Supervised Machine Learning

Part Two: Classification

DistrictDataLabs

Agenda

- Classification Models (Algorithms)
    - K Nearest Neighbors
    - Decision Trees
    - Support Vector Machines
    - Naive Bayes
- Model Evaluation

Classification Models

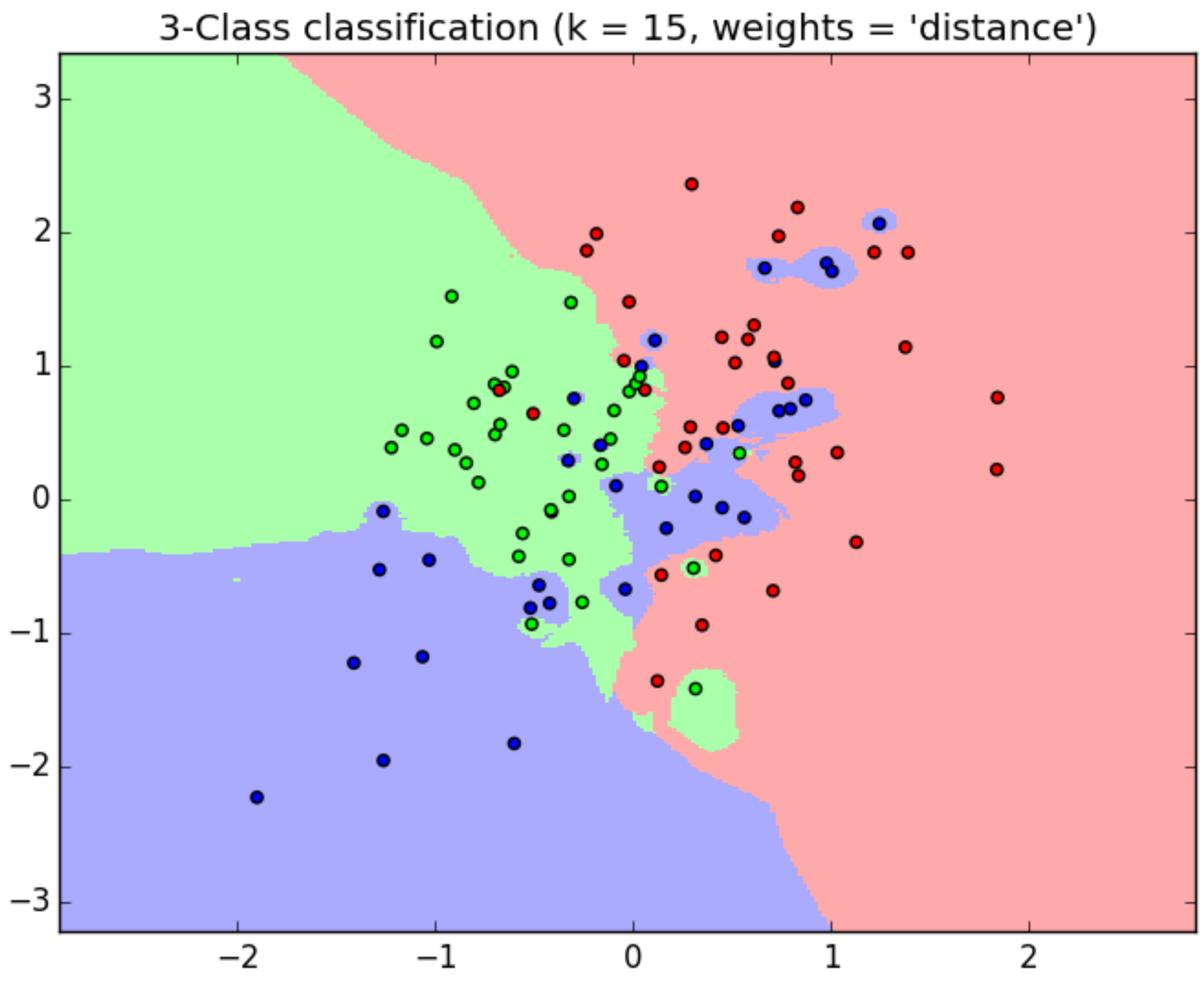
Classification Models

- Supervised learning algorithms that assign 2 or more discrete classes (categories) to an instance.
- Based on what is learned from the training data.
- An example of pattern recognition.
- Common algorithms
  - K Nearest Neighbors
  - Decision Trees
  - Support Vector Machines (SVM)
  - Naive Bayes

K Nearest Neighbors

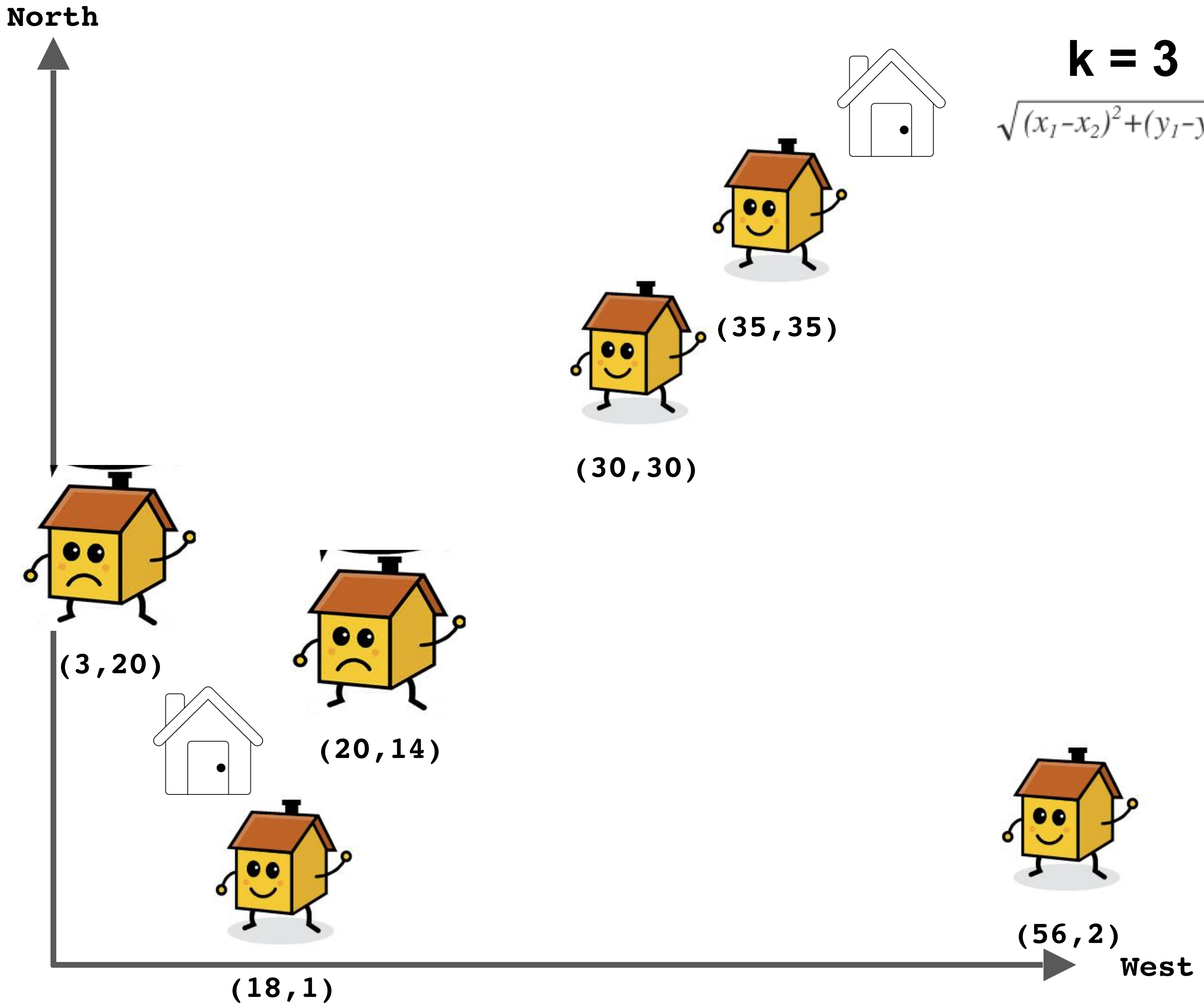
K Nearest Neighbors (kNN)

- In Scikit-Learn the `neighbors` package provides both supervised and unsupervised nearest neighbors methods
  - these are simple and effective!
- Find a predefined (k) samples closest in distance to the input point and predict the label from those.
- Distance == similarity in this case, and can be any metric measure. Euclidean distance is most common.
- Non-generalizing: must “remember” all training data.


3-Class classification (k = 15, weights = 'distance')
3
2
1
0
−1
−2
−3
−2
−1
0
1
2

k = 3

$$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$



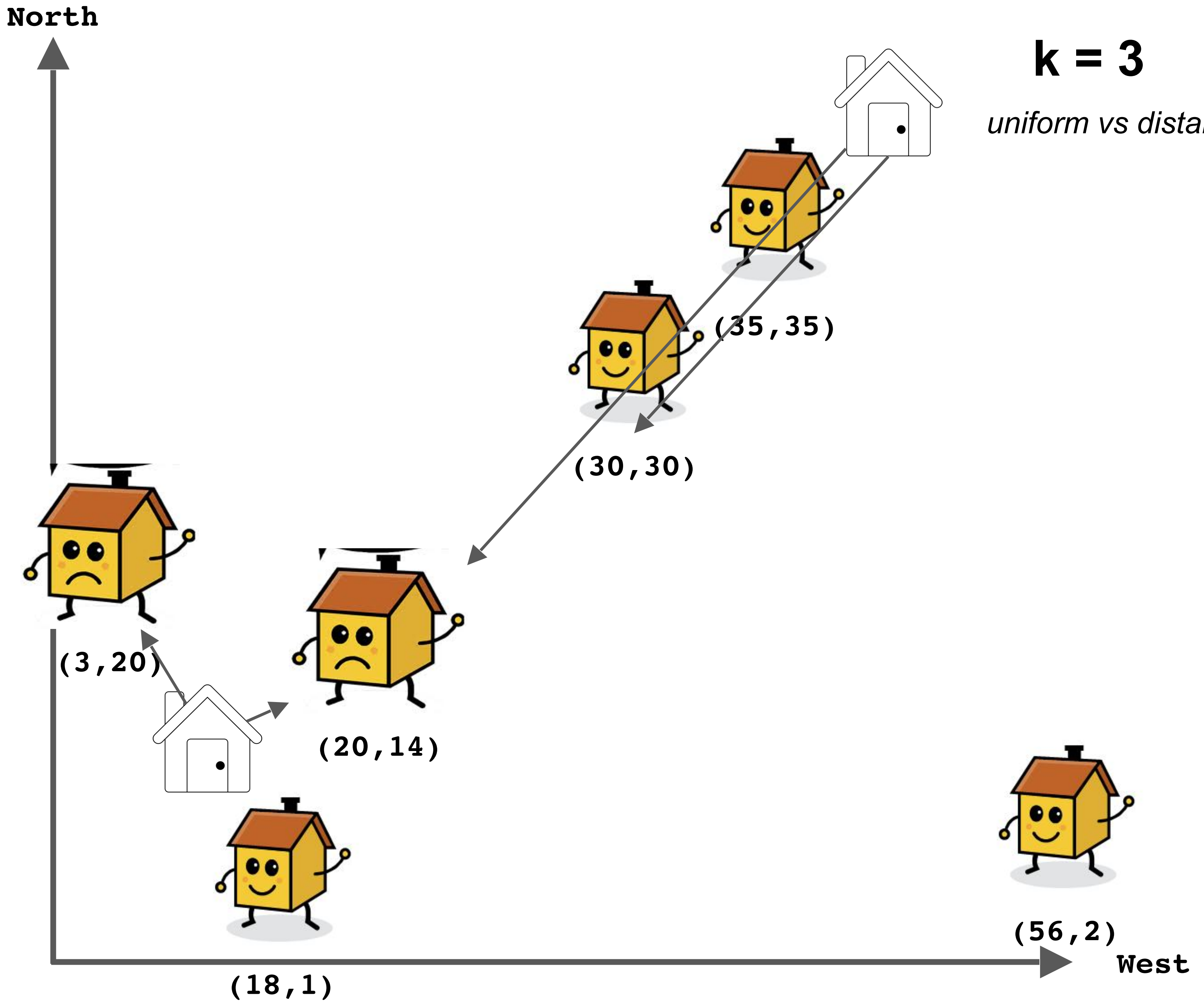
Compute Distance Metrics

```
from sklearn.metrics.pairwise import euclidean_distances

X = [[3,20],[20,14],[18,1],[30,30],[35,35],[56,2]]
y = [[10,10],[40,40]]

euclidean_distances(X,y)

array([[ 12.20655562,  42.05948169],
       [ 10.77032961,  32.80243893],
       [ 12.04159458,  44.77722635],
       [ 28.28427125,  14.14213562],
       [ 35.35533906,   7.07106781],
       [ 46.69047012,  41.23105626]])
```


North
k = 3
uniform vs distance
(35,35)
(30,30)
(3,20)
(20,14)
(56,2)
West
(18,1)

Pros and Cons of kNN

Pros

- Works well with distance-sensitive data
- Simple and effective for a wide array of tasks
- Non parametric - works well with irregular decision boundaries.
- Often a first approach
- No worse than 2x Bayes error rate as data → ∞

Cons

- kNN can be very arbitrary and requires hand-holding to model.
- How do you choose k?
- What is the definition of neighbor/similarity?
- Suffers from the curse of dimensionality.

Choosing K

K is an arbitrary number between 1 and the number of instances in the data set. Three options for choosing:

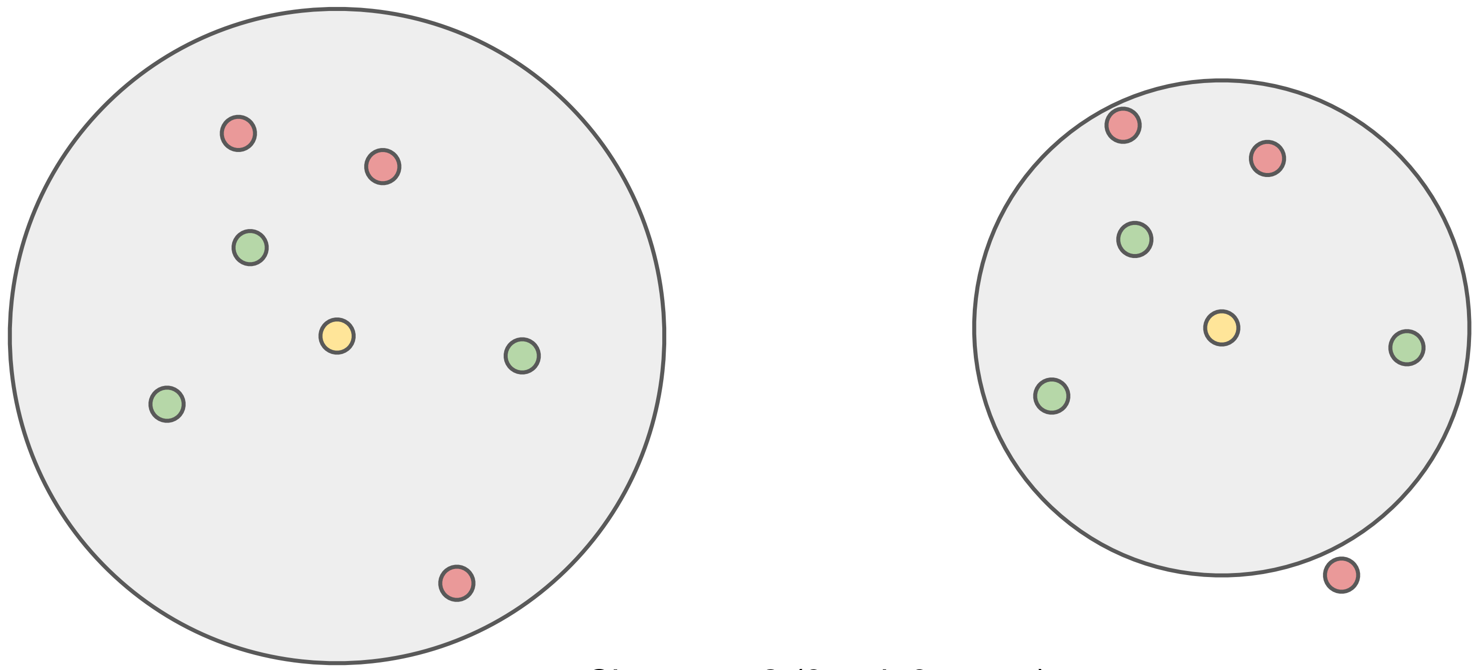
1. Guess
2. Heuristic
3. Optimization

Tips

- Avoid an even k with only 2 classes (tie break)
- Choose k >= # classes + 1
- Choose as low a k as possible

Use Coprime Class and k

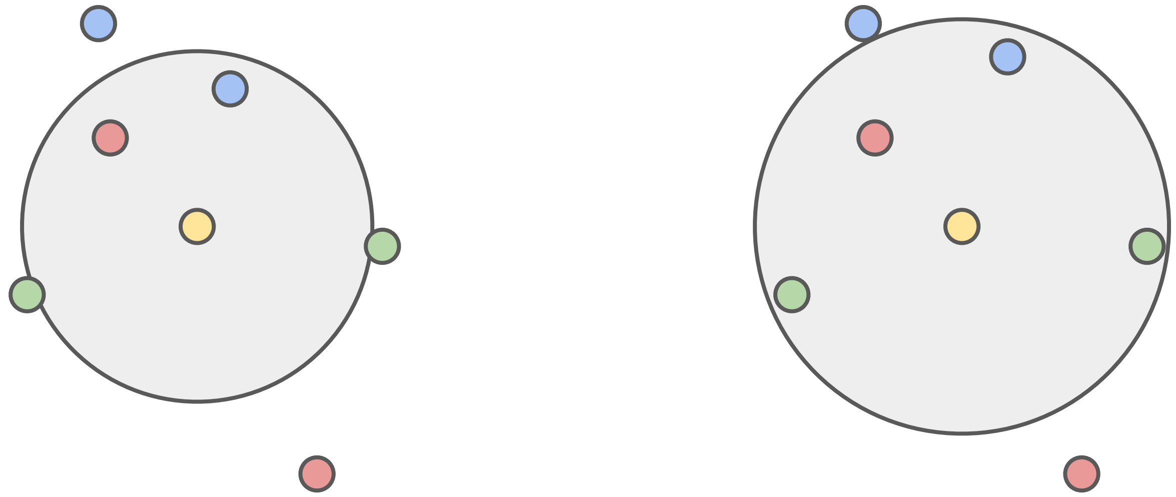
Coprime numbers don’t share common divisors except for 1. So 4 and 9 are coprime but not 3 and 9.



Classes = 2 (3 red, 3 green)
k = 6 and k=5 respectively

Use Number of Classes + 1

With k < # of classes, there is no chance that all classes will be represented - but we still want to prevent ties.



Classes = 3 (2 red, 2 green, 2 blue)
k = 2 and k=4 respectively

Optimization

- Many people assert k should be chosen via *domain knowledge* - as K increases, the *complexity* increases and slows down performance.
- Minimize error by trying different values of k = Hill Climbing problem. Algorithms: Genetic parameter optimization and simulated annealing.
- Iterate 2x through 1% of the data with a variety of K and compute error curve for minimization.

Lab: K Nearest Neighbors

Sklearn

Decision Trees

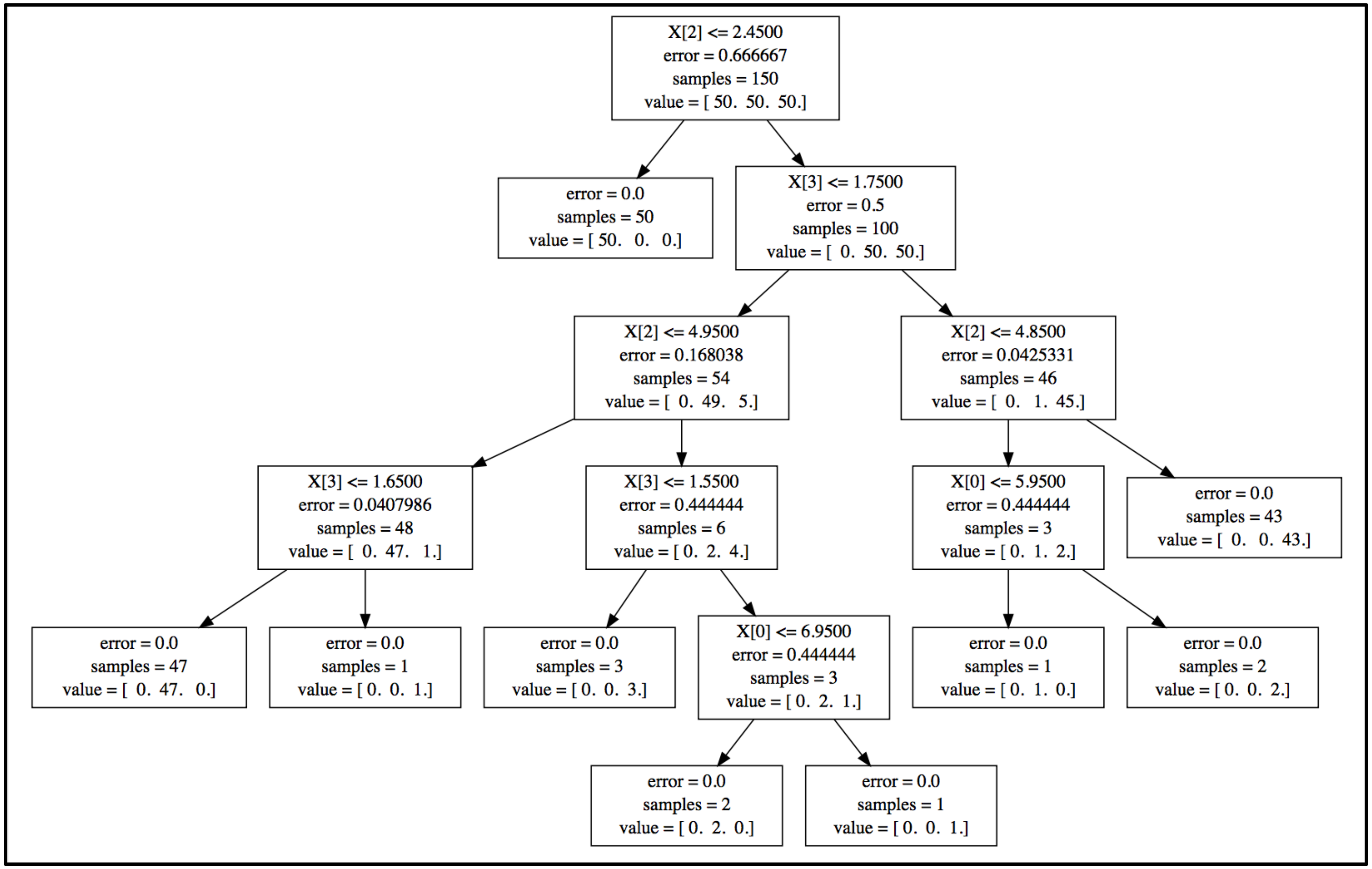
Decision Trees

- Another example of non-parametric inductive learning - but one that generalizes better.
- Creates a graphical model of rules that partitions the data until a decision is reached at one of the leaf nodes.
- Complexity is related to the amount of data and the partitioning method.

Decision Trees

- In Scikit-Learn the `tree` package provides classifiers and regressors based on a decision tree model.
- In Scikit-Learn the ensemble package provides Random Forest classifiers.
- In Spark, the pyspark.mllib.tree package provides DecisionTree and DecisionTreeModel as well as the RandomForest models.

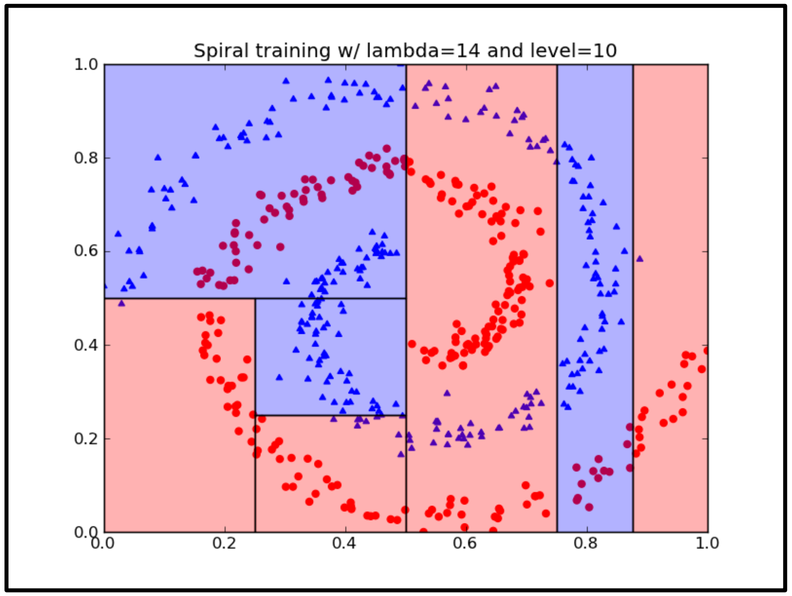
Decision Tree Example - Iris Dataset



Decision Tree Algorithm

Partition data as follows:

1. Start with whole training data
2. Select attribute along dimension that gives "best" split
3. Create child nodes based on split
4. Recurse on each child using child data until stop



What is the "best" split?

Split Metrics

Gini Impurity (used by CART)

Measures how often randomly chosen elements would be incorrectly labeled according to distribution in partition. Zero value means all cases fall into single target

Information Gain (ID3 and C5)

Uses entropy - the amount of information contained in every partition that represents some part of data.

Variance Reduction (CART)

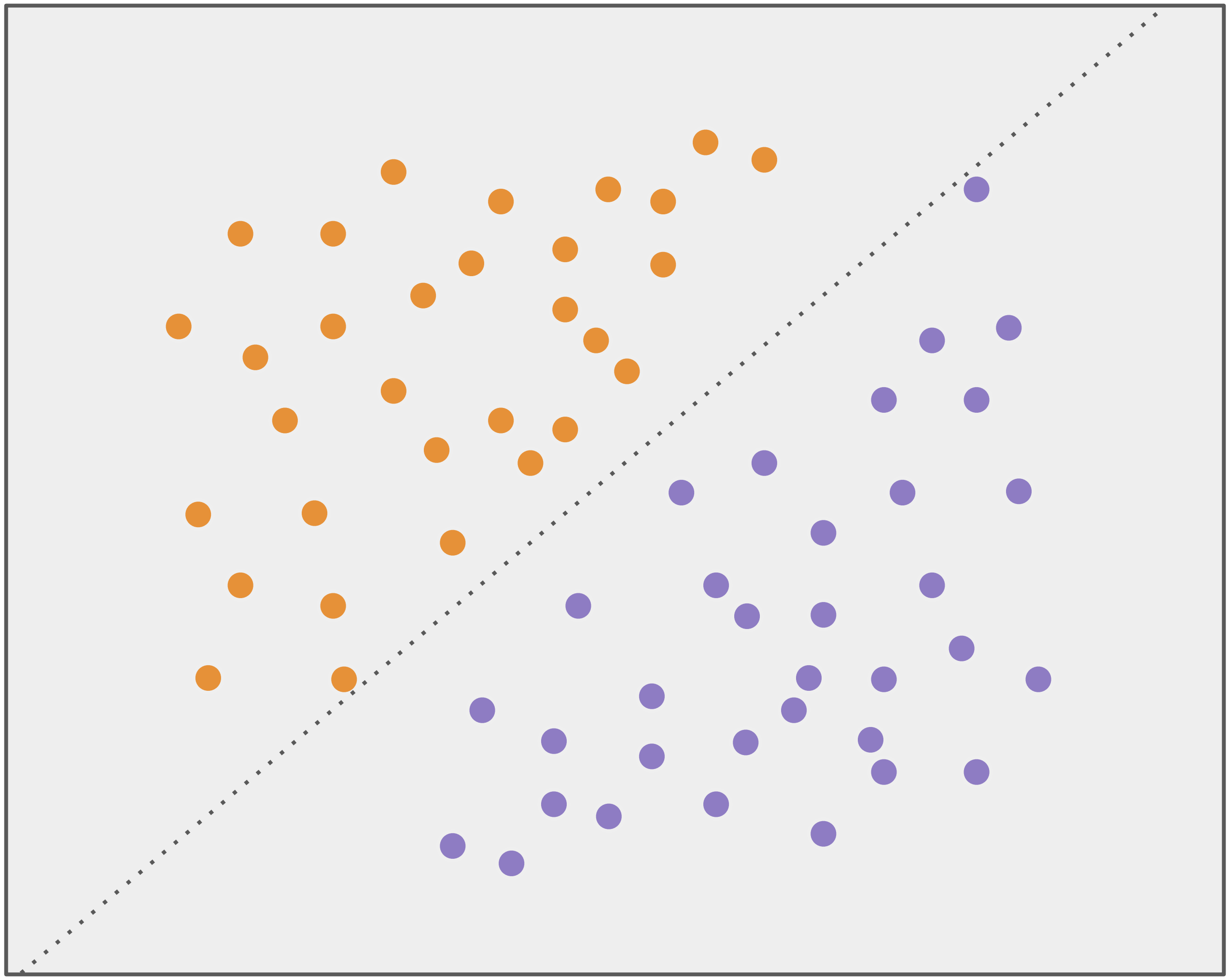
Maximally select the split where the most variance would be reduced, used for continuous values of data.

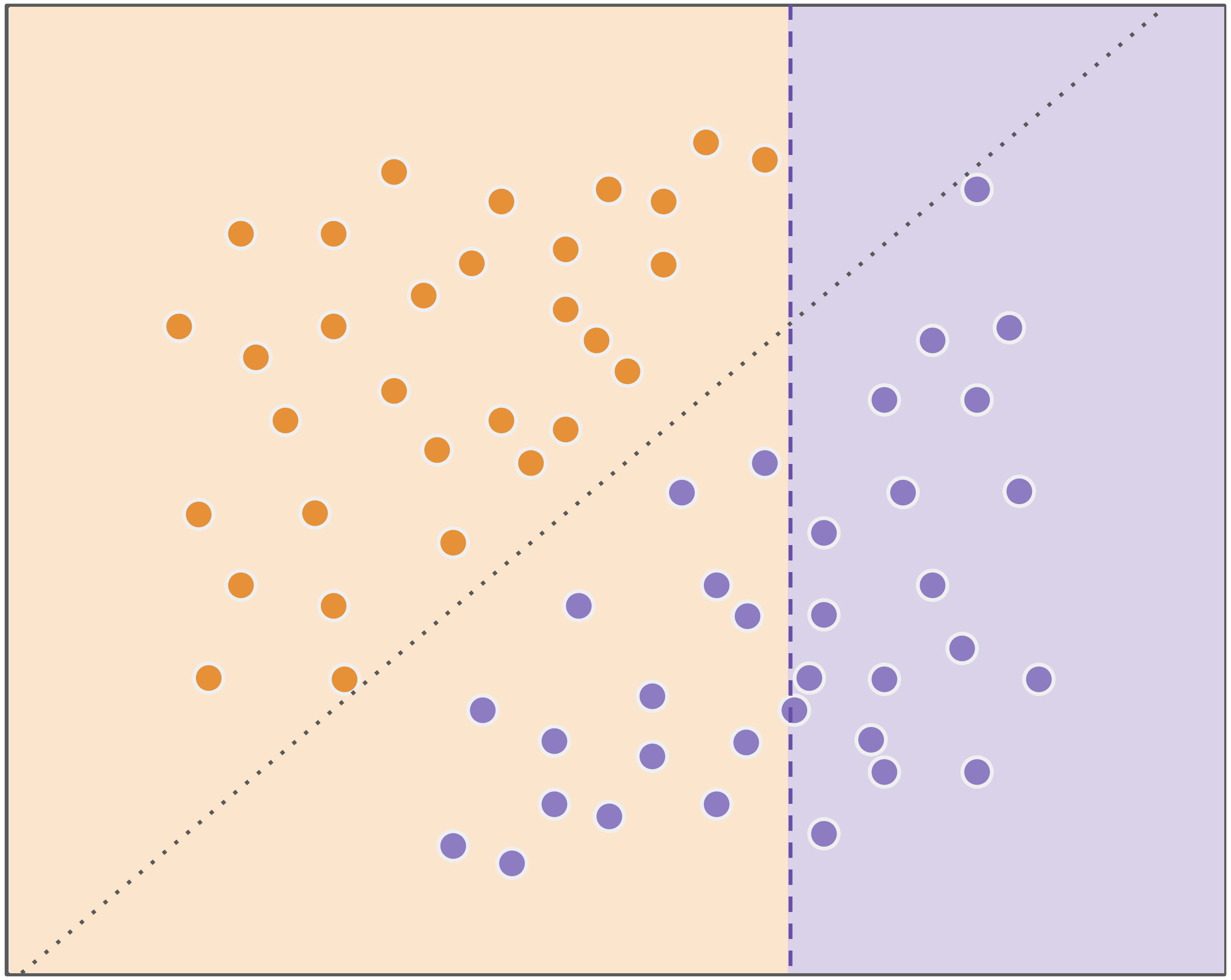
Gini

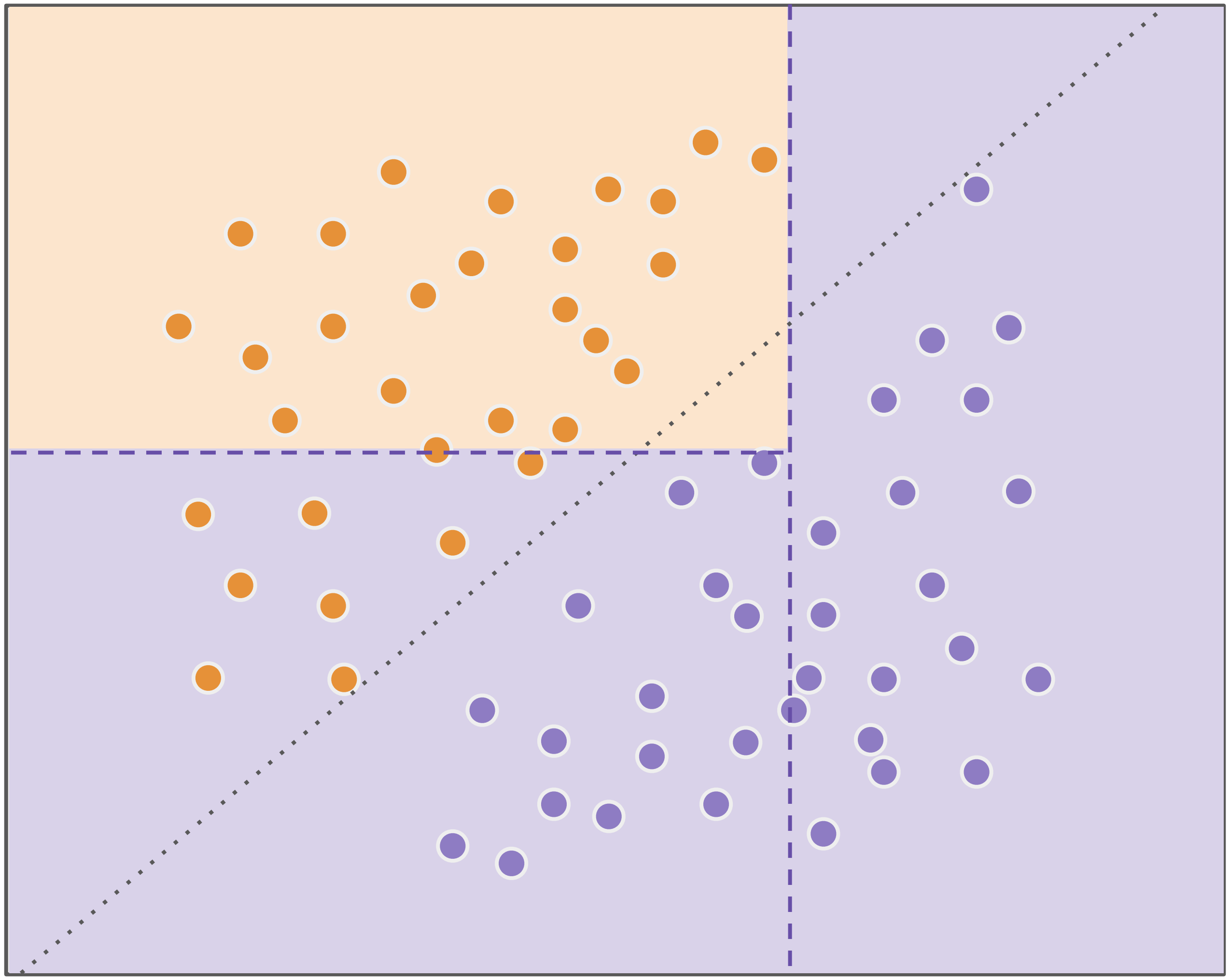
For a set of items where i ∈ {1,2, …, m} and f_i is the proportion of items in the split labeled i, the Gini Impurity is computed as follows:

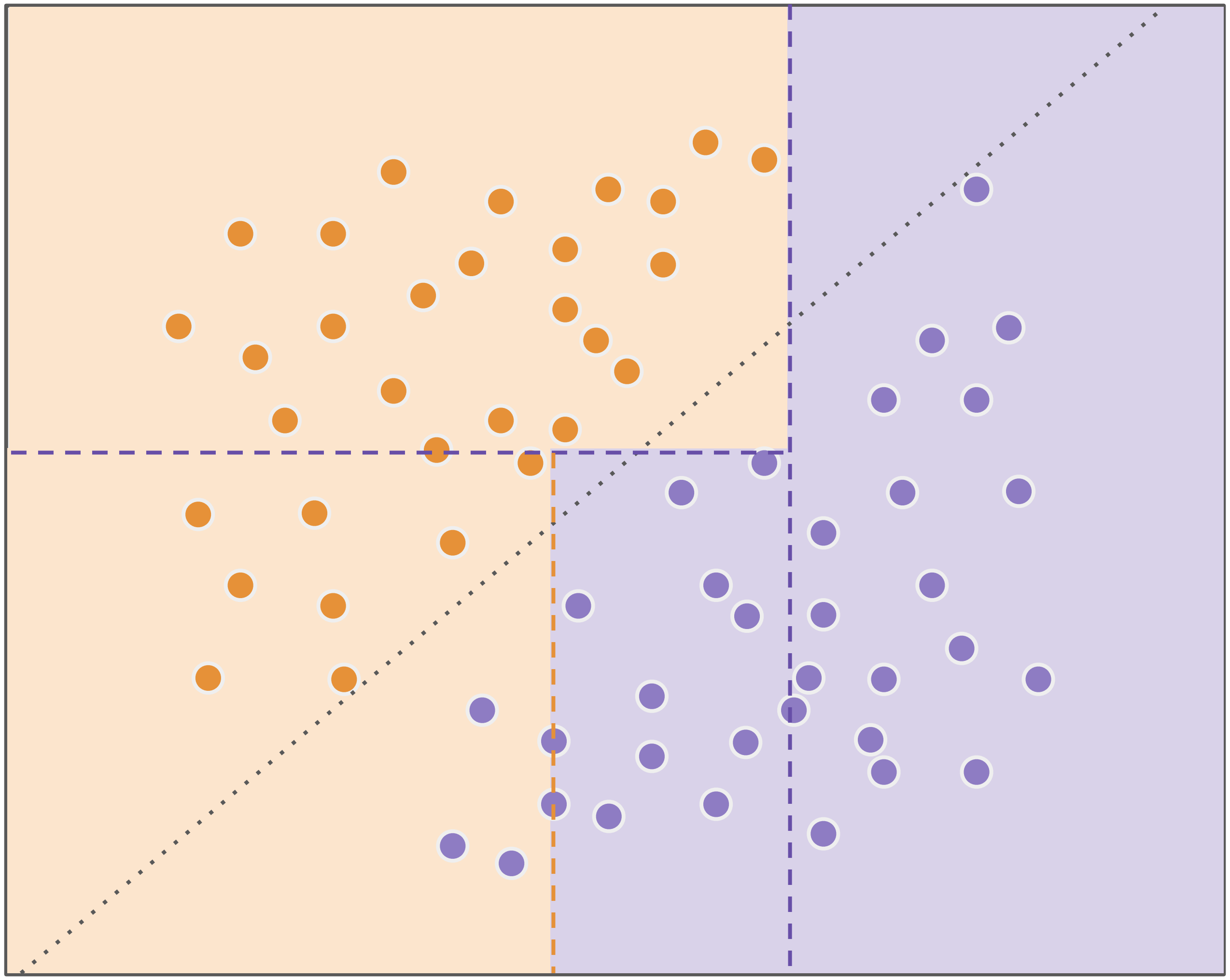
$$\boxed{I_G(f) = 1 - \sum_{i=1}^{m} f_i^2}$$

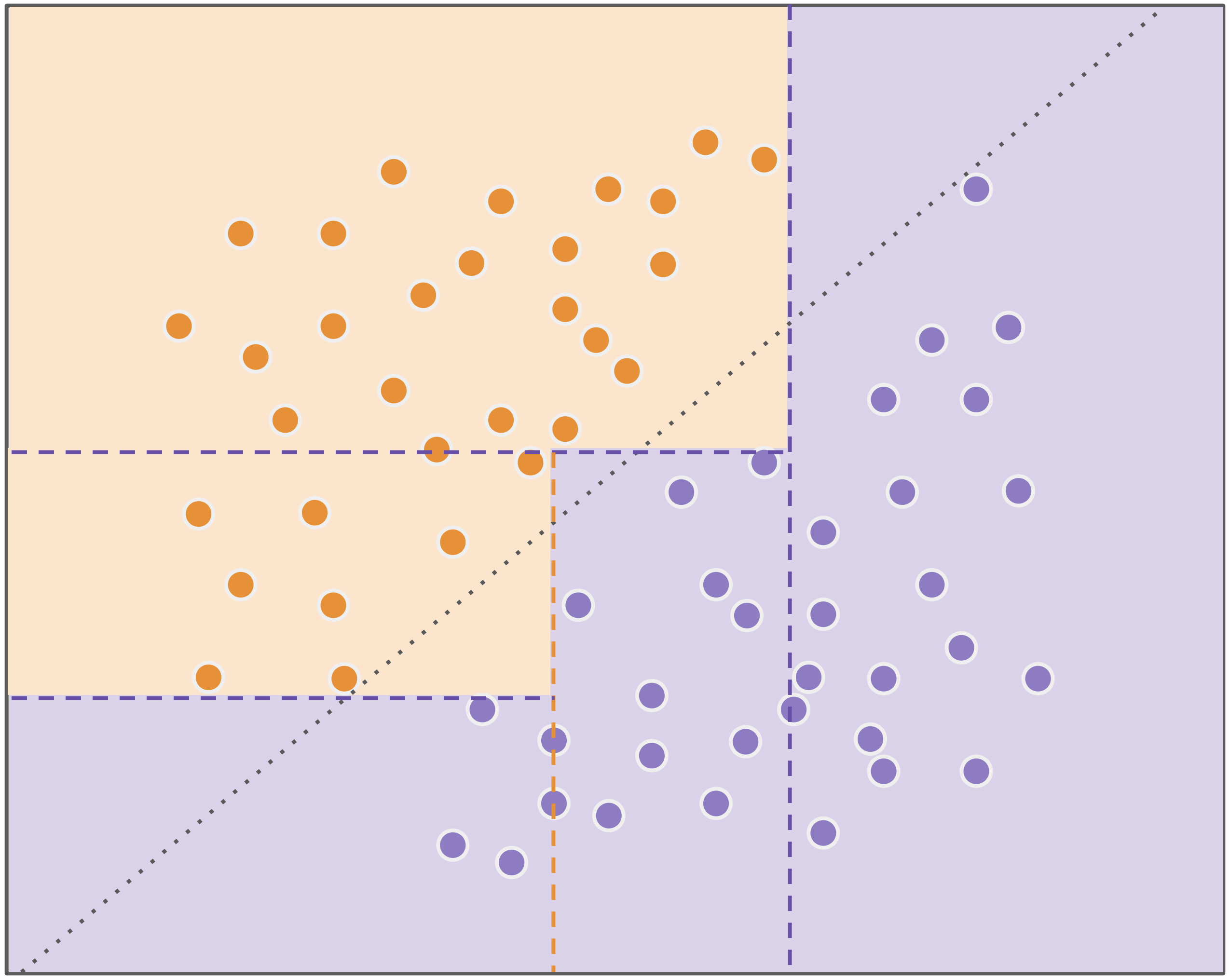
Essentially the sum of the probability of an item being chosen times the probability of a mistake being made.











Pros and Cons of Decision Trees

Pros

- Simple to understand; no representational mysticism. Trees can be drawn out.
- No need to normalize or standardize (need to deal with missing values).
- Low cost
- Handle multi-output
- Validate with Statistics!
- Handles wrong assumptions.

Cons

- Prone to overfit (deep trees don't generalize)
- Minor variations in data cause big changes in tree structure (unstable) - fix with ensemble methods.
- Create biases if some classes dominate
- Some functions are impossible to model

Tips for Using Decision Trees

Ration of samples to number of features is important, trees in high dimensional space is very likely to overfit.

Perform dimensionality reduction to give tree a better chance of finding features that are discriminative (PCA, ICA, Feature Selection)

Visualize your tree as you are training by using the export function. Use max_depth=3 as an initial tree depth to get a feel for how the tree is fitting to your data, and then increase the depth.

Tips for Using Decision Trees

Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to. Use max_depth to control the size of the tree to prevent overfitting.

Balance your dataset before training to prevent the tree from creating a tree biased toward the classes that are dominant.

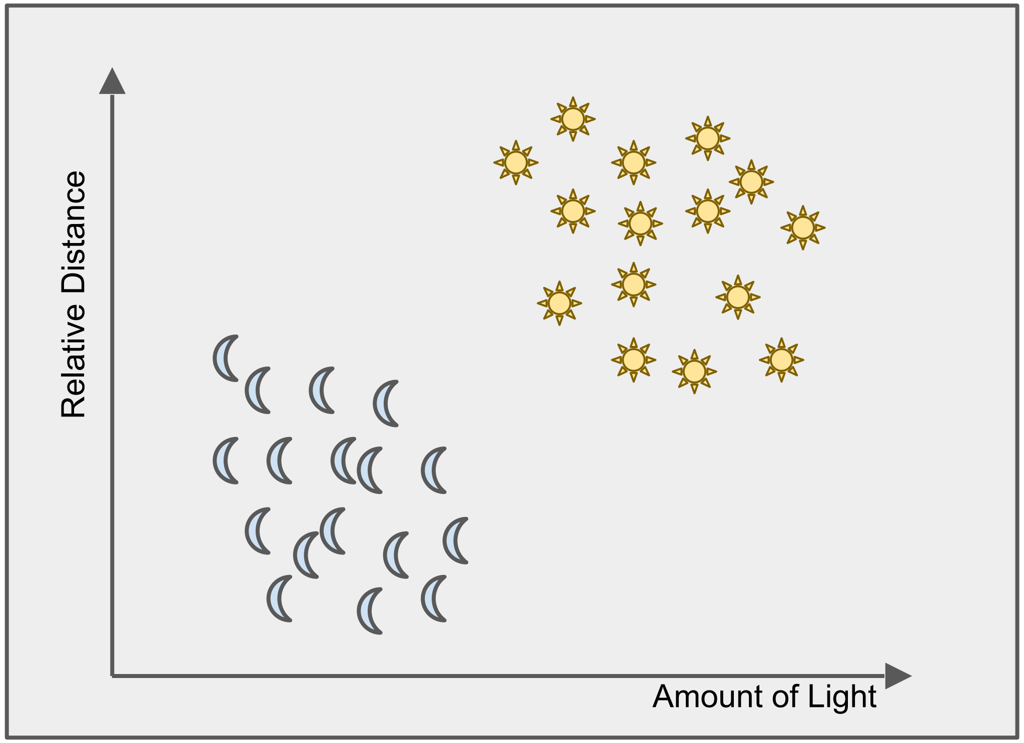
Lab: Decision Trees

Sklearn & MLlib

Support Vector Machines

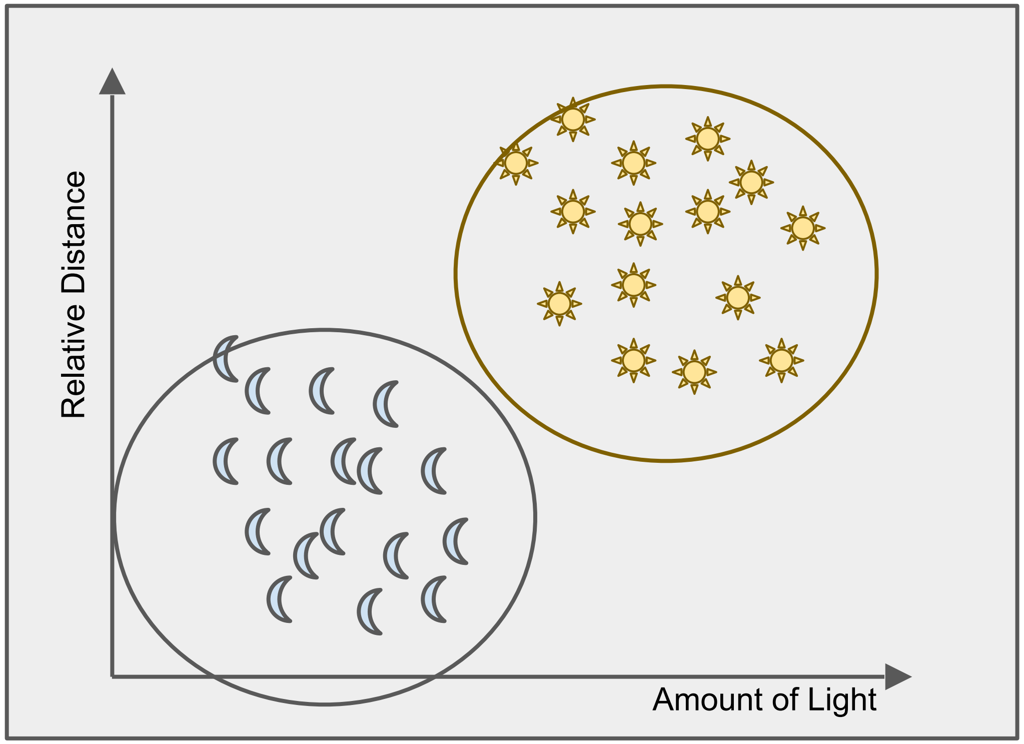
Stars or Moons?

Back to our classification problem, let's say we know we have two discernible groups (a strong hypothesis):



Stars or Moons?

Using kNN we're highly dependent on the instances, and we get centers:



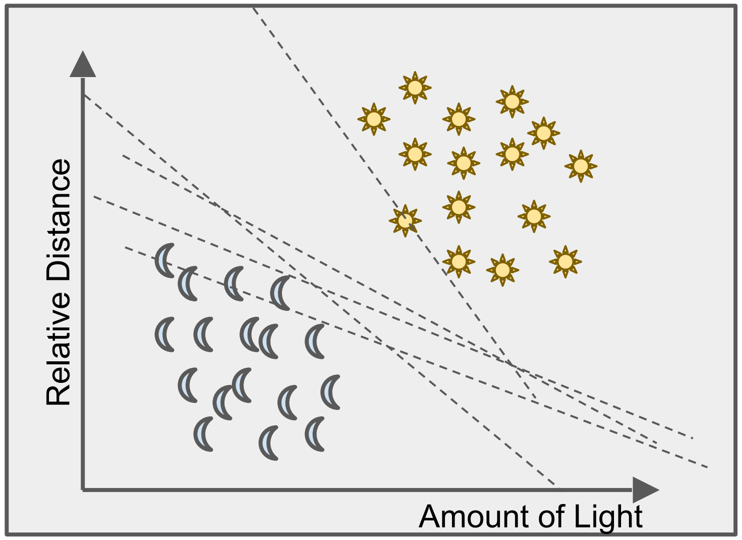
Decision Boundary

In order to *generalize* this model, we really want to find a *decision boundary* - a line between two classes of data.

How many lines fit between moons and stars?

Instead of picking an arbitrary line, we attempt to maximize the distance between classes.

Definition
In geometry a **hyperplane** is a *subspace* of one dimension less than its ambient space. If a space is 3-dimensional then its **hyperplanes** are the 2-dimensional planes, while if the space is 2-dimensional, its **hyperplanes** are the 1-dimensional lines.

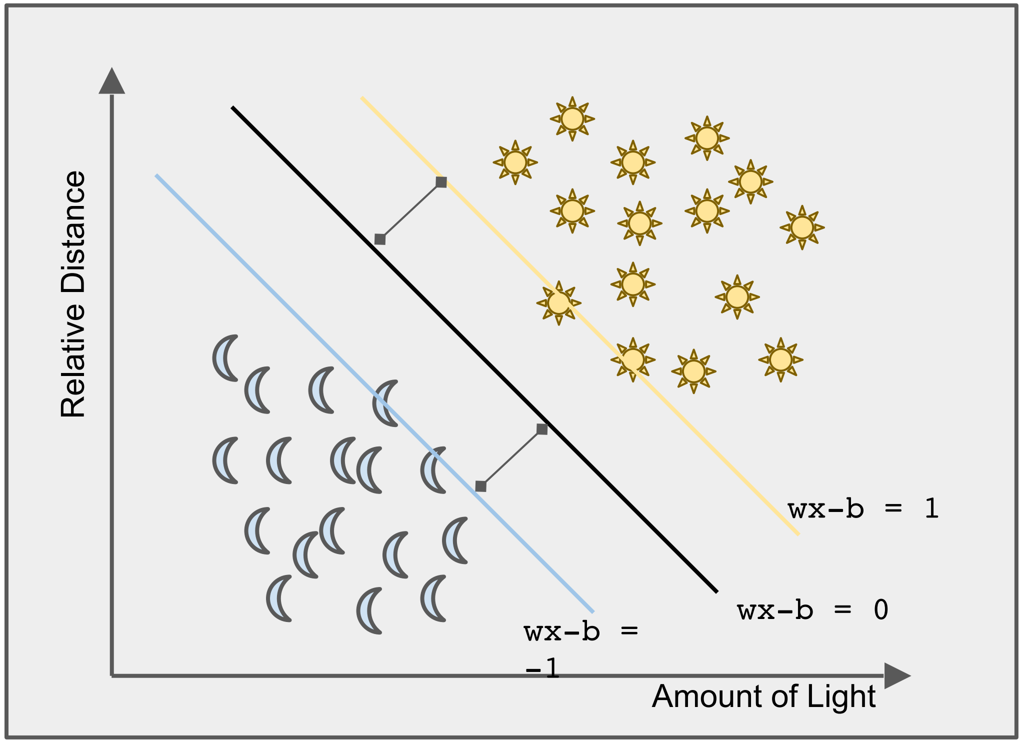


Support Vector Machines

- Introduced by Vladimir Vapnik in the 1980s
- Modern interpretation is less stringent (added slack)
- Designed to solve a two-group classification
  - boolean (true, false)
  - ids (3,4)
  - sign (1, -1)
- Desirable
  - Avoids curse of dimensionality
  - Works in high dimensional space
  - FAST to compute

Maximizing Distance

Solve for `w` for the function `wx-b=0` to determine the hyperplane maximizing the distance between two groups.

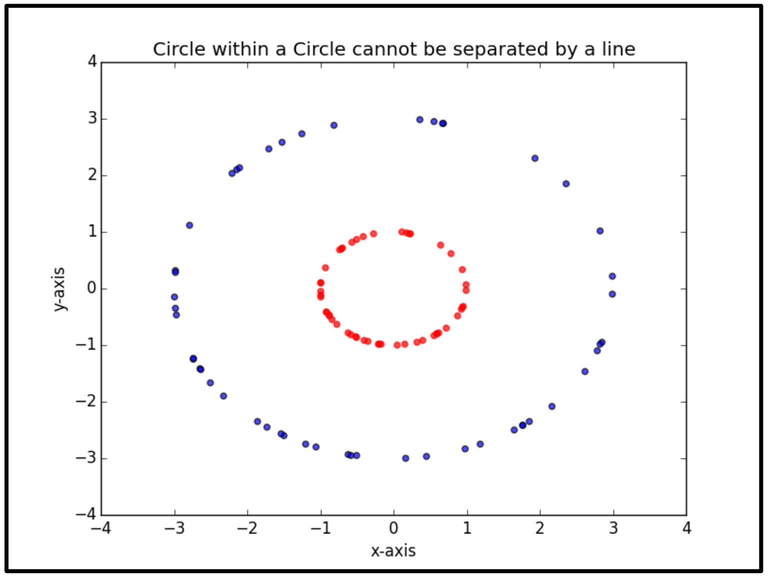


Solvable by finding the perpendicular hyperplane to all three parallel planes and dividing by 2, which happens to be:

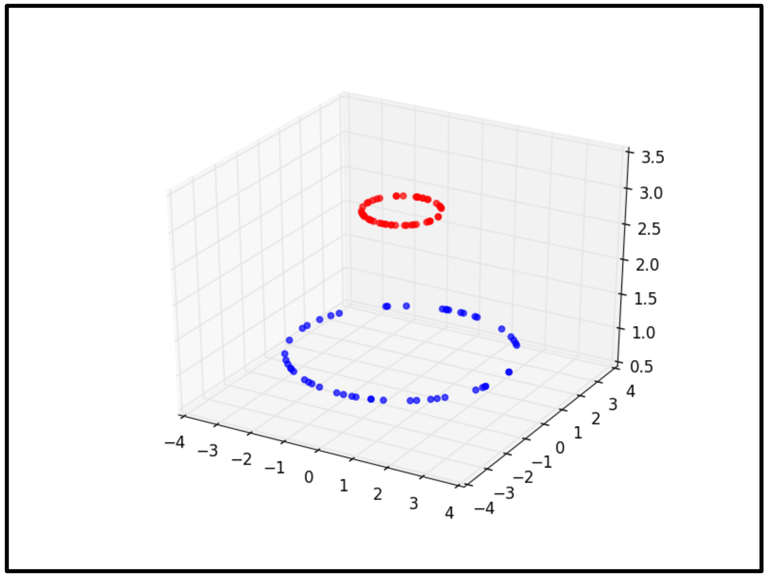
`2 / ||w||`

The Kernel Trick

- Examples are linear, real data is not (and also usually in a sparse, high-dimensional space).
- Kernel trick: transpose data into a higher dimensional space. E.g. for circles (polynomial kernel):



<x,y>



$<x^2, \sqrt{2}xy, y^2>$

Trickier Kernel

Take a linear $\mathtt{x}$ and transform it into $\phi(\mathtt{x})$ or some function that better suits the data. Points to consider:

- $\phi(\mathtt{x}_i)\phi(\mathtt{x}_j)$ might take a long time to compute
- ϕ might be a complicated function
- Extensive training data will be slow

So, we try another trick: $K(\mathtt{x}_i,\mathtt{x}_j) = \phi(\mathtt{x}_i) \cdot \phi(\mathtt{x}_j)$

Mapping the dot product + cheap function (e.g. inner dot product is already calculated) = *kernel function*

Kernel Functions

Homogeneous Polynomial
d is the degree, any number > 0
works best in most cases.

$$K(x_i, x_j) = (x_i^T x_j)^d$$

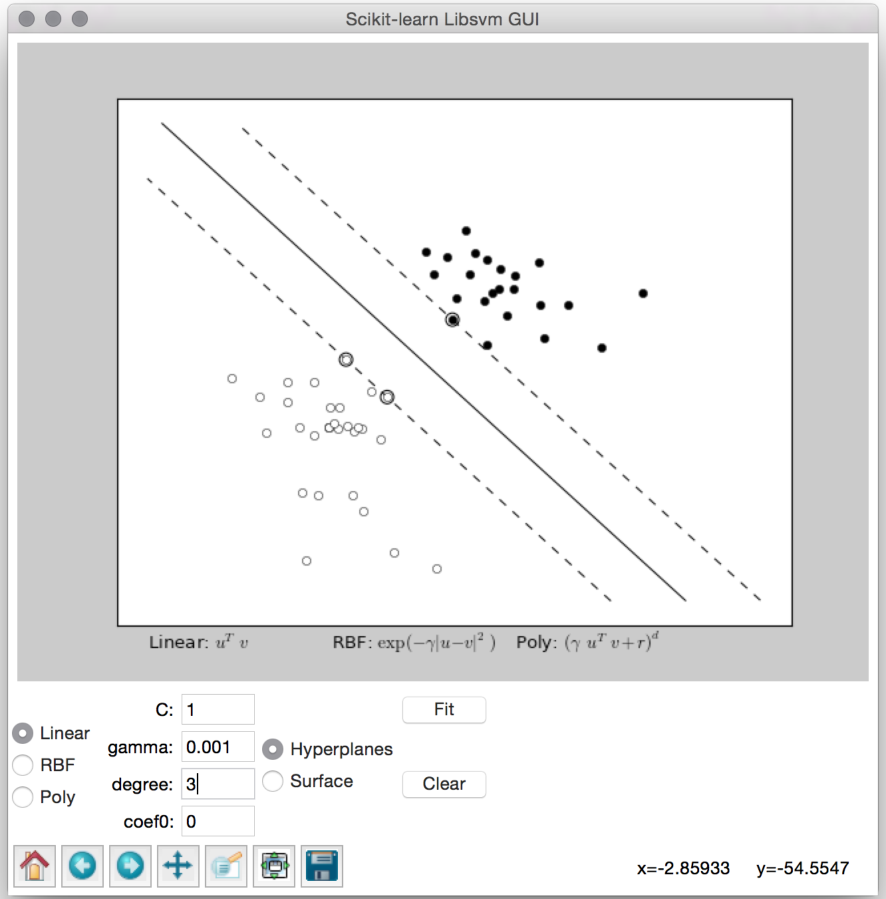
Heterogeneous Polynomial
Add a non-negative, non-zero constant, c
Increases relevance of higher order features

$$K(x_i, x_j) = (x_i^T x_j + c)^d$$

Radial Basis Function (RBF)
Often used more: high-D performance
Can create infinite new dimensions
Numerator is euclidean distance
σ is a free parameter

$$K(x_i, x_j) = exp(\frac{||x_i - x_j||_2^2}{2\sigma^2})$$

SVM Demo



[✓ Linear] RBF Poly

C: 1
gamma: 0.001
degree: 3
coef0: 0

[✓ Hyperplanes] Surface

x=-2.85933 y=-54.5547

Lab: Support Vector Machines

Sklearn & MLlib

Naive Bayes

Detecting Fraud

- Online store with at least 1000 orders per month and where 10% of orders are fraudulent
- 60 seconds to check an order, $15 per hour to a customer service representative = $3k per year!
- Goal: construct a model of probability that detects if an order is 50% likely to be fraudulent (reduce checking)

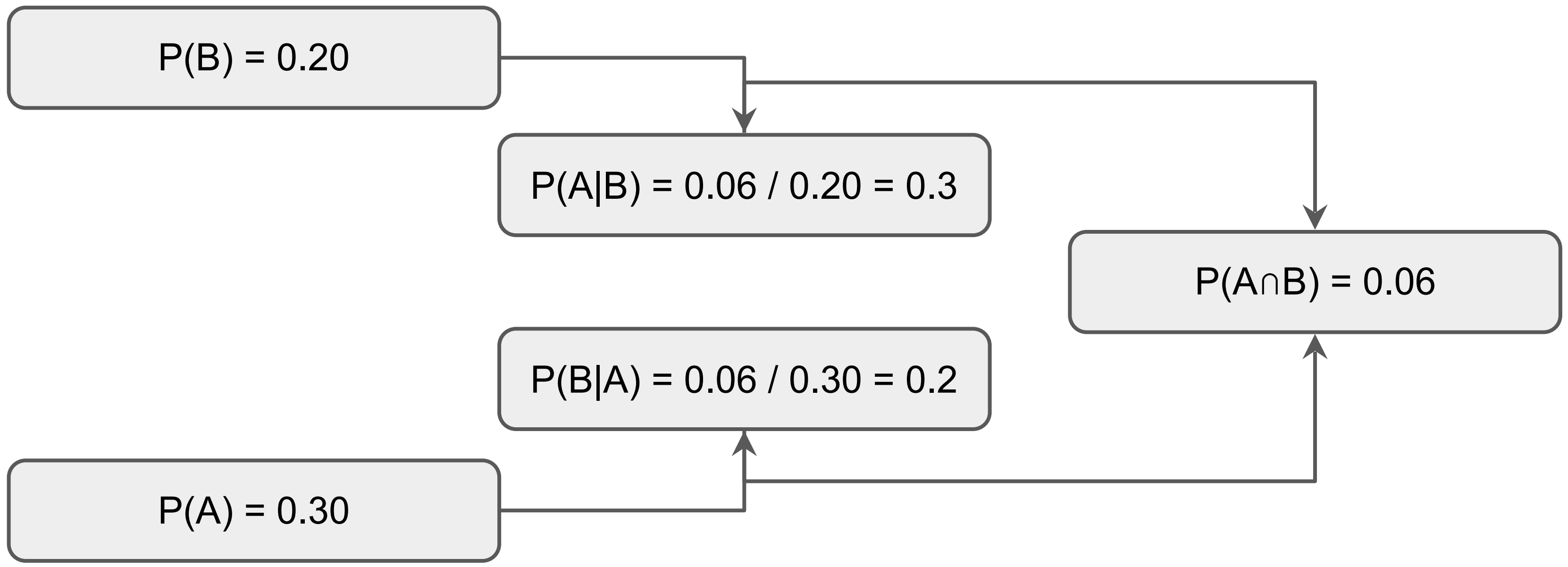
How likely is an order to be fraudulent?

What if we have more information, e.g. frauds are likely to use gift cards and promo codes?

Conditional Probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

The probability of A *given* B is equal to the probability of A *and B* divided by the probability of B.



P(A|B) sits between P(A), P(B) and P(A∩B)

Detecting Fraud

We can now frame our problem in terms of the following probabilities:

$$P(Fraud|Giftcard) = \frac{P(Fraud \bigcap Giftcard)}{P(Giftcard)}$$

$$P(Fraud|Promo) = \frac{P(Fraud \bigcap Promo)}{P(Promo)}$$

But how do we compute P(Fraud∩Giftcard)?

Bayes Theorem

In the 18th century, Reverend Thomas Bayes came up with the research that Pierre-Simon Laplace would beautify:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Because:

$$P(B|A) = \frac{\frac{P(A \cap B)P(B)}{P(B)}}{P(A)} = \frac{P(A \cap B)}{P(A)}$$

Bayes and Fraud Detection

Now we can compute the following probabilities:

$$P(Fraud|Giftcard) = \frac{P(Giftcard|Fraud)P(Fraud)}{P(Giftcard)}$$

$$P(Fraud|Promo) = \frac{P(Promo|Fraud)P(Fraud)}{P(Promo)}$$

And these are easily computed by frequency. Let's say that gift cards are used 10% of the time, and 60% of gift card use is fraudulent: What is the probability of P(Fraud|Giftcard)?

Bayesian Classification

You might have guessed it: use of a promo or gift card during fraud are *features* of our Bayesian model.

What we want to solve now is: `P(Fraud|Giftcard,Promo)`

Using *joint probability* we can derive the *chain rule*:

$$P(A_1,A_2,\ldots,A_n)=P(A_1)P(A_2|A_1)P(A_3|A_1,A_2)\cdots P(A_n|A_1,A_2,\ldots,A_{n-1})$$

Bayesian Classification

With Bayes, we can feed lots of information into our model, which is now as follows:

$$P(Fraud|Giftcard,Promo) = \frac{P(Giftcard,Promo|Fraud)P(Fraud)}{P(Giftcard,Promo)}$$

Let's Be Naive!

Our model is still tough to solve, but let's look at the denominator - `P(Giftcard, Promos)` - this doesn't seem to be related to fraud.

We will make an ***assumption of independence*** and drop the denominator from our model. Using the chain rule:

$$P(Fraud, Giftcard, Promo) = P(Fraud)P(Gift|Fraud)P(Promo|Fraud, Gift)$$

Let’s Be Naive!

Now we have to deal with P(Promo|Fraud,Gift), using assumption again, along with a parameter, Z (sum of probabilities of all classes) we get the following:

$$P(Fraud|Giftcard,Promo)=\frac{1}{Z}P(Fraud)P(Gift|Fraud)P(Promo|Fraud)$$

Classification

Building our model

- Compute the probabilities of all features from a training set
- Compute the probabilities of all classes from training set
- Compute the parameter Z
- Build a truth table

Using our model

- Compute “probabilities” of input vector
- Use truth table, trained probabilities and parameters to compute likelihood of class

	Fraud	Not Fraud
Gift Card	0.6	0.1
Promos	0.5	0.3
Class	0.1	0.9

Smoothing

What happens if we receive new information?

If the information has a probability = 0, then the independence assumption will cause skew.

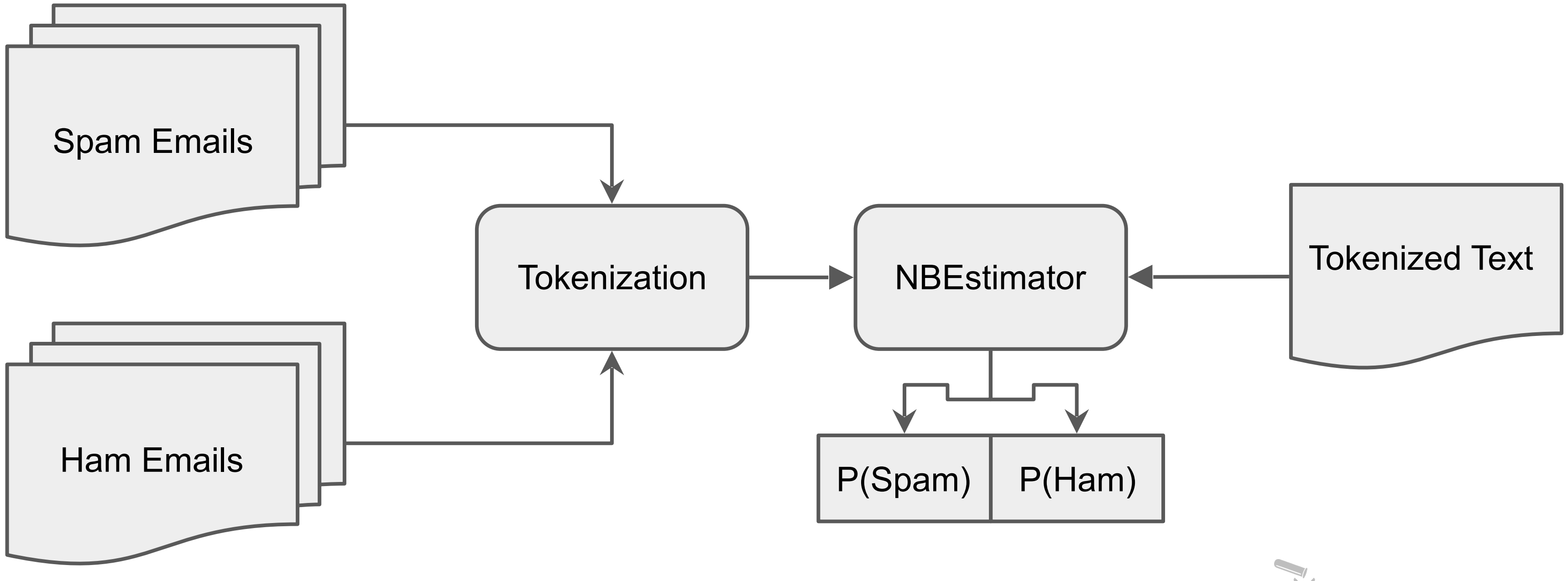
Smoothing: Donating some of the probability density to unknown information.

Easiest smoothing technique: *laplacian (additive)*
Simply give all unknown items a frequency of 1 - then recompute probabilities accordingly.

Text Classification with NB

Spam/Ham is the canonical Naive Bayesian classifier for text and is popular because of it's wide use.

Compute using "bag of words" - each word is a feature.



DistrictDataLabs

Lab: Naive Bayes

Sklearn & MLlib

Model Evaluation

Methods of Model Evaluation

- Training and testing on the same data
- Train/test split
- K-fold cross validation

Training and Testing on the Same Data

- Rewards overly complex models that “overfit” the training data and won’t necessarily generalize.

DON’T DO THIS
(REALLY, DON’T)

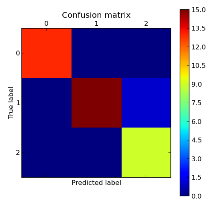
Train/Test Split

- Split the dataset into two pieces, so that the model can be trained and tested on different data; typically 80/20.
- Better estimate of out-of-sample performance, but still a "high variance" estimate
- Useful due to its speed, simplicity, and flexibility

K-Fold Cross Validation

Assess how model will generalize to independent data set (e.g. data not in the training set).

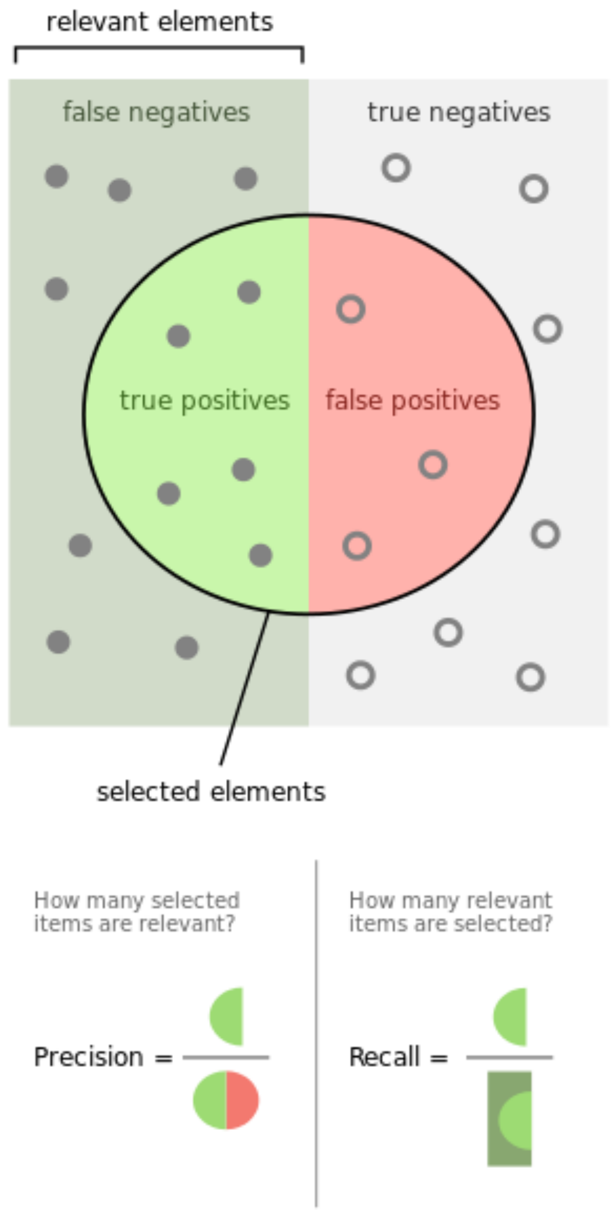
1. Divide data into training and test splits
2. Fit model on training, predict on test
3. Determine *accuracy*, *precision* and *recall*
4. Repeat *k* times with different splits then average as *F1*



	Predicted Class A	Predicted Class B	
Actual A	True A	False B	#A
Actual B	False A	True B	#B
	#P(A)	#P(B)	total

F1 Score

- A measure of a test's accuracy
- Considers the precision (p) and recall (r) of the test to compute the score.



https://en.wikipedia.org/wiki/Precision_and_recall

F1 score =
`2 * ((precision * recall) / (precision + recall))`

precision =
`true positives / (true positives + false positives)`

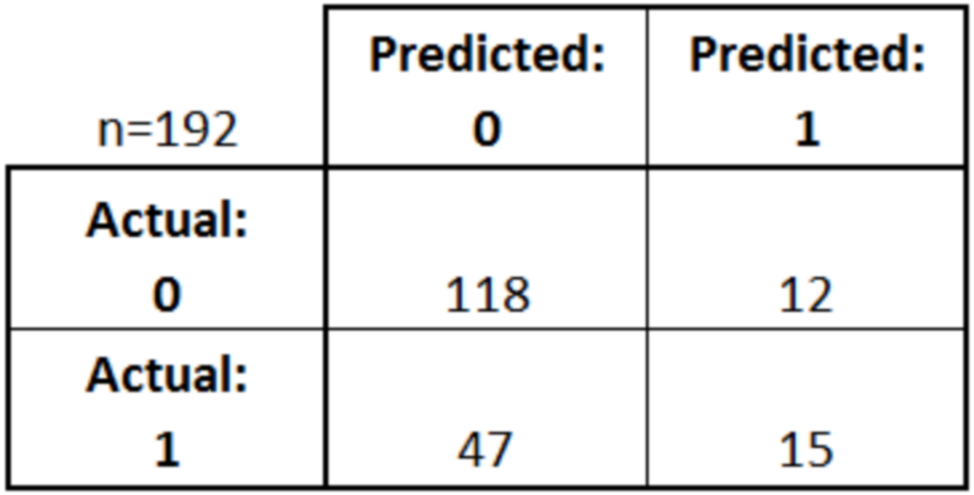
recall =
`true positives / (false negatives + true positives)`

accuracy =
`true positives + true negatives / total`

Confusion Matrix

```
print(metrics.confusion_matrix(y_test, y_pred_class))
```

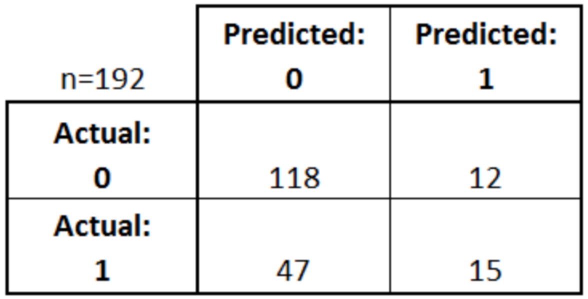
```
[[118  12]
 [ 47  15]]
```



n=192	Predicted: 0	Predicted: 1
Actual: 0	118	12
Actual: 1	47	15

Confusion Matrix

```
[[118  12]
 [ 47  15]]
```

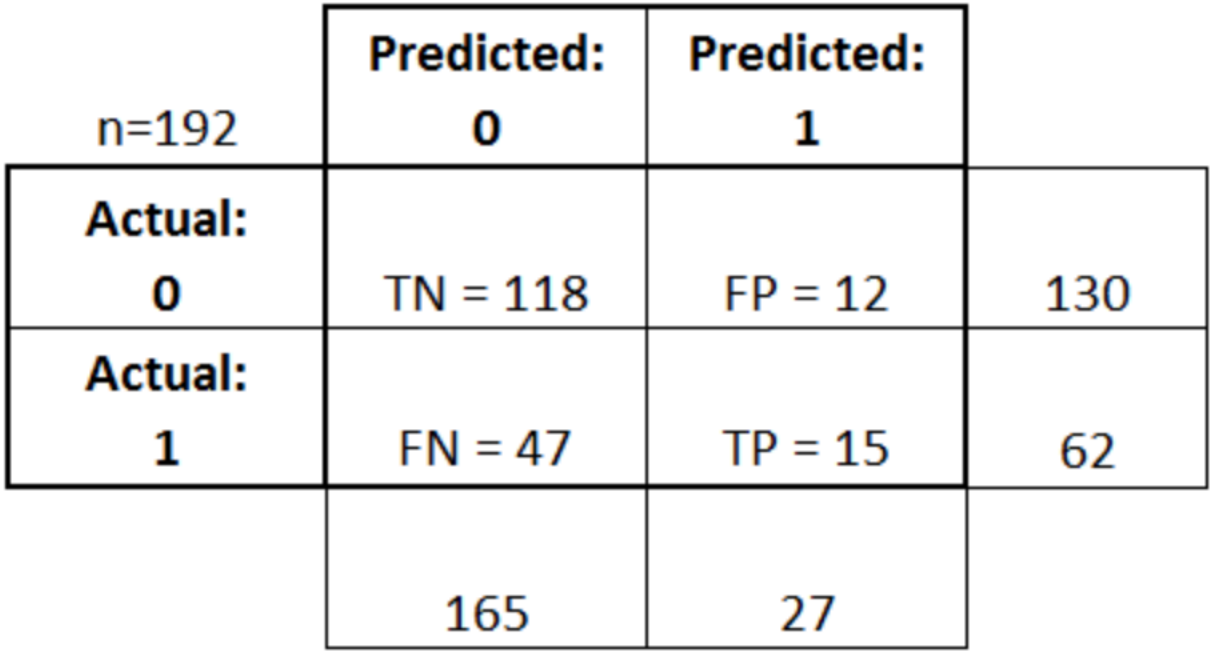


n=192	Predicted: 0	Predicted: 1
Actual: 0	118	12
Actual: 1	47	15

0: negative class
1: positive class

- True Positives: correctly predicted a positive (15)
- True Negatives: correctly predicted a negative (118)
- False Positives: incorrectly predicted a positive (12) - Type I Error
- False Negative: incorrectly predicted a negative (47) - Type II Error

Confusion Matrix



n=192	**Predicted: 0**	**Predicted: 1**	
Actual: 0	TN = 118	FP = 12	130
Actual: 1	FN = 47	TP = 15	62
	165	27	

0: negative class
1: positive class

What a Confusion Matrix Gives Us

- Classification accuracy: overall how often is the classifier correct?
- Classification error: overall, how often is the classifier incorrect?
- False Positive Rate: when the actual value is negative, how often is the prediction incorrect?
- Precision: when a positive value is predicted, how often is the prediction correct?

What a Confusion Matrix Gives Us

- Sensitivity: when the actual value is positive, how often is the prediction correct?
  - How "sensitive" is the classifier to detecting positive instances?
  - Also known as "True Positive Rate" or "Recall"
  - All Positive = TP + FN
- Specificity: when the actual value is negative, how often is the prediction correct?
  - How "specific" (or "selective") is the classifier in predicting positive instances?
  - All Negative = TN + FP