Machine Learning and Deep Learning
Lecture-05

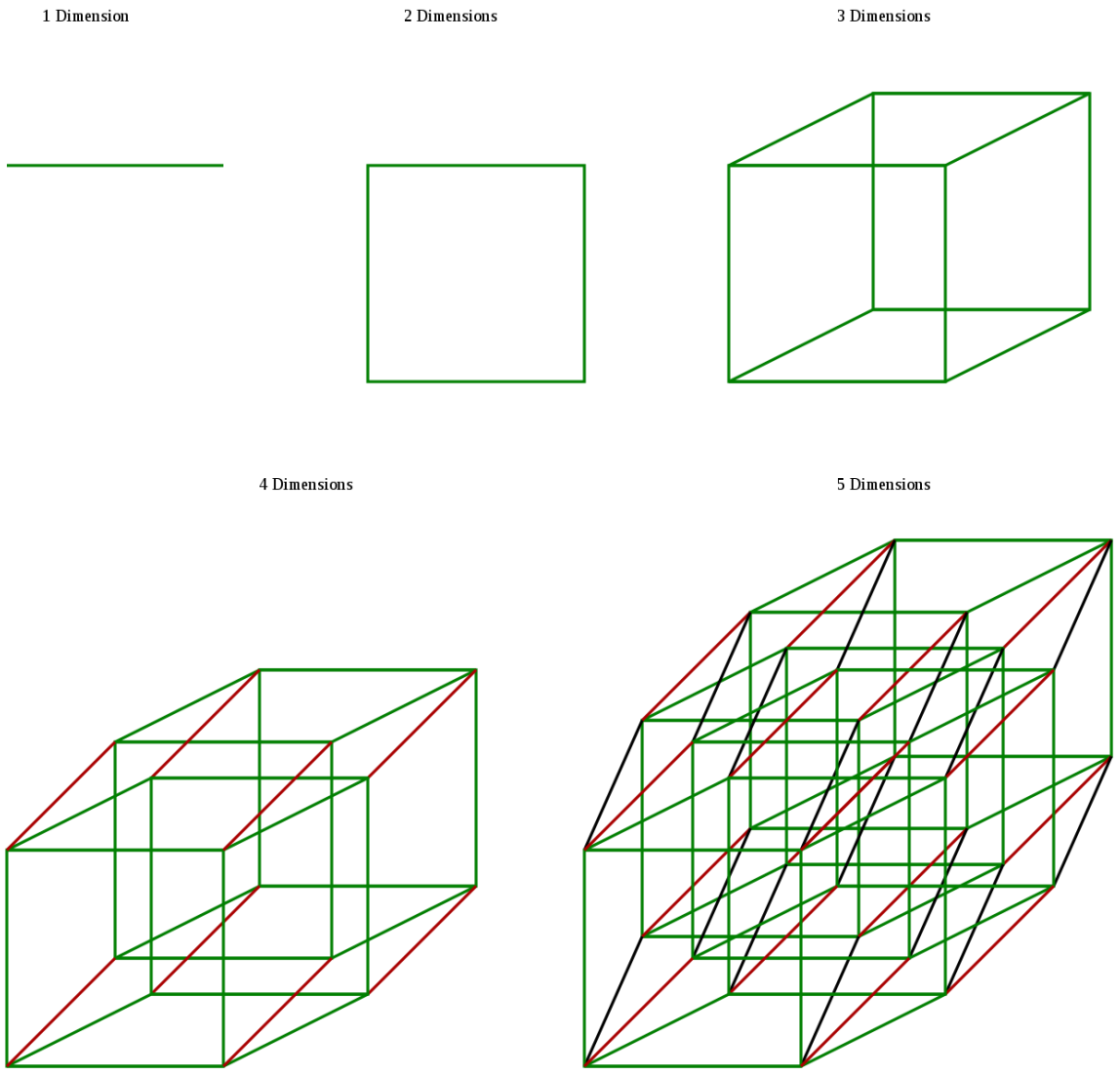
By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Overview

- Dimensionality Reduction
- Principal Component Analysis (PCA)
- Singular value decomposition (SVD) [Not in Lecture Plan]
- Decision Trees
- Random Forests

Dimensionality Reduction

Dimension

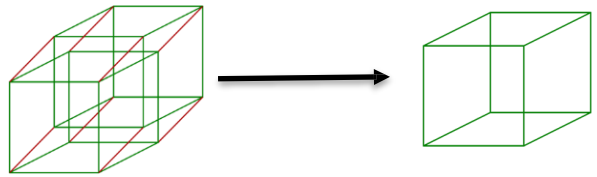


- Dimension of an object can be defined as minimum number of coordinates needed to specify any point within it.

Src: Wikipedia

Dimensionality Reduction

- Process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of a data set.
- Goal: To reduce dimensions by removing redundant and dependent features.
- How? By transforming features from higher dimensional space to a lower dimensional space.



- Approaches: Unsupervised and Supervised

Dimensionality Reduction

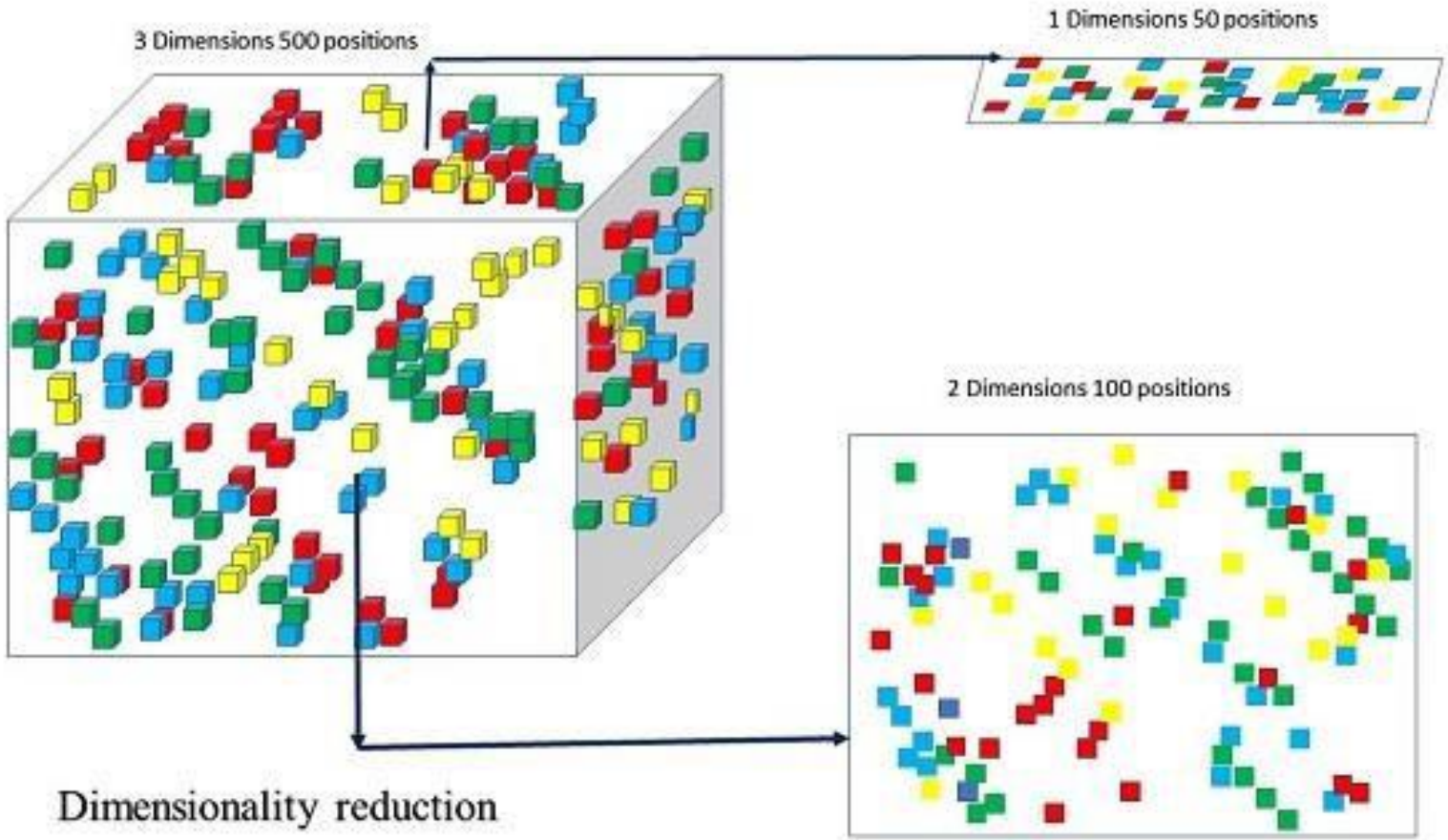
- Approaches: Unsupervised and Supervised
  - Unsupervised where no need for labeling classes of data.
    - Independent Component Analysis (ICA)
    - Non-negative Matrix Factorization (NMF)
    - Principal Component Analysis (PCA)
      - Ideal for visualization and noise removal.

  - Supervised where class labels are considered.
    - Mixture Discriminant Analysis (MDA)
    - Linear Discriminant Analysis (LDA)
      - Ideal for biometrics, Bioinformatics, and chemistry.

In a nutshell

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

Dimensionality Reduction



- Application:
  - Machine learning
  - Data mining
  - Bioinformatics
  - Biometric
  - Information Retrieval

Example of multidimensional cube. Perform slicing operations to separate dimensions.

Image Src: https://www.c-sharpcorner.com/article/a-quick-overview-of-machine-learning-tasks/

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

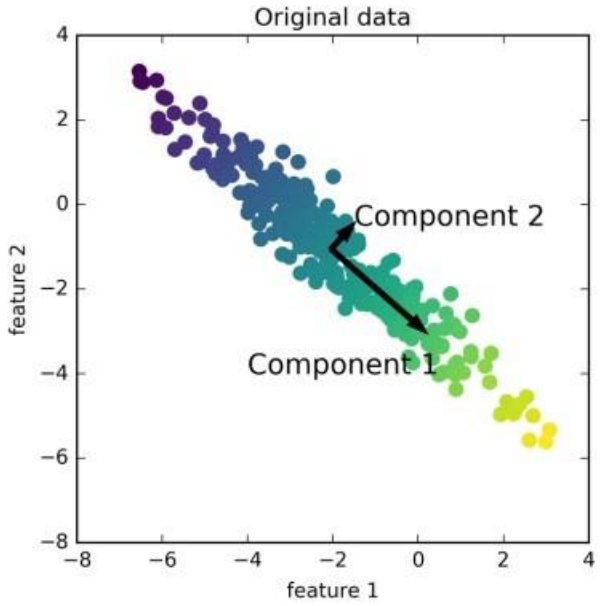
- PCA is a
  - popular technique for dimensionality reduction.
  - "classical" approach only characterize linear sub-spaces in data.

- Involves a dataset with observations on numerical variables.
  - An exploratory data analysis tool.
  - A simple, non-parametric method of extracting relevant information from data sets.

Principal Component Analysis (PCA)

- PCA reduce dimension by exposing underling information in data sets.
  - An unsupervised approach.
  - Explain most of variability in data with a smaller number of variables.
  - Identifies axis that accounts for largest amount of variance in training set.

- PCA rotates dataset to get statistically uncorrelated features.
- Rotation followed by selection of subset of new features, based on their importance.



Principal Component Analysis (PCA)

- PCA is influenced by magnitude of each variable.

- First PC (PC_1) of data set is linear combination of features that has the largest variance.
- Second PC (PC_2) is linear combination of data sets that has maximal variance out of all linear combinations that are uncorrelated with PC_1.

- PCs are uncorrelated with each other.
- PCs form a basis of new space (true for irrespective of dimensions).

```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
```

Principal Component Analysis (PCA)

- Primary PCA calculation steps:
  - Calculate covariance matrix.
  - Calculate ordered eigenvalues and eigenvalues of the matrix.

- Application:
  - Neuroscience
  - Computer graphics
  - Atmospheric science.

Principal Components (PCs)

- Overall PC calculation process:
  - For each PC:
    - PCA finds a zero-centered unit vector pointing in the direction of PC.
    - Direction of unit vectors returned by PCA is not stable*.

  - If you perturb training set slightly and run PCA again
    - Unit vectors may point in opposite direction as original vectors.
      - Still, they will lie on same axes.

*Since two opposing unit vectors lie on the same axis.

Principal Component Analysis (PCA)

- How did we know to use two PCs?
  - Proportion of variance explained (PVE) measures exactly what percentage of variance should retain in these PCs.
  - PVE of m^{th} PC is calculated roughly by:
    - (taking the m^{th} eigenvalue)/(number of PCs).

- How many PCs to Use?
  - A general n(observations)xp(variables) data matrix X, there are up to min(n- 1,p) PCs.

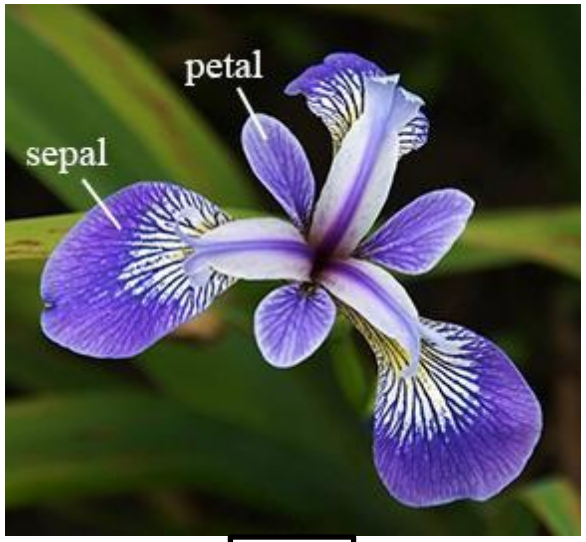
- No fixed method to determine how many PCs to use.

Demo_PCA_2.ipynb

```
# Access the first principal component (PC1)
pc1 = pca.components_[0]

# Get feature names and their corresponding contributions to PC1
feature_contributions = pd.DataFrame(pc1, index=X.columns, columns=["PC1"])
print("Features contributing to the first principal component (PC1):")
print(feature_contributions )


# Sort the features by the absolute value of their contributions to PC1
sorted_features = feature_contributions.abs().sort_values(by="PC1", ascending=False)

# Get the names of the features that contribute to the first principal component
top_features = sorted_features.index.tolist()

# Print out the feature names sorted by contribution
print("Features contributing to the first principal component (PC1):")
print(top_features)
```

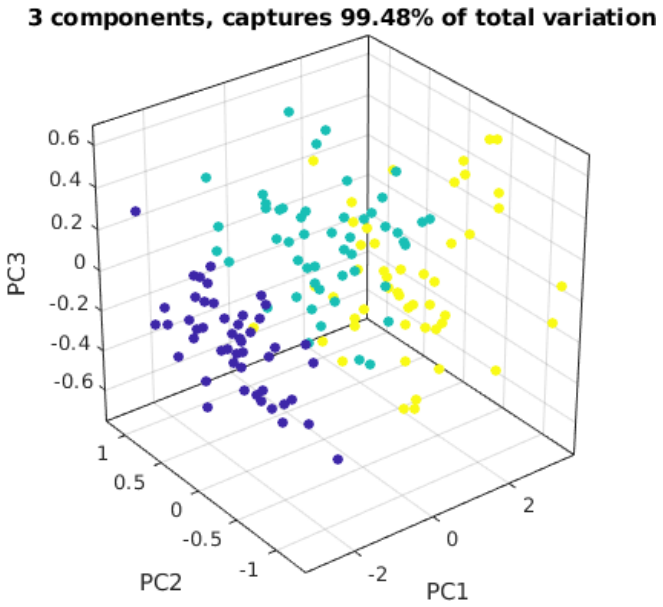
```
Features contributing to the first principal component (PC1):
                    PC1
Month          0.002627
Year           0.000108
Patient_Visits  0.999997
Features contributing to the first principal component (PC1):
['Patient_Visits', 'Month', 'Year']
```

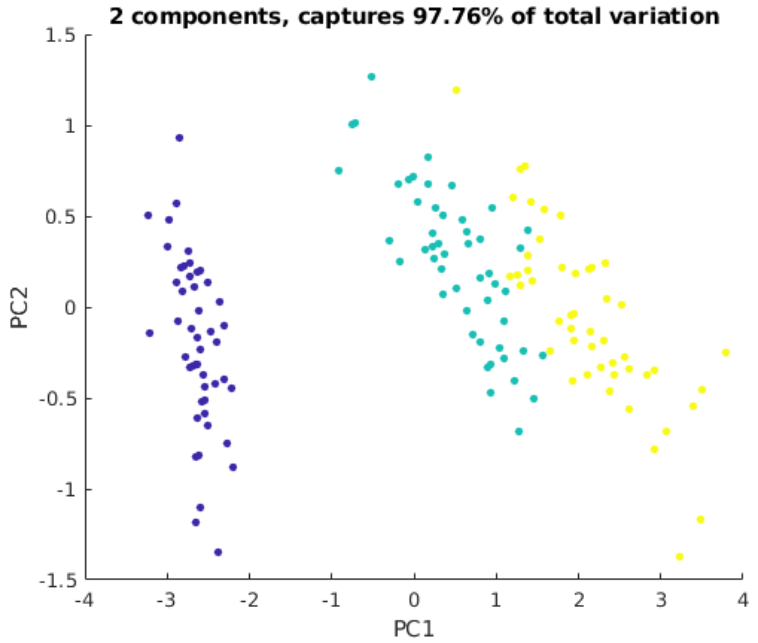
PCA Example: Iris data



- Total 150 iris flowers.
- Measurement: sepal length, sepal width, petal length, petal width

Flowers belong to three different species: blue, green, yellow

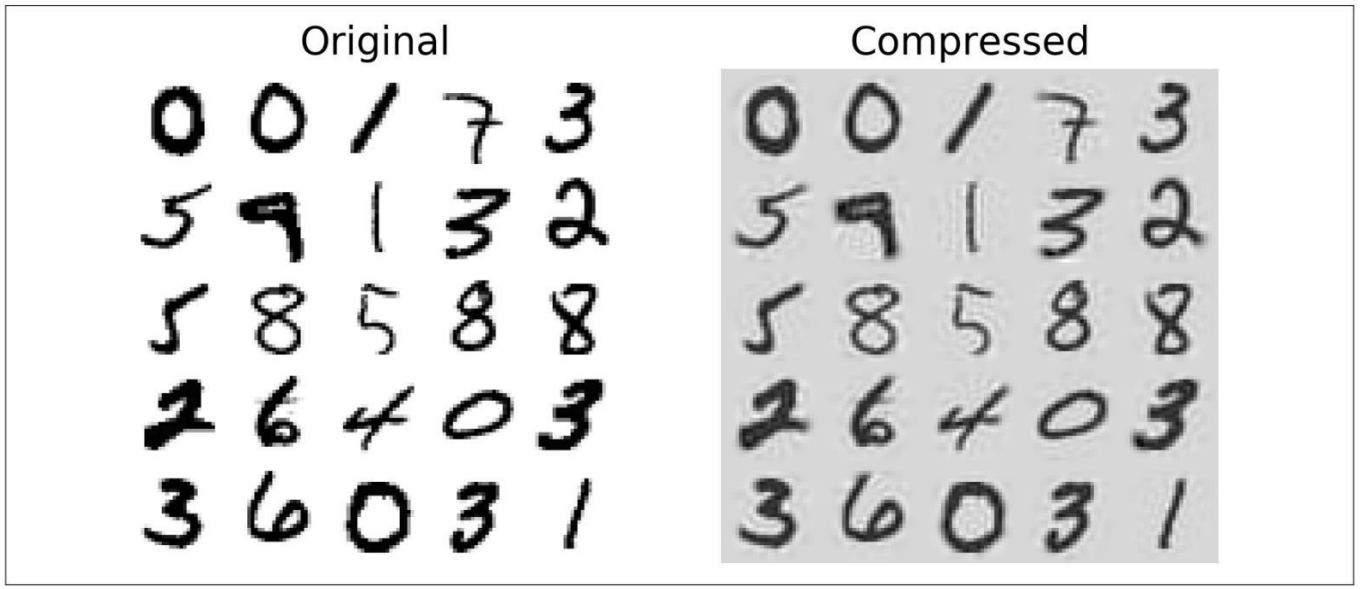




https://www.math.umd.edu/~petersd/666/html/iris_pca.html

PCA Example: MNIST Compression

- Apply PCA to MNIST dataset while preserving 95% of its variance.
- Each instance have just over 150 features, instead of the original 784 features.
- Dataset is now less than 20% of its original size.




```
pca = PCA(n_components = 154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

MNIST compression that preserves 95% of the variance

Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019.

PCA Types

- Types of PCA:
  1. Randomized PCA quickly finds an approximation of the first d principal components.
     - Issue: whole training set need to fit in memory.
  2. Incremental PCA (IPCA) splits the training set into mini-batches and feed an IPCA algorithm one mini-batch at a time.
  3. Kernel PCA helps to perform complex nonlinear projections for dimensionality reduction.

Incremental & Kernel PCA

- IPCA finds a similar projection of data to PCA by processing few samples at a time.
- Stores estimates of component and noise variances to update variance ratio incrementally.

- Kernel PCA can find a projection of the data that makes data linearly separable.
- Kernel PCA supports both transform and inverse transform.

Singular Value Decomposition (SVD)

[Not in Lecture Plan]

Singular Value Decomposition (SVD)

- SVD is a method for
  - transforming correlated variables into a set of uncorrelated set
    - Exposes various relationships among original data items.
  - Identifying and ordering dimensions along which data points exhibit most variation.
    - Perform data reduction.
  - Power Method is used to compute SVD.

Singular Value Decomposition (SVD)

- Basic steps of SVD:
  - Consider a high dimensional, highly variable set of data points.
  - Reduce it to a lower dimensional space that exposes substructure of original data (more clearly).
  - Orders it from most variation to least.

Singular Value Decomposition (SVD)

- SVD decomposes any matrix (A) into three matrixes with special properties and whose multiplication gives back the same matrix: $A_{mn} = U_{mm}S_{mn}V^{T}_{nn}$
  - U: Orthogonal matrix
  - S: Diagonal matrix
  - V^{T}: Transpose of an orthogonal matrix

- Note: $U^{T}U = I$ (Identity Matrix), $V^{T}V = I$
  - Columns of U are orthonormal eigenvectors of AA^{T}
  - Columns of V are orthonormal eigenvectors of $A^{T}A$
  - S is a diagonal matrix containing square roots of eigenvalues from U or V in descending order.

Singular Value Decomposition (SVD)

- SVD can also be represened as: $A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$
- Let A be a n×d matrix with singular vectors $v_1, v_2, \ldots, v_r$ and corresponding singular values $\sigma_1, \sigma_2, \ldots, \sigma_r$.
- Then $u_i = (1/\sigma_i)*Av_i$, for i = 1, 2, . . . , r, are left singular vectors. Where Rank(A) = r.
- A can be decomposed into a sum of rank one matrices as:
- $A = U\Sigma V^T$

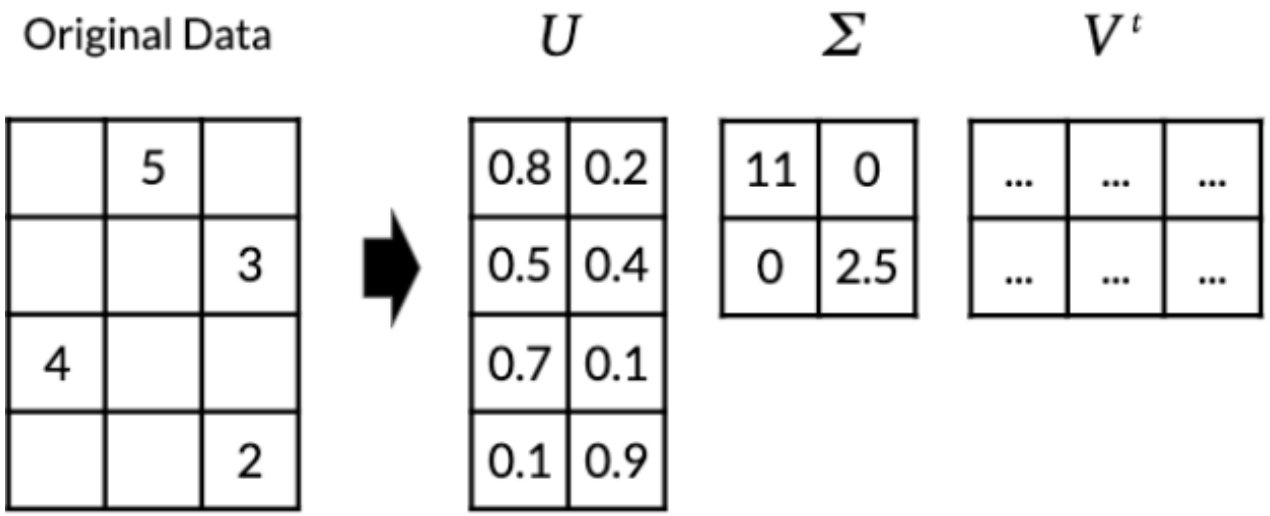
OR

$$A = \sum_{i=1}^{r} \sigma_i \mathbf{u_i} \mathbf{v_i}^T$$

$U \in R^{m \times r}$ has orthonormal columns; $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_r)$, where $\sigma_1 \geq \cdots \geq \sigma_r > 0$; $V \in R^{n \times r}$ has orthonormal columns.

Singular Value Decomposition (SVD)

```
from sklearn.decomposition  import TruncatedSVD
import numpy as np

np.random.seed(0)
X = np.random.rand(100,  100)

# four components
svd = TruncatedSVD(n_components=4,  n_iter=10, random_state=5)

U = svd.fit_transform(X)
Sigma  = np.diag(svd.singular_values_)
V = svd.components_


# in case we want to do the  multiplication

# U x Sigma
U_x_Sigma = np.dot(U, Sigma)

# (U x Sigma)  x V
U_Sigma_V  = np.dot(U_x_Sigma  , V)
```



Application of SVD: Compression

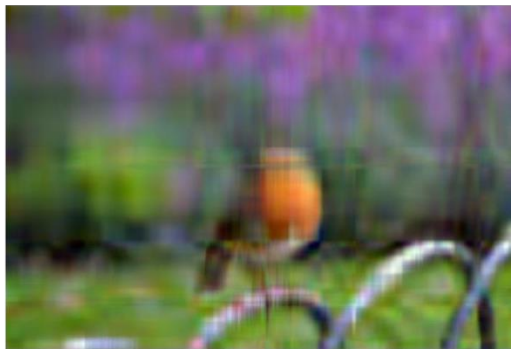
- For many images, k much smaller than n can be used to reconstruct the image provided that a very low-resolution version of the image is sufficient.
- Thus, one could use SVD as a compression method.

Singular Values: 478

Singular Values: 60

Singular Values: 25

Singular Values: 10



https://www.cs.cmu.edu/~venkatg/teaching/CStheory-infoage/book-chapter-4.pdf
https://github.com/kounelisagis/Image-Compression---SVD

Decision Tree

Decision Trees

- Decision Trees can perform both classification and regression* tasks, and multi-output tasks.

- Decision Trees are used
  - for inductive inference.
  - for approximating discrete-valued target functions.

- Decision Trees are
  - robust to noisy data and can learn disjunctive expressions.
  - capable of fitting complex datasets.
  - capable of leading to a decision via if/else questions.

* estimate the relationships between a dependent variable ('outcome variable') and one or more independent variables.

Decision Trees

- Decision tree searches a completely expressive hypothesis.
  1. Represent a disjunction of conjunctions of constraints on attribute values of instances.
  2. Each path from tree root to a leaf corresponds to a conjunction of attribute tests
  3. Tree itself is a disjunction of these conjunctions.

- Solves classify problems.
  - Learned function is represented by a decision tree.

When to Consider Decision Trees?

- Instances are represented by attribute-value pairs.
- Target function has discrete output values (two or more) (e.g., Boolean classification).
- Disjunctive descriptions may be required (represent disjunctive expressions).
- Training data may contain errors (robust to errors) or missing values (can handle).

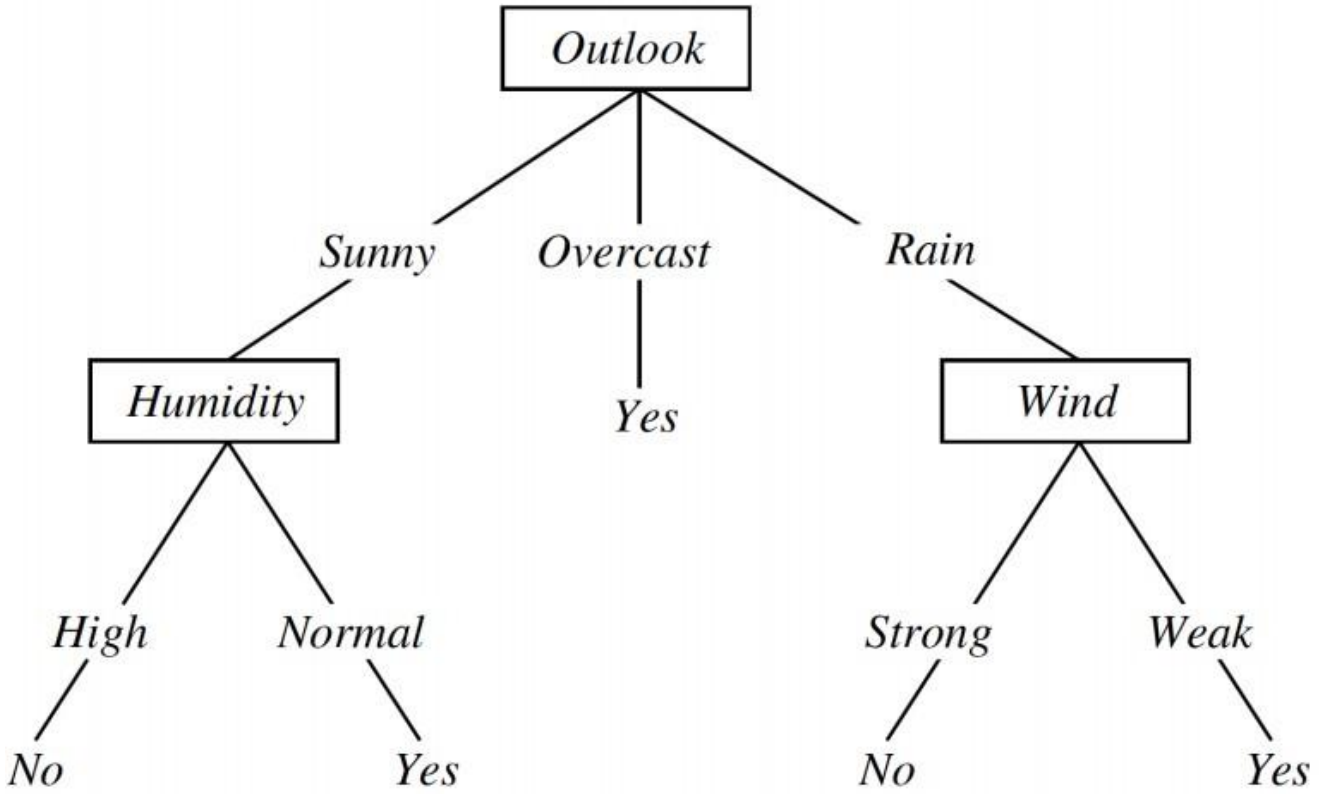
Decision Trees

- Decision trees classify instances by sorting from root to leaf node.
- Each node of tree specifies a test of some attribute of instance.
- Each branch descending from a node corresponds to one of possible values for attribute.
- Each leaf node assigns a classification.

Example

(Outlook = Sunny ∧ Humidity = Normal) v (Outlook = Overcast) v (Outlook = Rain ∧ Wind = Weak)

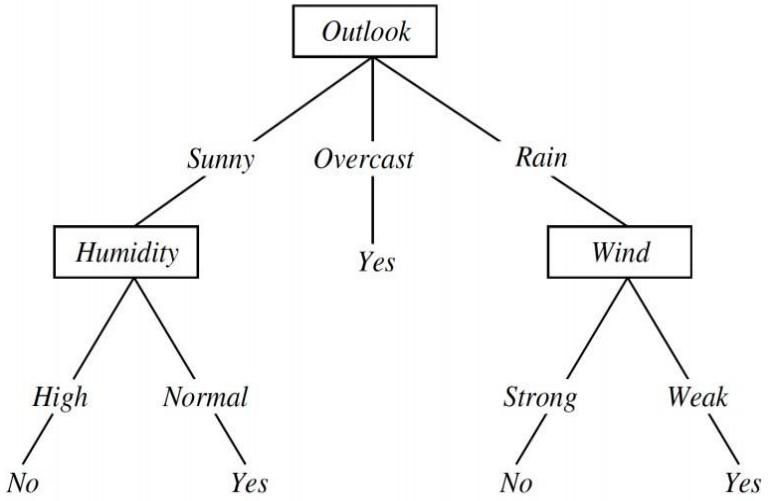
(Outlook=Sunny, Humidity=High, Wind=Strong) ==> negative instance.

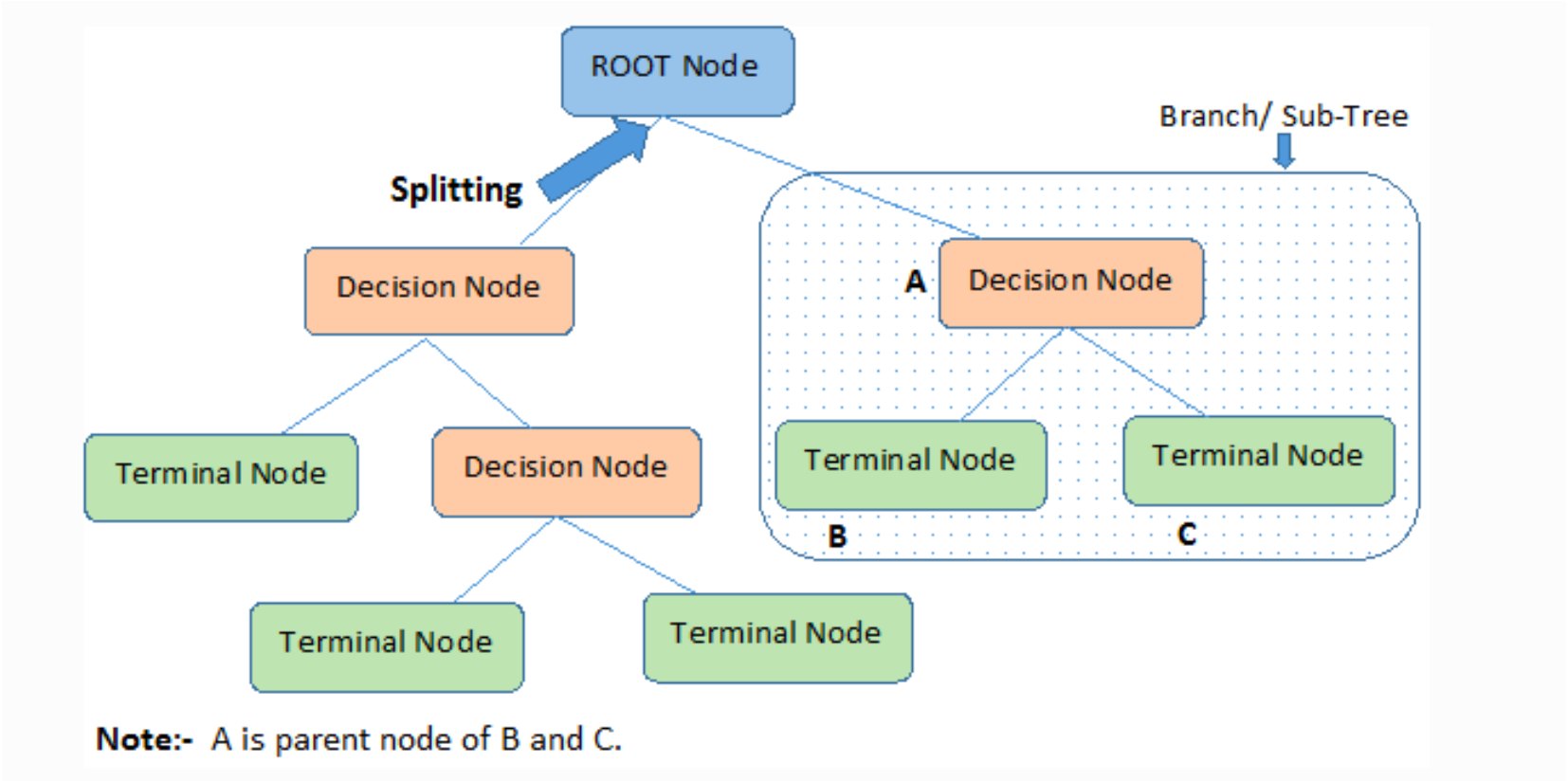


Example

(Outlook = Sunny ∧ Humidity = Normal) v (Outlook = Overcast) v (Outlook = Rain ∧ Wind = Weak)

(Outlook=Sunny, Humidity=High, Wind=Strong) ==> negative instance.



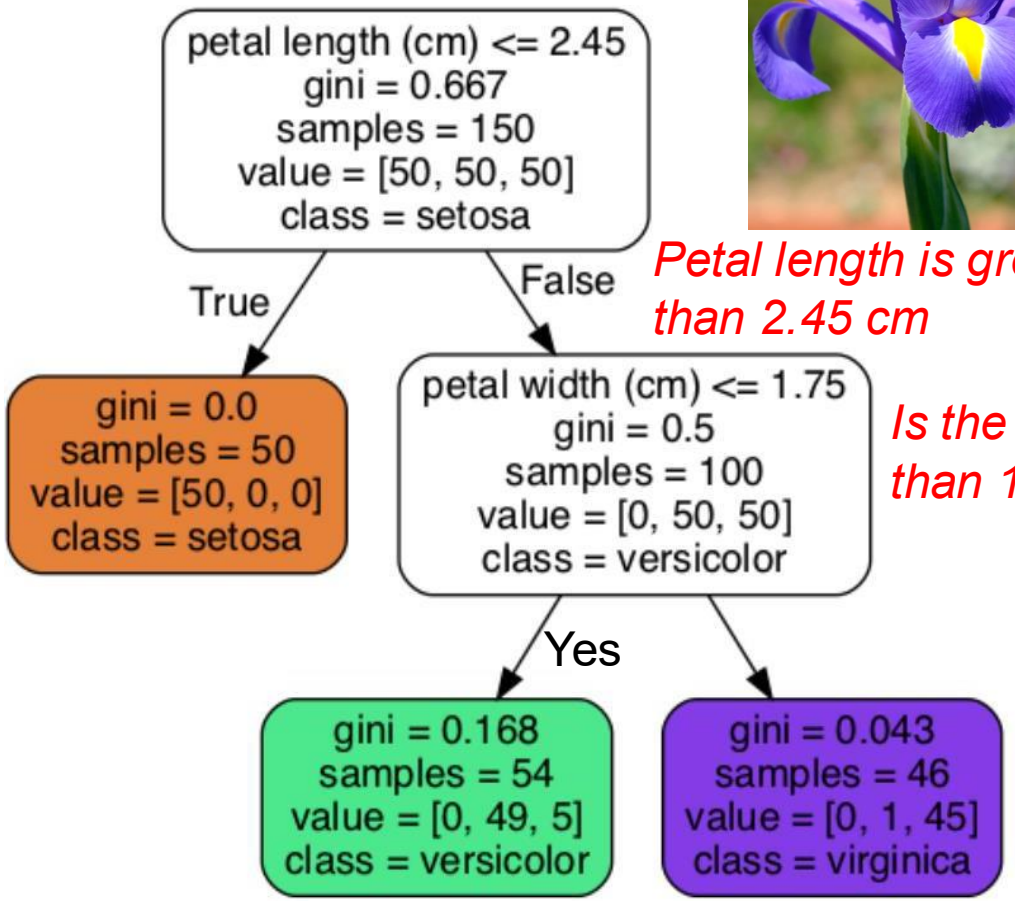


Source: Kdnuggets

Training and Visualizing a Decision Tree

```
from sklearn.datasets  import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz

iris = load_iris()
X = iris.data[:, 2:] # petal length and width
y = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
export_graphviz(
        tree_clf,
        out_file=image_path("iris_tree.dot"),
        feature_names=iris.feature_names[2:],
        class_names=iris.target_names,
        rounded=True,
        filled=True
)
```

Q. You want to find classify an iris flower



Petal length is greater than 2.45 cm

Is the petal width smaller than 1.75 cm?

Require very little data preparation.

Don’t require feature scaling.

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron. 2019.

Learning Decision Trees

- Given a set of training data in form of examples (e.g., rating), construct questions that you can ask.
- Learning is about searching for the "best" tree to describe data.

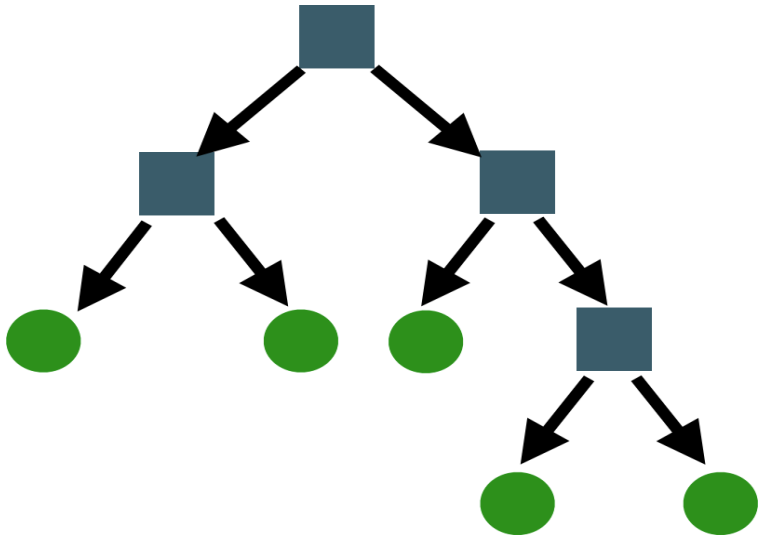
- We could enumerate all possible trees, and evaluate each
  - How many trees are there given 3 features? too many!
  - It is computational infeasible to consider all trees, so decision trees must be built greedily by asking important question.

- Each node represents a question that split data
  - Learning a decision tree amounts to choosing what internal nodes should be.

Decision Tree Training Algorithm

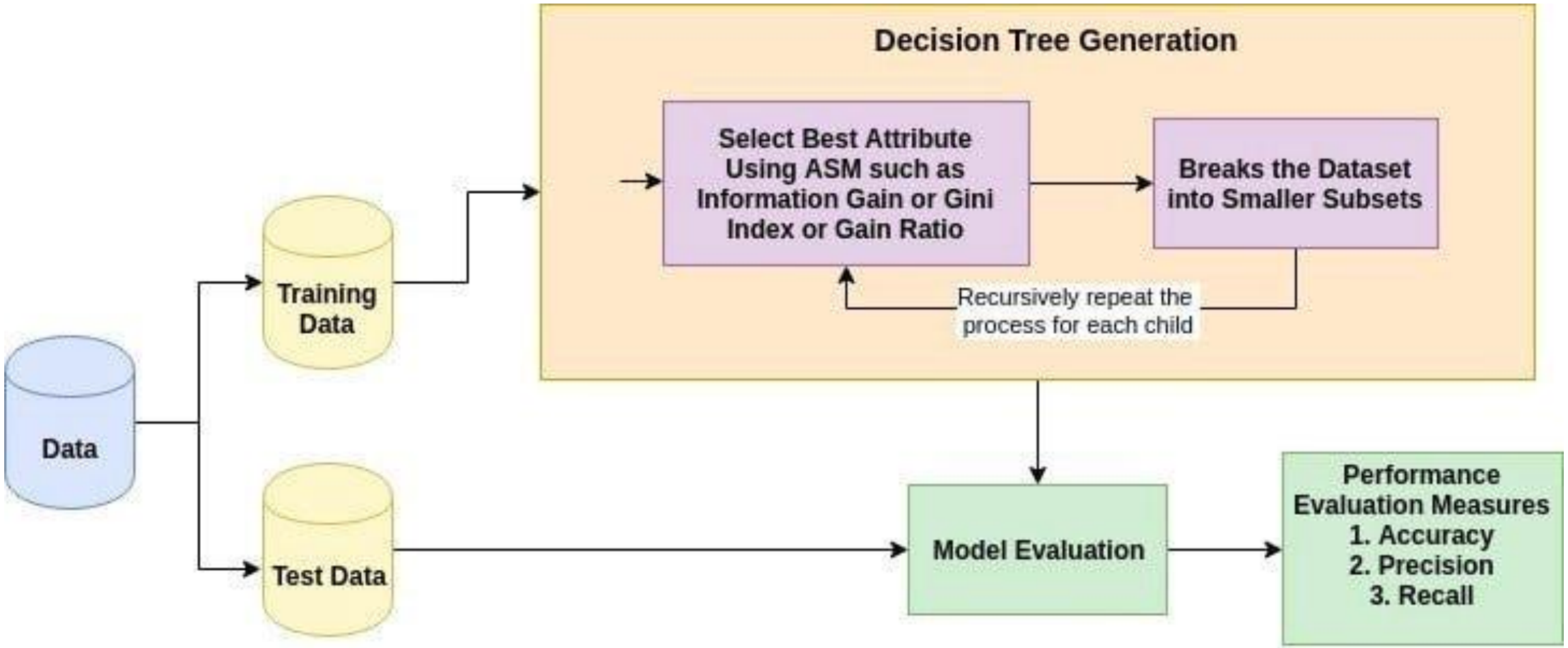
- Given a set of labeled training instances:

1. If all the training instances have same class, create a leaf with that class label and exit.
2. Else Pick the best, test to split the data on.
3. Split the training set according to the value of the outcome of the test.
4. Recursively repeat step 1-3 on each subset of the training data.



Decision Trees

ASM: Attribute Selection Measure



Source: DataCamp

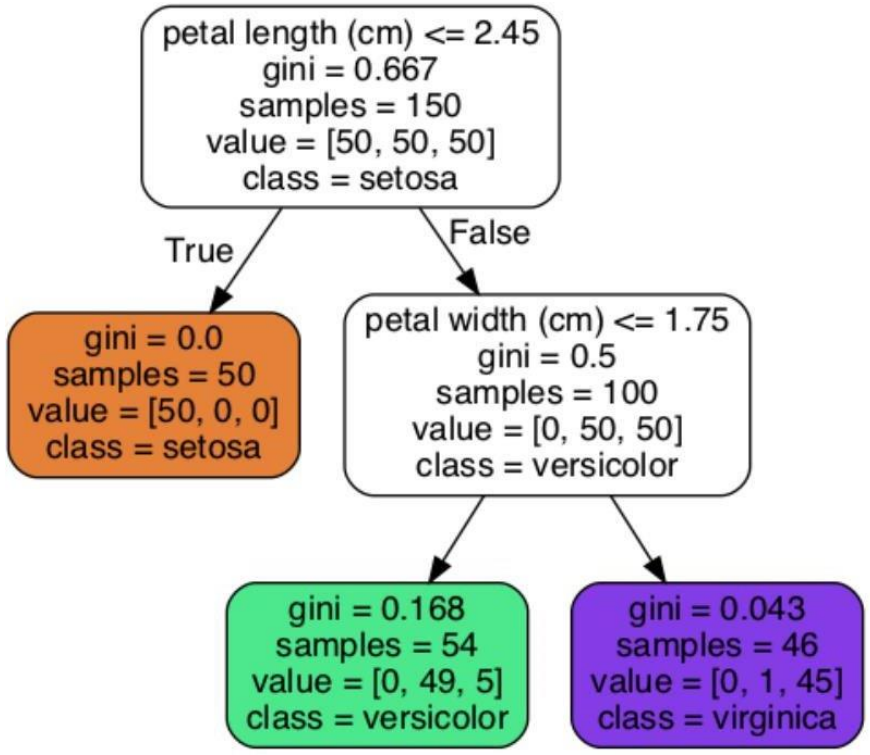
Impurity: Gini and entropy

- Gini Index is to divide decision tree.
- Gini Index (Impurity) calculates the likelihood that somehow a randomly picked instance would be erroneously cataloged.
- Works on categorical variables & does not work with continuous targets.
- Results in “success” /1 or “failure”/0.
- Gini used to find which attribute holds the maximum information about a class.
  - Enhances accuracy and reliability.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$$

$p_{i,k}$: ratio of instances of class k in node i

Range: 0 (perfectly pure) to 0.5 (maximum impurity, for a binary classification).



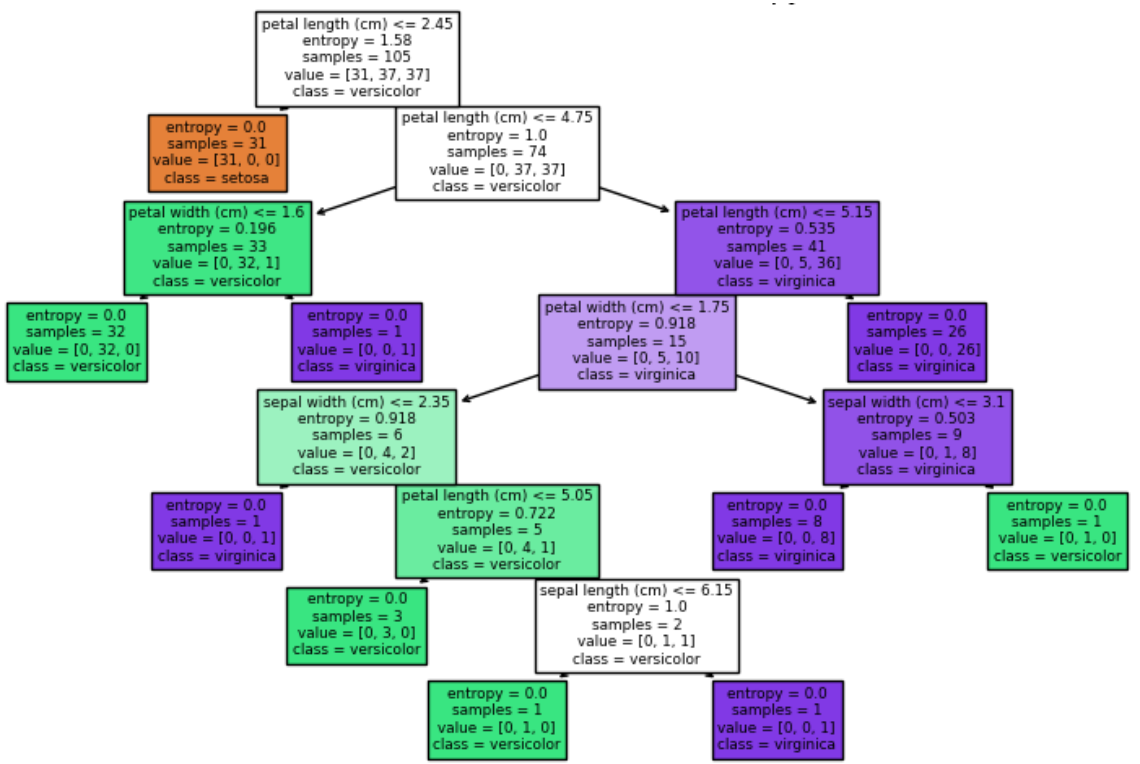
Gini impurity at the root node can be more than 0.5.

- It means the root node is evenly split among the different classes (see the value = [50, 50, 50]), making it harder to classify the data points correctly without further splitting that's what we did.

Impurity: Gini and Entropy

- Indicates disorder.
- Used as an impurity measure.
  - a set's entropy is zero when it contains instances of only one class (see side pic=>).
- Reduction of entropy is called an information gain.
  - The difference between a parent node's entropy and the weighted sum of its child node entropies.

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2 \left(p_{i,k}\right)$$



Range: 0 (perfectly pure) to $\log_2(n)$ (maximum impurity, n= # of class)

Ensembles: Random Forests

Ensemble Learning

- Noise, variance, and bias are important factors in ML based prediction.
- Ensemble can reduce variance, and bias.
  - is a collection of predictors to give a final prediction.
  - Further classified into Bagging and Boosting.

- Aggregating several predictors is often better than using just one predictor
  - Several “weak” learners → “strong” learner

- Differences between ensemble learning approaches:
  - Which one to combine?
  - How to combine?

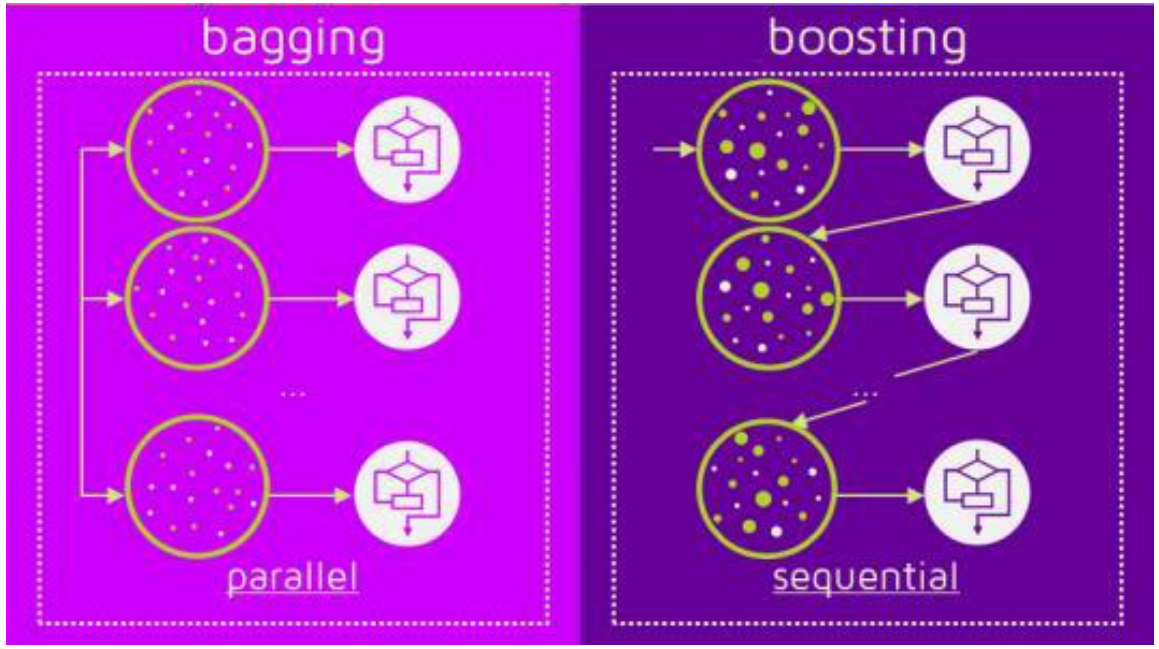
Bagging and Boosting

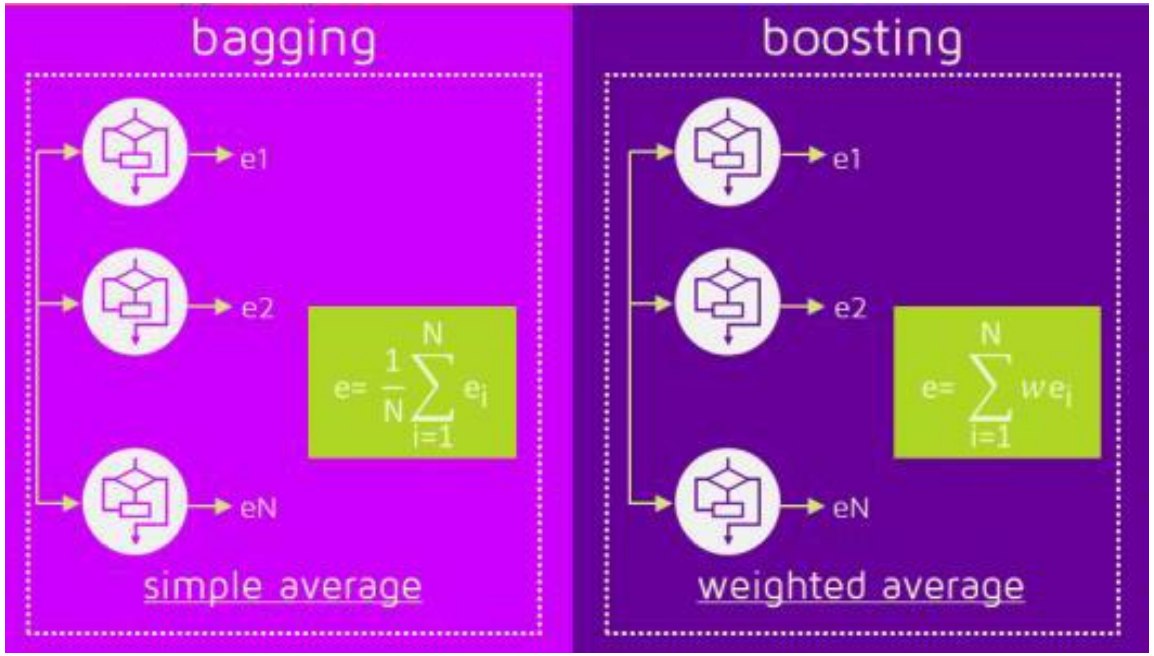
- Bagging builds many independent predictors and combine them using some model averaging techniques. (e.g. weighted average, majority vote)
  - Random Forest
    - Handle overfitting

- In Boosting, predictors are not made independently, but sequentially.
  - Gradient Boosting
    - Can overfit

- Ensemble methods work best when the predictors are as independent from each other.

Bagging and Boosting





- Train classifiers using different algorithms.
- Will make different types of errors, improving ensemble's accuracy.

Img src: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Random Forests

- Random Forest is an ensemble of Decision Trees.
- Trained via bagging method.
- Random Forest Classifier more convenient and optimized for Decision Trees.
- Random Forest classifier with 500 trees:

Lower number of leaf nodes restricts the depth of each tree -> reducing overfitting

```
from sklearn.ensemble  import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500,  max_leaf_nodes=16,  n_jobs=-1)
rnd_clf.fit(X_train,  y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Random Forests

- RandomForestClassifier has all hyperparameters of a DecisionTreeClassifier (mostly).
  - Can control growth of trees.
  - Hyperparameters of a BaggingClassifier to control the ensemble.

- Random Forest algorithm introduces extra randomness when growing trees
  - Searches for best feature among a random subset of features.
  - Results in greater tree diversity –> overall better model.

```
bag_clf = BaggingClassifier(
DecisionTreeClassifier(splitter="random", max_leaf_nodes=16), n_estimators=500,
max_samples=1.0, bootstrap=True, n_jobs=-1)
```

Each tree is trained using 100% of the available training data

Each tree is trained on a randomly sampled subset

Random Forests

- Tree growth in a Random Forest: at each node only a random subset of features is considered for splitting.
- Possible to make trees more random by using random thresholds for each feature.
- Easy to measure relative importance of each feature (weighted average).
- Ex: Trains a RandomForestClassifier on the iris dataset and outputs each feature's importance.

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

rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
print(name, score)
```

```
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Most important features are petal length (44%) and petal width (42%)