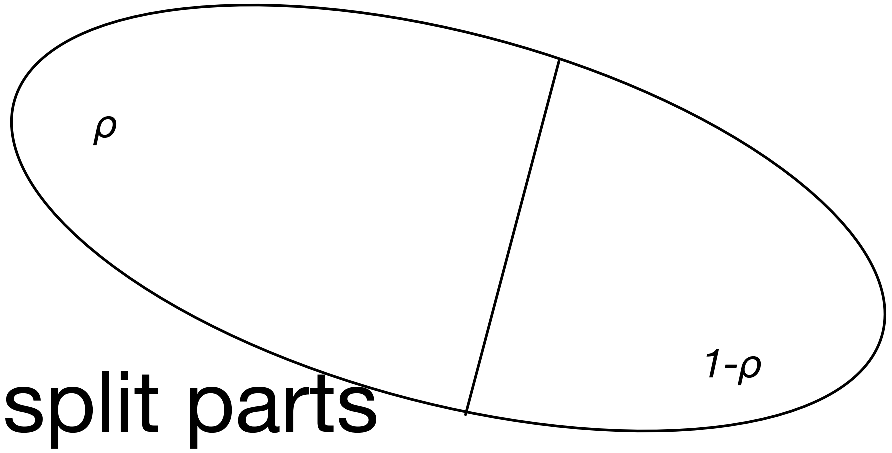
- How do we build decision trees
  - Third rule:
    - Don't get carried away
      - Prune trees to avoid overfitting

Building a Decision Tree

- Algorithm for decision trees:
  - Find a simple rule:
    - Maximizes the *information gain*
  - Continue sub-diving the regions
    - Stop when a region is homogeneous or almost homogeneous
    - Stop when a region becomes too small

Building a Decision Tree

- Information Gain from a split:



μ information measure before

μ_1, μ_2 information measures in the split parts

Information gain $= \mu - (\rho\mu_1 + (1-\rho)\mu_2)$

Processing Iris

- Need to get the data:
  - make tuples of float
  - last element:
    - use numbers 0, 1, 2 to encode categories

```
def get_data():
    """ opens up the Iris.csv file
    """
    lista = []
    with open("Iris.csv") as infile:
        infile.readline() # remove first line
        for line in infile:
            values = line.strip().split(',')
            if values[5] == "Iris-setosa":
                cat = 1
            elif values[5] == "Iris-versicolor":
                cat = 2
            else:
                cat = 0
            tupla = (float(values[1]),
                     float(values[2]),
                     float(values[3]),
                     float(values[4]),
                     cat)
            lista.append(tupla)
    return lista
```

Processing Iris

- Let's count categories

```
def stats(lista):
    counts = [0,0,0]
    for element in lista:
        counts[element[-1]] += 1
    return counts
```

Processing Iris

- Calculate the Gini index of a list

```
def gini(lista):
    counts = stats(lista)
    counts = [counts[0]/len(lista),
              counts[1]/len(lista),
              counts[2]/len(lista)]
    return 1-counts[0]**2-counts[1]**2-counts[2]**2
```

Processing Iris

- Calculate the entropy of a list

```
def entropy(lista):
    counts = stats(lista)
    proportions = [counts[0]/len(lista),
                   counts[1]/len(lista),
                   counts[2]/len(lista)]
    entropy = 0
    for prop in proportions:
        if prop!=0:
            entropy -= prop*math.log(prop,2)
    return entropy
```

Processing Iris

- Need to find all ways to split a list
  - First, let's have a helper function to remove doublettes

```
def unique(lista):
    result = []
    for value in lista:
        if value not in result:
            result.append(value)
    return result
```

Processing Iris

- Possible cutting points are the midpoints between values

```
def midpoints(lista, axis):
    """ calculates the midpoints along the coordinate axis
    """
    values = unique(sorted([pt[axis] for pt in lista]))
    return [ round((values[i-1]+values[i])/2,3) for i in
range(1, len(values))]
```

Processing Iris

- Splitting happens along a coordinate (axis) and a value:

```
def split(lista, axis, value):
    """ returns two lists, depending on pt[axis] < value or not
    """
    left, right = [], []
    for element in lista:
        if element[axis] < value:
            left.append(element)
        else:
            right.append(element)
    return left, right
```

Processing Iris

- Now we can find the axis and value that gives the maximum information gain
    - Set up frequently used values and the value to beat
    - best_split is going to contain axis and value
    - threshold does not look at splits that are too small

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
```

Processing Iris

- We need to try out all axes

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
    for axis in range(4):
        for value in midpoints(lista, axis):
            left, right = split(lista, axis, value)
            if len(left) > threshold and len(right) > threshol
                gain = gini_total - len(left)/nr*gini(left)-
len(right)/nr*gini(right)
                if gain > best_gain:
                    best_gain = gain
                    best_split = (axis, value)
    return best_split
```

Processing Iris

- We need to try out all axes, and then all midpoints

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
    for axis in range(4):
        for value in midpoints(lista, axis):
            left, right = split(lista, axis, value)
            if len(left) > threshold and len(right) > threshold:
                gain = gini_total - len(left)/nr*gini(left)-
len(right)/nr*gini(right)
                if gain > best_gain:
                    best_gain = gain
                    best_split = (axis, value)
    return best_split
```

Processing Iris

- If the left and right side have more than threshold members, calculate the gain

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
    for axis in range(4):
        for value in midpoints(lista, axis):
            left, right = split(lista, axis, value)
            if len(left) > threshold and len(right) > thresh
                gain = (gini_total - len(left)/nr*gini(left)
                       -len(right)/nr*gini(right))
                if gain > best_gain:
                    best_gain = gain
                    best_split = (axis, value)
    return best_split
```

Processing Iris

- If the information gain is the best, we store it

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
    for axis in range(4):
        for value in midpoints(lista, axis):
            left, right = split(lista, axis, value)
            if len(left) > threshold and len(right) > threshold:
                gain = gini_total - len(left)/nr*gini(left)-
len(right)/nr*gini(right)
                if gain > best_gain:
                    best_gain = gain
                    best_split = (axis, value)
    return best_split
```

Processing Iris

- At the end, we return the best split point

```
def best_split(lista, threshold = 3):
    best_gain = 0
    best_split = None
    gini_total = gini(lista)
    nr = len(lista)
    for axis in range(4):
        for value in midpoints(lista, axis):
            left, right = split(lista, axis, value)
            if len(left) > threshold and len(right) > threshold:
                gain = gini_total - len(left)/nr*gini(left)-
len(right)/nr*gini(right)
                if gain > best_gain:
                    best_gain = gain
                    best_split = (axis, value)
    return best_split
```

Processing Iris

- We could save the result of the best split seen so far
  - but splits are fast, so we do not bother

Processing Iris

- We need to check how well our decision tree works
  - We split the data set into a training set and a test set
  - We use 80% - 20%, i.e. p=.80

```
def separate(lista, p):
    train, test = [], []
    for element in lista:
        if random.random() < p:
            train.append(element)
        else:
            test.append(element)
    return train, test
```

Processing Iris

- We build the decision tree by hand

```
>>> best_split(train)
(2, 2.45)
>>> l, r = split(train, 2, 2.45)
>>> stats(l)
[0, 43, 0]
>>> stats(r)
[40, 0, 43]
```

- First decision neatly separates Iris-versicolor from the rest

Processing Iris

- Now we look at the other set

```
>>> best_split(r)
(3, 1.75)
>>> rl, rr = split(r,3,1.75)
>>> stats(rl)
[5, 0, 43]
>>> stats(rr)
[35, 0, 1]
```

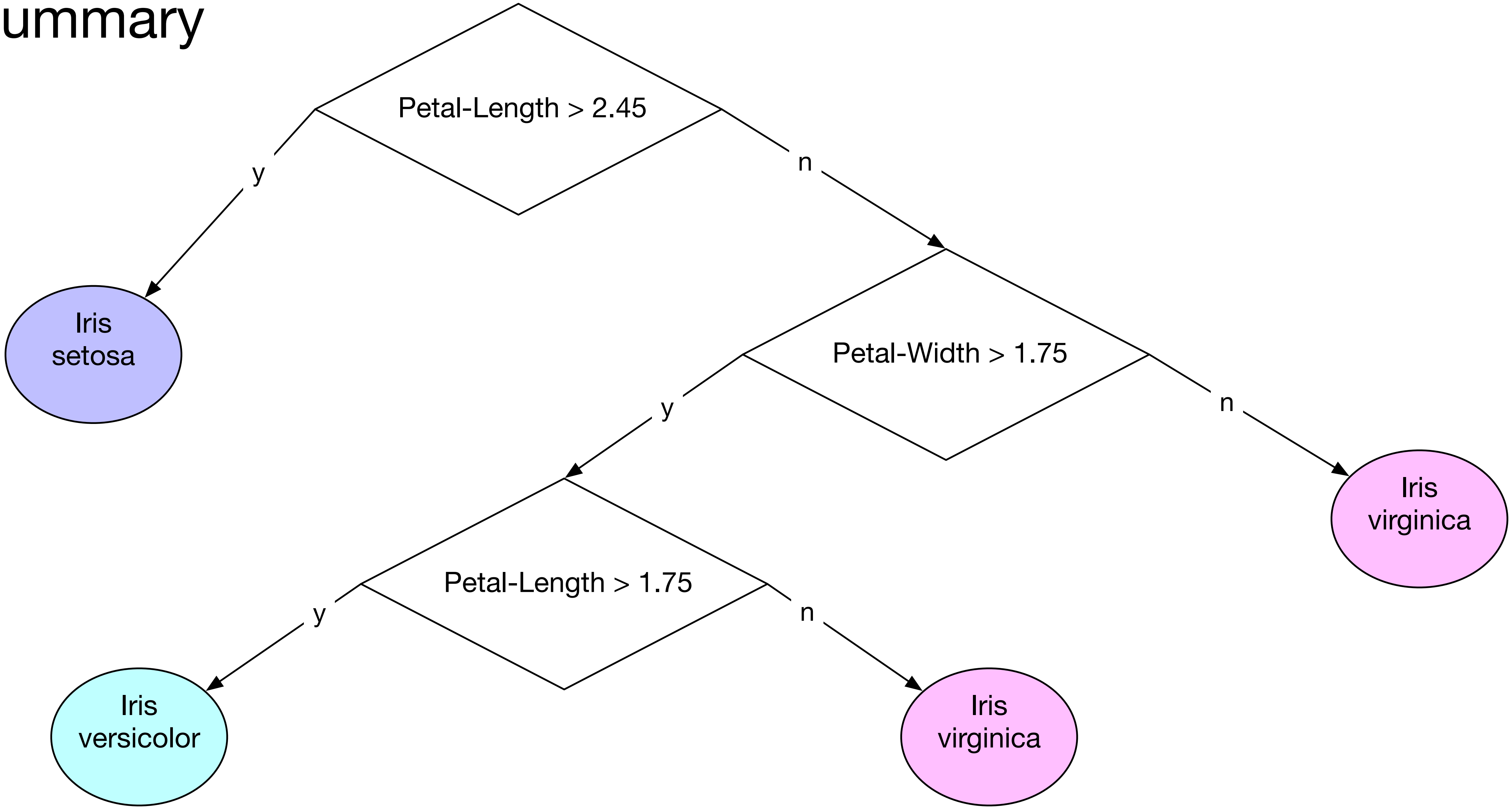
- This is almost an optimal split
    - rr should not be further subdivided
    - rl could work better

Processing Iris

```
>>> best_split(rl)
(2, 4.95)
>>> rll, rlr = split(rl, 2, 4.95)
>>> stats(rll)
[1, 0, 41]
>>> stats(rll)
[4, 0, 2]
```

Processing Iris

- Summary



Petal-Length > 2.45

y

n

Iris setosa

Petal-Width > 1.75

y

n

Iris virginica

Petal-Length > 1.75

y

n

Iris versicolor

Iris virginica

Testing

- Let's implement the decision tree:

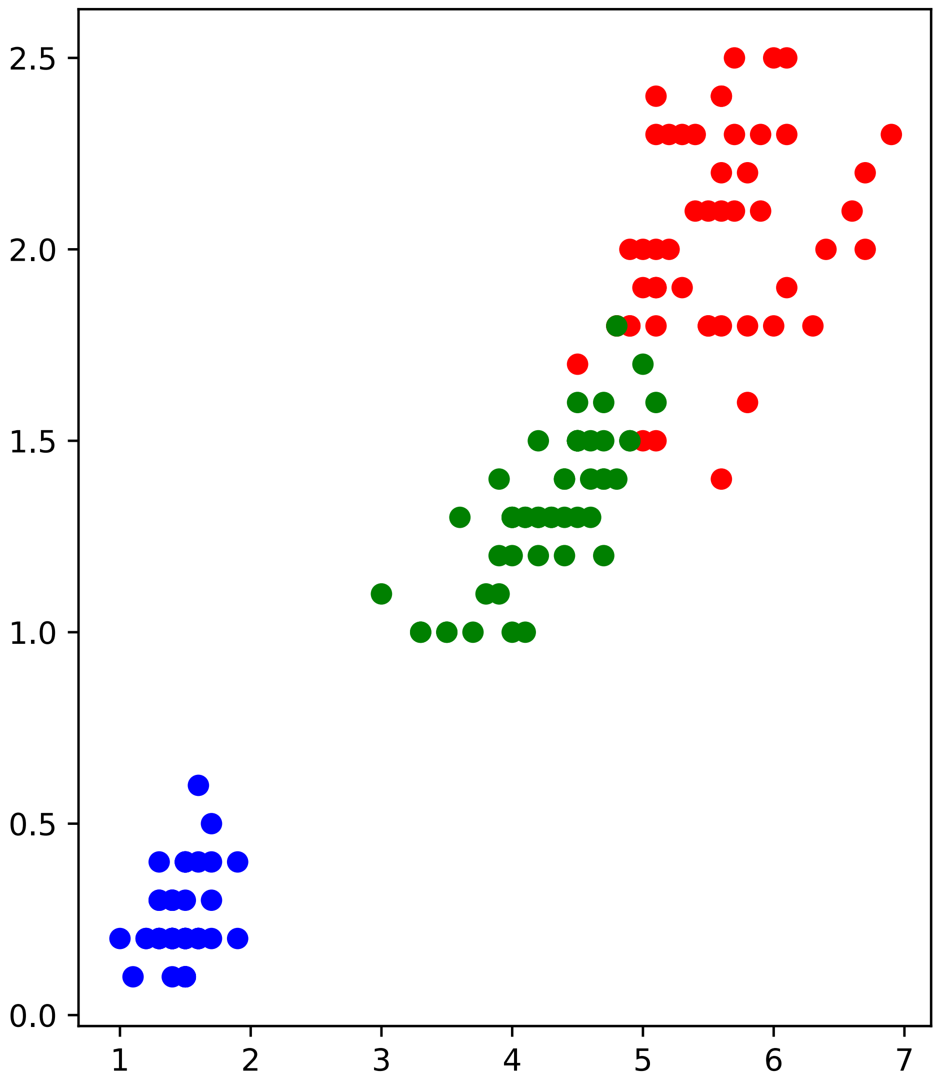
```
def predict(element):
    if element[2] < 2.45:
        return 1
    else:
        if element[3] < 1.75:
            if element[2] < 4.95:
                return 2
            else:
                return 0
        else:
            return 0
```

Testing

- And see how it works on the test data
  - One confused element or 1/36 error rate
- Total:
  - Four confused elements out of 150

Result

- Petal length and width are best at separating types



```
from matplotlib import pyplot as plt

plt.figure(figsize = (5,6))
plt.scatter( [el[2] for el in Iris if el[-1]==0],
             [el[3] for el in Iris if el[-1]==0],
             c='red' )
plt.scatter( [el[2] for el in Iris if el[-1]==1],
             [el[3] for el in Iris if el[-1]==1],
             c='blue' )
plt.scatter( [el[2] for el in Iris if el[-1]==2],
             [el[3] for el in Iris if el[-1]==2],
             c='green' )
plt.show()
```