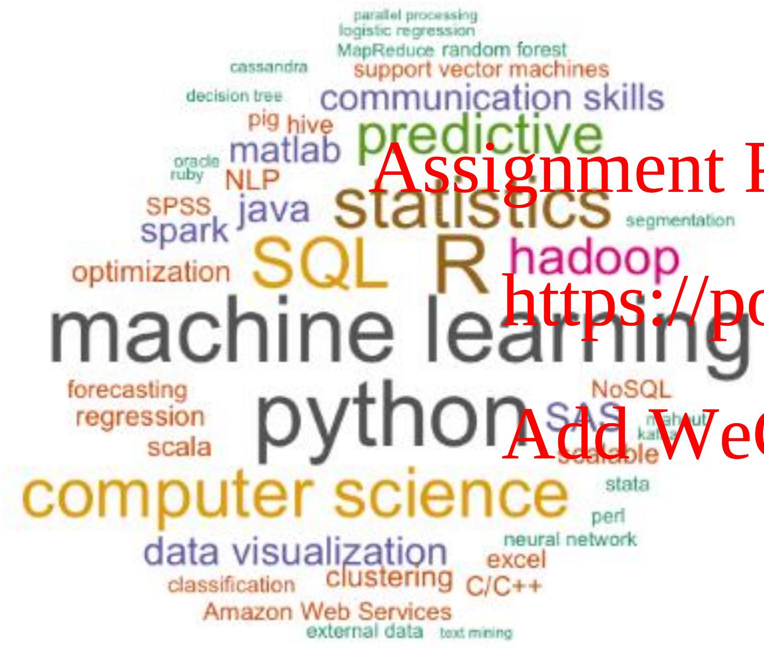
Classification

(Concepts – Part 1)

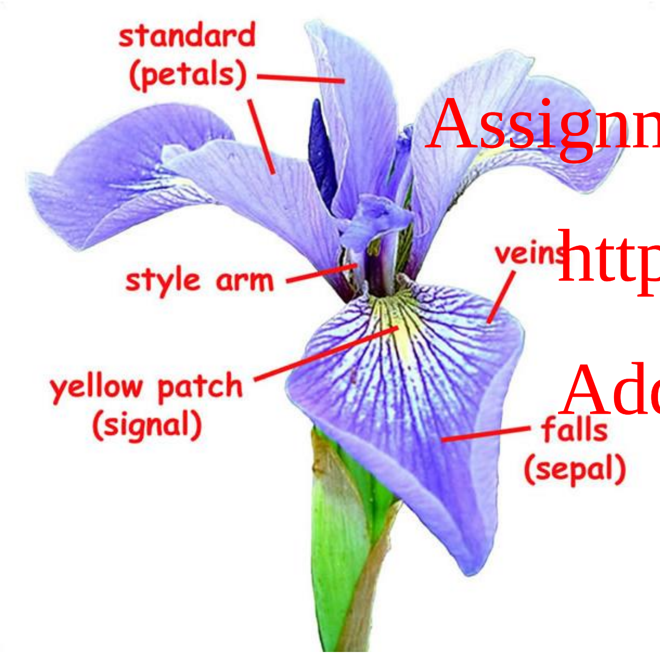


Contents

- The Iris Dataset
- Training & Testing Data Partitioning
- Evaluation by Accuracy Score
- k-Nearest Neighbours Classifier
- Decision Tree Classifier

The Iris Dataset

The Iris dataset has a long, rich history in machine learning and statistics



- Each row describes one iris in terms of the length and width of that flower's sepals and petals
  - Those are the big flowery parts and little flowery parts
- There are four measurements per iris
  - Each of the measurements is a length of one aspect of that iris
- The final column, the classification target, is the particular species - one of three - of that iris: setosa, versicolor, or virginica

The prediction is to choose 1 out of 3 species meaning that the target variable is categorical with 3 possible values

IRIS SETOSA

IRIS VERSICOLOR

IRIS VIRGINICA

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0
10	5.4	3.7	1.5	0.2	0
11	4.8	3.4	1.6	0.2	0
12	4.8	3.0	1.4	0.1	0
13	4.3	3.0	1.1	0.1	0
14	5.8	4.0	1.2	0.2	0

Observations & features

- All measurements are recorded as float in cm
- The target can be one of three classes that are encoded as follows:
  - 0 = setosa
  - 1 = versicolor
  - 2 = virginica
- Altogether 150 observations, 50 for each class

Python: Basic Settings

import the plotting module and binds it to the name "plt"
display all warnings

```
import matplotlib.pyplot as plt
import warnings
```


display the output of plotting commands inline
use the "retina" display mode, i.e. to render higher resolution images


```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```


customize the display style
set the dots per inch (dpi) from the default 100 to 300
suppress warnings related to future versions

```
plt.style.use('seaborn')
plt.rcParams['figure.dpi'] = 300
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Python: Understanding the Iris Dataset (1)

```
# import the relevant modules
import pandas as pd
import seaborn as sns
from sklearn import datasets
```



```
# load the Iris dataset
# https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html
iris = datasets.load_iris()
```



```
# create a dataframe from the feature values and names
# set the dataframe target column with the target values of the dataset
data = pd.DataFrame(iris.data, columns=iris.feature_names)
data['target'] = iris.target
data['target'] = data['target'].astype('category').cat.rename_categories(iris.target_names)
```



```
# display the data shown earlier
data.head(15)
```

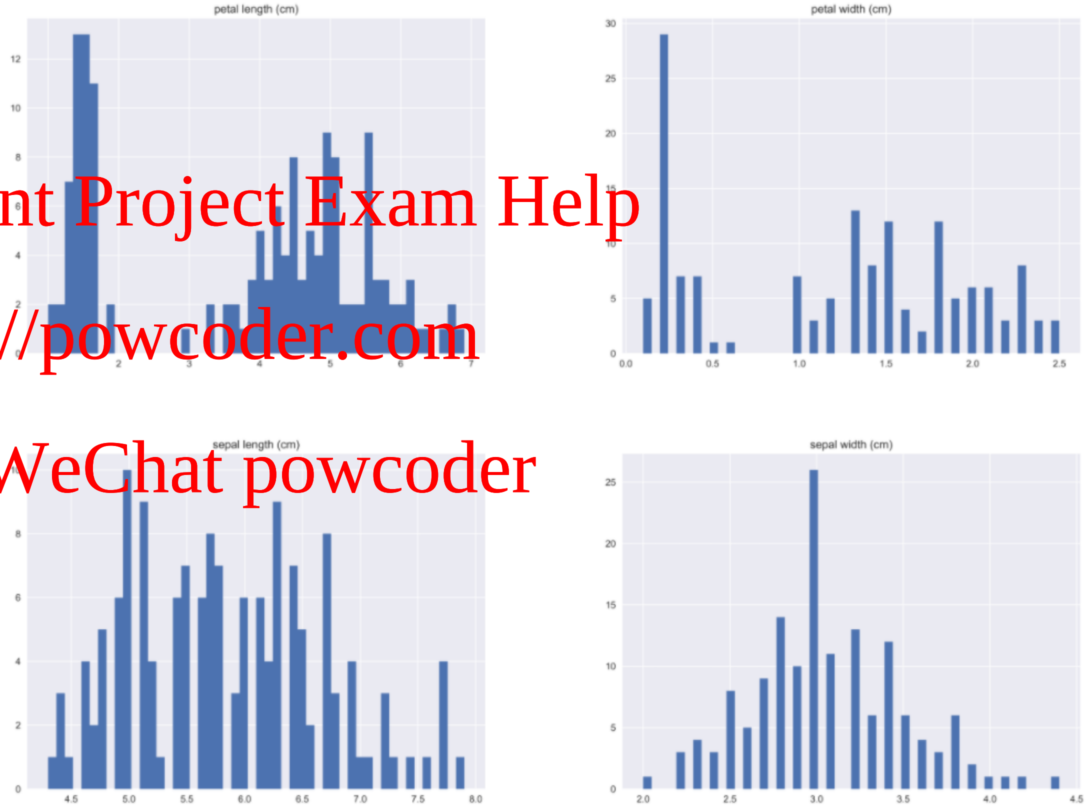
Python: Understanding the Iris Dataset (2)

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa
10	5.4	3.7	1.5	0.2	setosa
11	4.8	3.4	1.6	0.2	setosa
12	4.8	3.0	1.4	0.1	setosa
13	4.3	3.0	1.1	0.1	setosa
14	5.8	4.0	1.2	0.2	setosa

Python: Understanding the Iris Dataset (3)

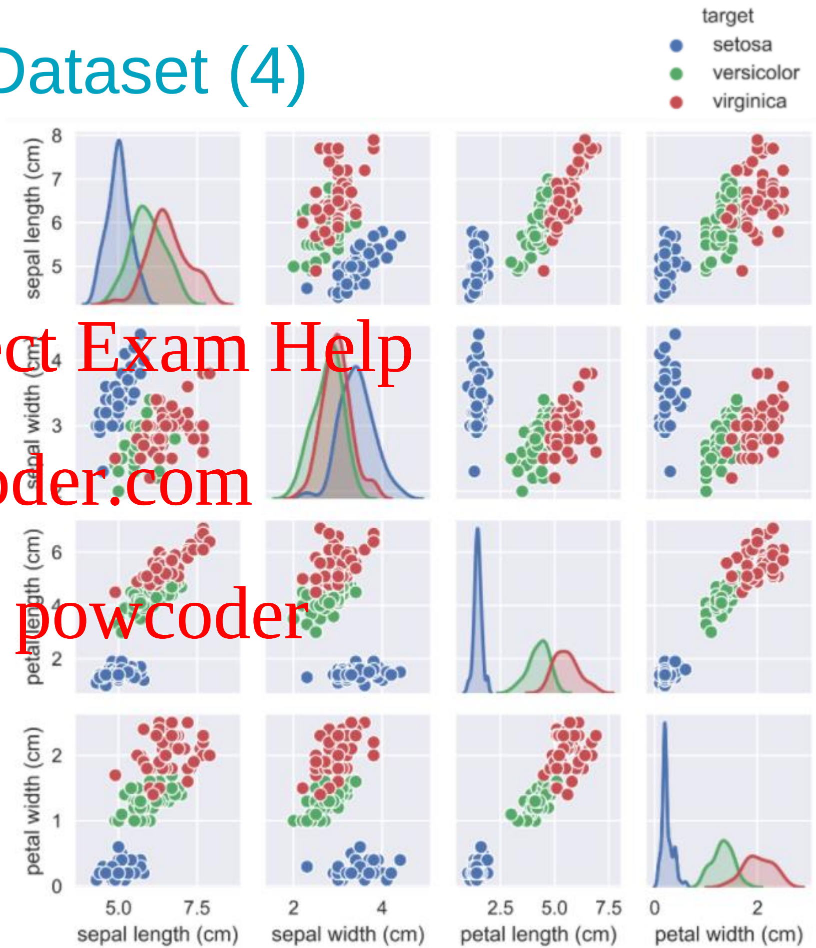
```
# plot a histogram for each feature
# hist() plots the whole dataset
```

```
data.hist(bins=50, figsize=(20,15))
```



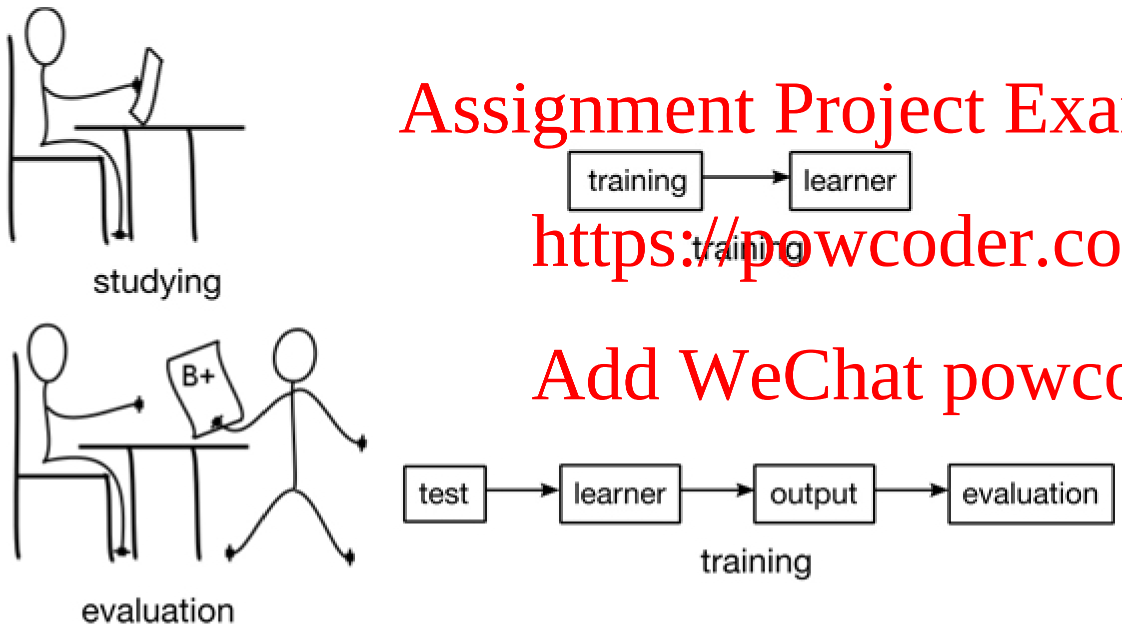
Python: Understanding the Iris Dataset (4)

```
# draw a pairs plot to show
# (1) the distribution of single variables
# (2) relationships between two variables
# https://seaborn.pydata.org/generated/seaborn.pairplot.html
# specify the feature to color using "hue"
sns.pairplot(data, hue='target', size=1.5)
```



Training & Testing Data Partitioning

Teaching to the test is usually regarded as a bad thing



- The goal of machine learning is to do well on unseen observations
- Performance on unseen observations is called generalization
- If an ML model is tested on data that have already been seen, an overinflated estimate of its abilities on novel data will result, i.e. overfitting

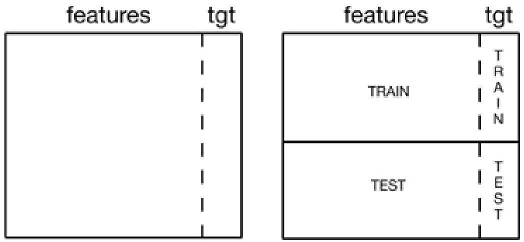
Python: Partitioning the Dataset

```
# load relevant modules
from sklearn.model_selection import train_test_split
```

```
# partition dataset into training data and testing data
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
# (1) separate the features from the target by dropping and projecting a column
# (2) specify the proportion of the testing dataset (30%)
# (3) control the shuffling using a seed (i.e. 0) to ensure reproducible output
X_train, X_test, y_train, y_test = train_test_split(
    data.drop('target', axis = 1), data['target'], test_size = 0.3, random_state = 0)
```

```
# display the number of rows and columns
print("Training data shape: ", X_train.shape)
print("Testing data shape: ", X_test.shape)
```

```
Training data shape:  (105, 4)
Testing data shape:  (45, 4)
```



Evaluation by Accuracy Score

Python: Evaluation by Accuracy Score

```
# load relevant modules
import numpy as np
from sklearn.metrics import accuracy_score
```



```
# for illustration purpose only
# build an array containing the correct answers and another array for the ML model results
y_t = np.array([True, True, False, True])
ys = np.array([True, True, True, True])
```



```
# calculate the accuracy score using scikit-learn built-in function
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
# it is the number of correct answers over the total number of answers
print("sklearn accuracy:", accuracy_score(y_t, ys))
```

```
sklearn accuracy: 0.75
```

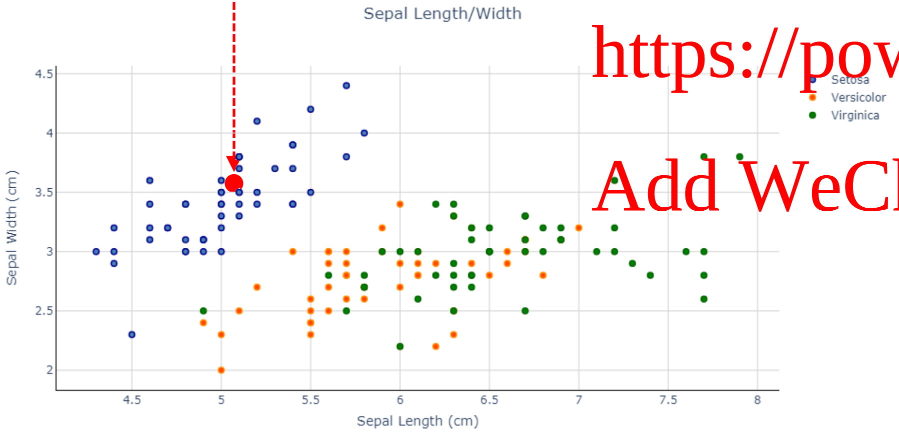
$$Accuracy = \frac{Number\ of\ Corrrect\ Answers}{Number\ of\ Answers}$$

k-Nearest Neighbours (k-NN) Classifier

k-NN prediction is based on the nearest neighbours

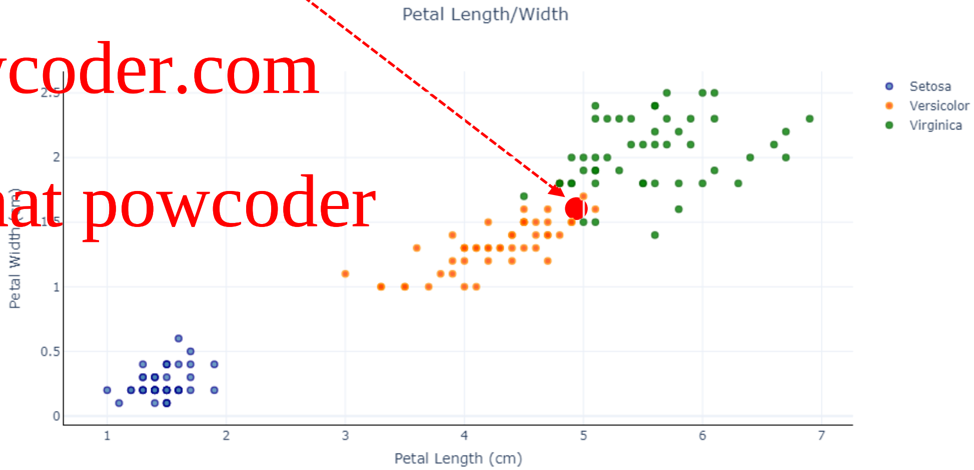
The 10 nearest neighbours are all setosa

prediction is obvious



The nearest neighbours are not of the same type

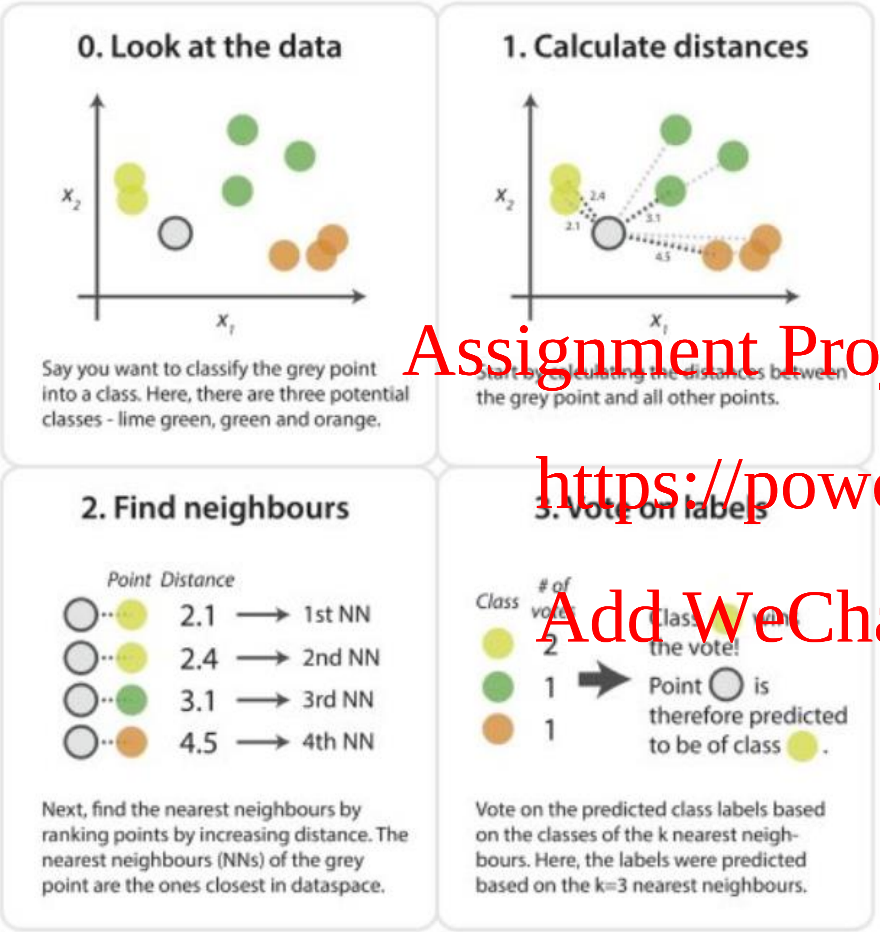
prediction is not so obvious



Prediction over a labelled dataset can be made by taking the classification of the nearest k neighbours

- Key ideas behind the nearest neighbours algorithms
  - Find a way to describe the similarity of two different observations
  - When a prediction on a new observation is needed, simply take the value from the most similar know observation

- The ideas can be generalised by taking values from several neighbours and combine their values to generate the prediction
  - Common numbers for neighbours are 1, 3, 10, 20, …
  - The optimal number is typically obtained through Grid Search



Distances to neighbours & how to combine them?

- One way to measure similarity is the distance between two observations in the multi-dimensional feature space
- neighbors.DistanceMetric in the scikit-learn module has defined some 20 metrics
- The values from the nearest neighbours can be combined by taking the most frequent value as the prediction

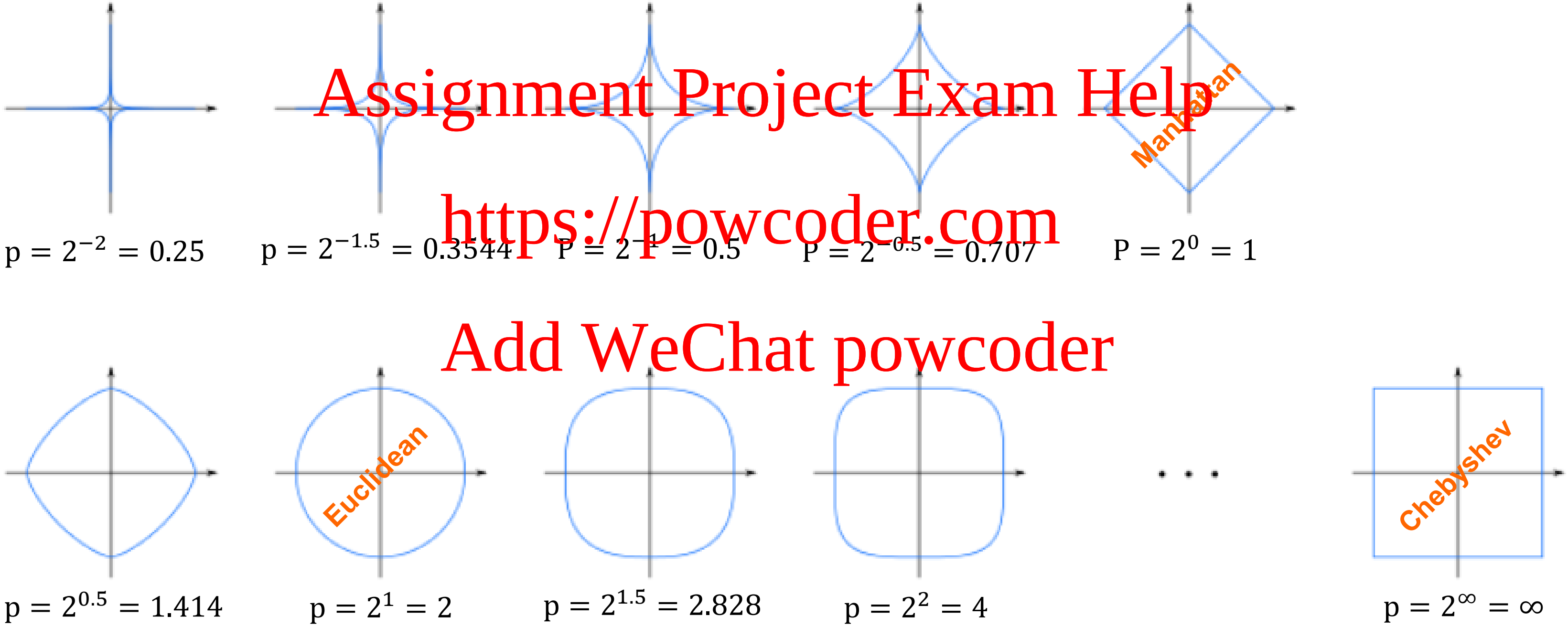
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html

Minkowski distance measures that are positive, symmetric and satisfying the triangle inequality are distance metrics

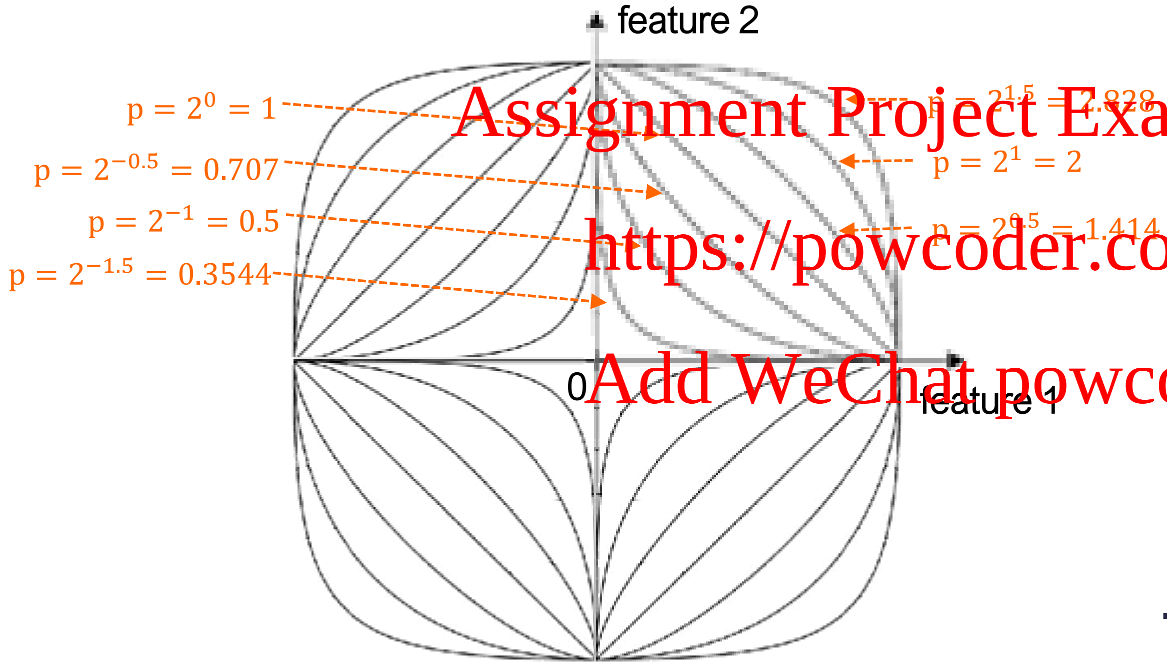
- Euclidean (numeric features) $D(v, v') = \sqrt{\sum_{i=1}^{d} |v_i - v'_i|^2}$
  - symmetric, spherical, treats all dimensions equally
  - sensitive to extreme differences in a single feature
- Hamming (categorical features) $D(v, v') = \sum_{i=1}^{d} 1_{v_i \neq v'_i}$
  - counting - distance is 1 if two categorical feature values are not the same; otherwise, 0
- Minkowski (numeric features, L_p norm distance) $D(v, v') = \sqrt[p]{\sum_{i=1}^{d} |v_i - v'_i|^p}$
  - A metric satisfies
    - Positivity: $D(v, v') > 0$ if $i \neq j$ and $D(v, v) = 0$
    - Symmetry: $D(v, v') = D(v', v)$
    - Triangle Inequality: $D(v, v') \leq D(v, p) + D(p, v')$

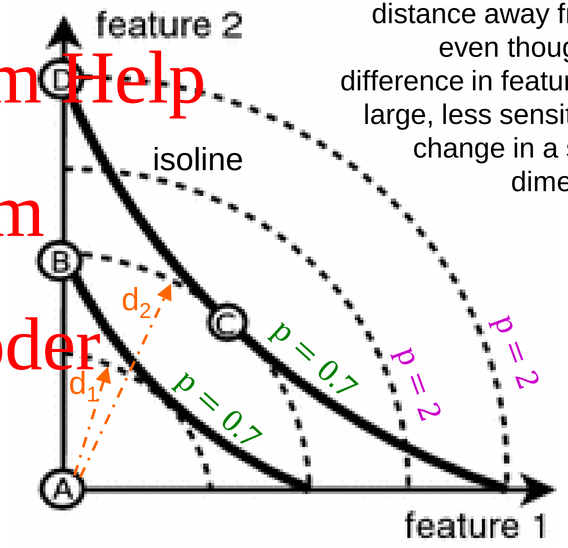
Recall the use of L_1 norm and L_2 norm distance metrics in Scaling Feature Vector to Unit Vector.

Changing the order parameter generates different distance metrics including other commonly used metrics



The choice of p determines the impact of value change in different feature dimensions to the distance metric





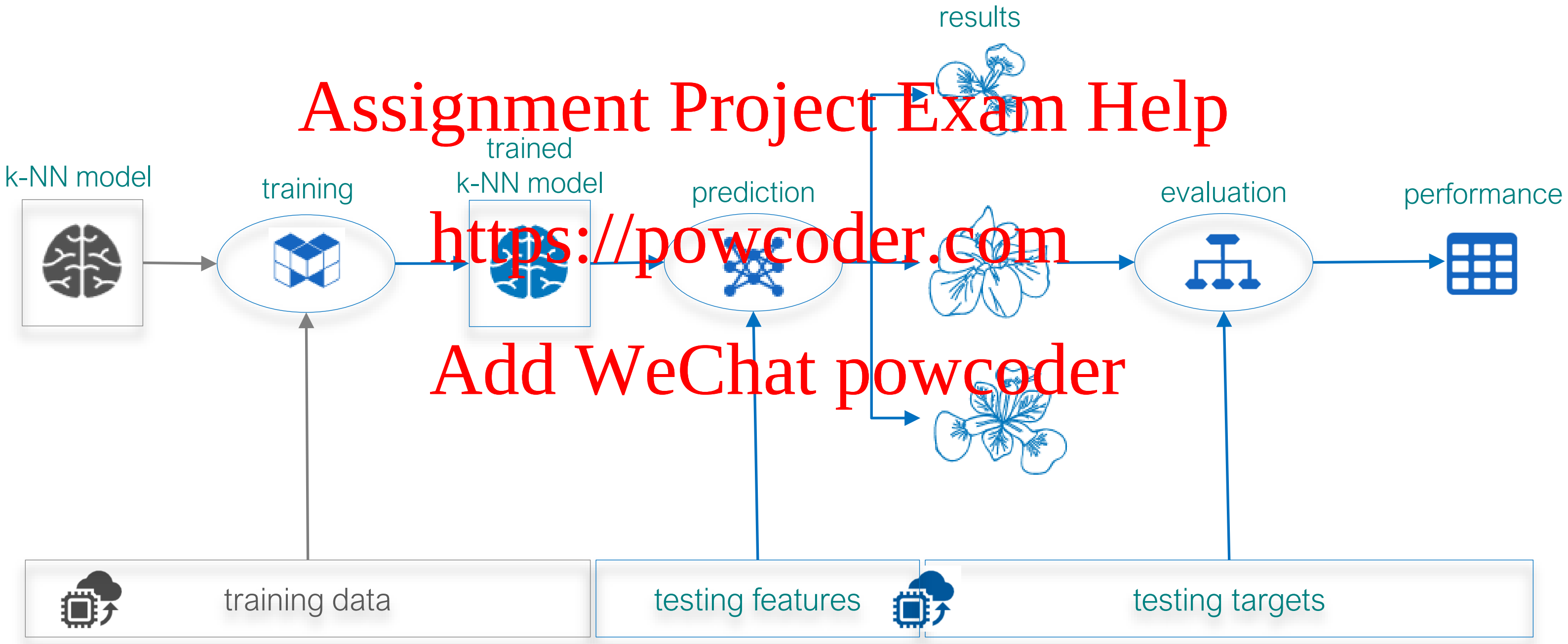
$p = 0.7$

D & C are the same distance away from A even though the difference in feature 2 is large, less sensitive to change in a single dimension

$p = 2$

B & C are the same distance away from A, D is farther away, sensitive to change in a single dimension

Like all other ML models, a k-NN model needs to be trained, tested, and evaluated against a hold-out dataset



Python: Fitting a k-NN model and Making Prediction with it

load relevant modules

```
from sklearn.neighbors import KNeighborsClassifier
```

instantiate a 3-NN classifier
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
instantiate a 3-NN classifier

```
knn = KNeighborsClassifier(n_neighbors=3, metric='minkowski')
```

fit/train the classifier to the training dataset

```
model = knn.fit(X_train, y_train)
```

predict the targets for the test features

```
test_t = model.predict(X_test)
```

calculate the accuracy score for the predicted targets using the known targets

```
print("3NN accuracy:", accuracy_score(y_test, test_t))
```

```
3NN accuracy:  0.9777777777777777
```

k-NN is non-parametric and has hyperparameters

- k-NN is a non-parametric model
  - Unlike many other models, k-NN outputs (the predictions) cannot be computed from an input example and the values of a small, fixed set of data
  - All of the training data is required to figure out the output value
  - Throwing out just one of the training data, which might be the nearest neighbor of a new test data, will affect the output
- The 3 in the k-NN model is not something that is adjusted by the k-NN training process but a hyperparameter
  - Adjusting a hyperparameter involves conceptually, and literally, working outside the learning box of model training

k-NN Model is simple to understand and implement

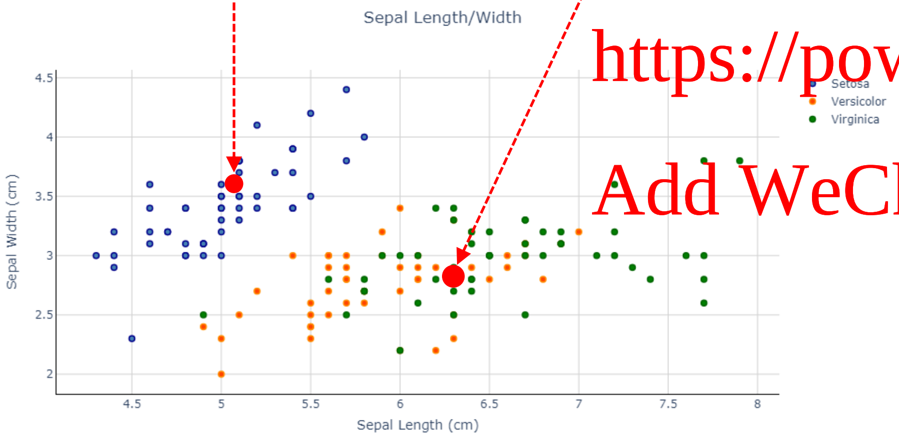
	Property	Description
1	Feature Data Types	Scale data. Variable encoding is therefore necessary for categorical data.
2	Target Data Types	Nominal data including binary. Representing class membership.
3	Key Principles	An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors
4	Hyperparameters	Number of neighbours (k). The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise, but make boundaries between classes less distinct (high bias). In binary classification, k should be an odd number to avoid tied votes. Use k-fold (this k is the not same k in k-NN) cross validation to select k. Distance metric (Python default: Minkowski).
5	Data Assumptions	Non-parametric – no assumption about data distribution. All data are used.
6	Performance	Instance-based/Lazy learning, where the prediction is only approximated locally and all computation is deferred until prediction evaluation. No model per se is built. No training is therefore required. Model building is therefore quick but scoring becomes slower.
7	Accuracy	Can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance. Feature scaling is recommended. Imbalanced data should be avoided.
8	Explainability	Poor.

Decision Tree Classifiers

Decision tree partitions the sample space to form regions that can be captured as decision rules

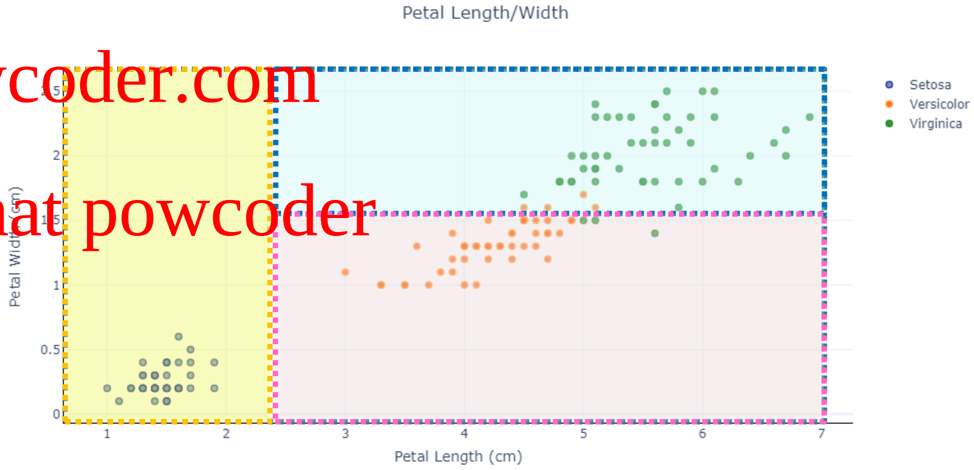
prediction is obvious

prediction is not so obvious

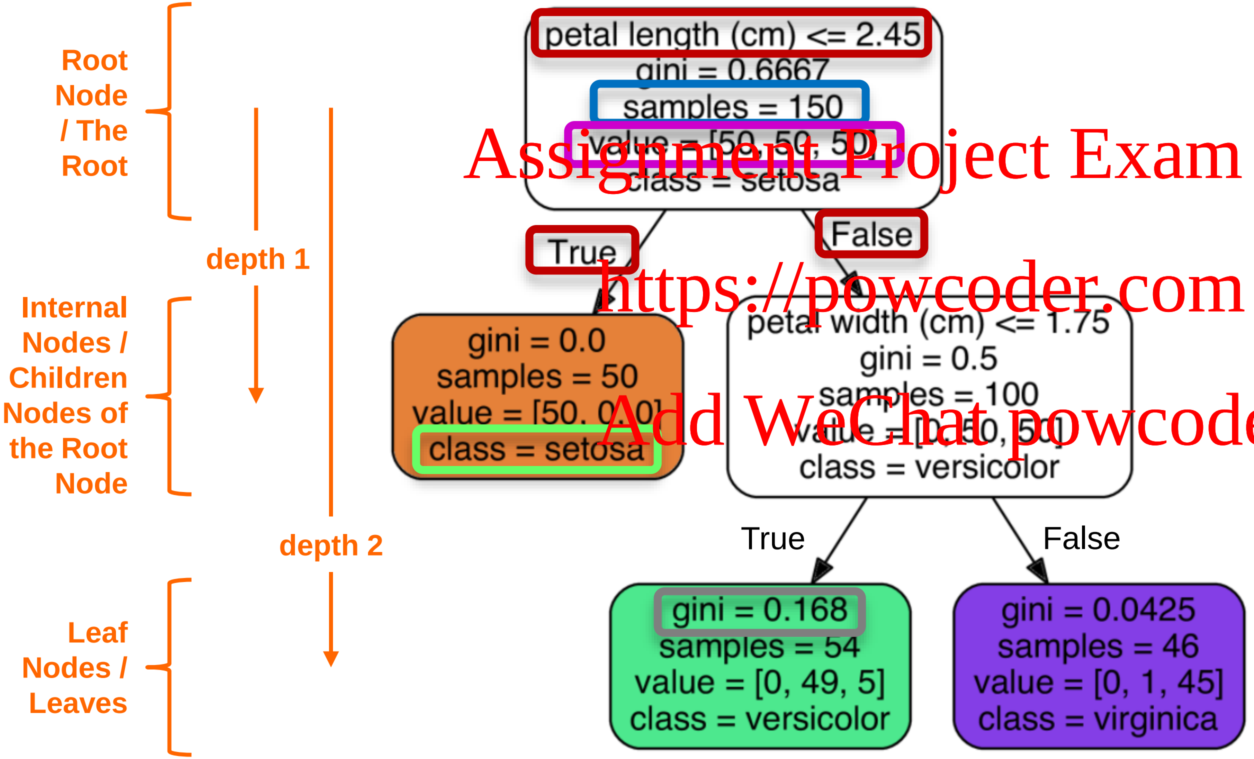


sample space

portioning sample space facilitates building prediction rules



Nodes in a decision tree make splitting decision on which sub-tree to explore next until reaching a leaf node



Decision with splitting condition

Number of data over which the decision is applied

Number of data in each category over which the decision is applied

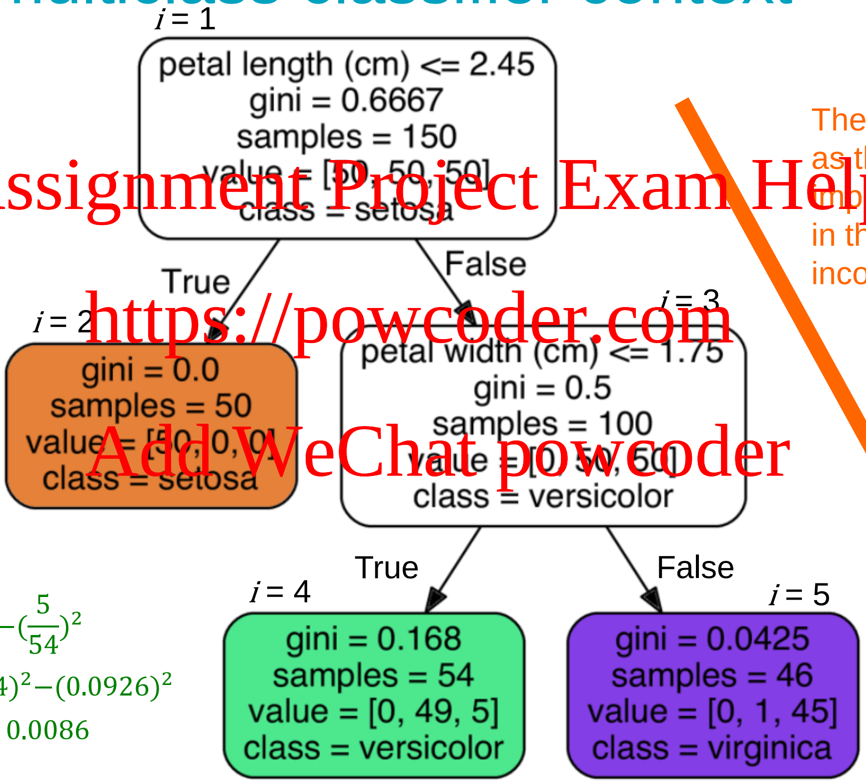
Predicted category

Gini impurity score

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

where
G_i is the Gini impurity score for node i
n is the number of categories
$p_{i,k}$ is the ratio of k instances among all the data covered at node i

Gini impurity is a measure of misclassification that is applicable in a multiclass classifier context



The Gini impurity score decreases as the depth increases implying the gradual reduction in the likelihood of incorrect classification.

Ideally, the Gini impurity score for the leaf nodes should be zero.

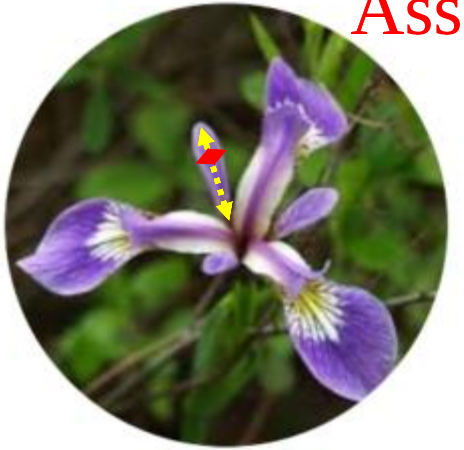
$$\begin{aligned} G_2 &= 1 - (\frac{50}{50})^2 - (\frac{0}{50})^2 - (\frac{0}{50})^2 \\ &= 1 - (1)^2 - (0)^2 - (0)^2 \\ &= 1 - 1 - 0. - 0 \\ &= 0.0 \end{aligned}$$

$$\begin{aligned} G_4 &= 1 - (\frac{0}{54})^2 - (\frac{49}{54})^2 - (\frac{5}{54})^2 \\ &= 1 - (0)^2 - (0.9074)^2 - (0.0926)^2 \\ &= 1 - 0 - 0.8234 - 0.0086 \\ &= 0.1680 \end{aligned}$$

$$\begin{aligned} G_5 &= 1 - (\frac{0}{46})^2 - (\frac{1}{46})^2 - (\frac{45}{46})^2 \\ &= 1 - (0)^2 - (0.0217)^2 - (0.9783)^2 \\ &= 1 - 0 - 0.0005 - 0.9570 \\ &= 0.0425 \end{aligned}$$

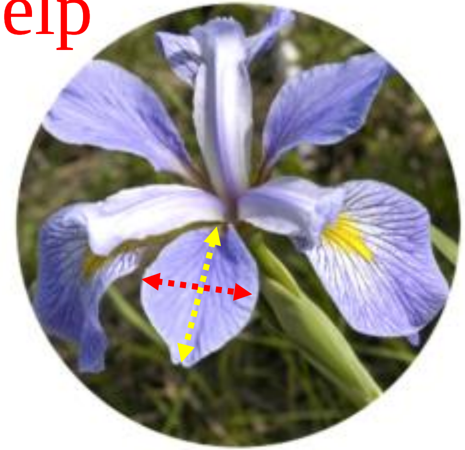
Petal length alone is enough to separate Iris-setosa while narrower petal width isolates Iris-versicolor

petal length　　petal width



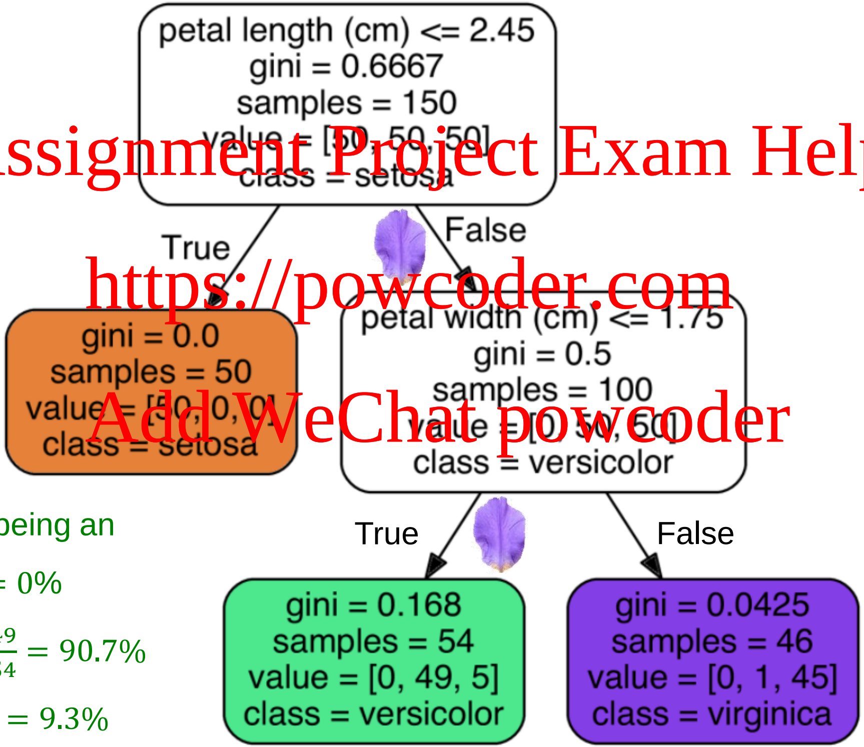
IRIS SETOSA

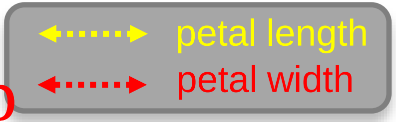
IRIS VERSICOLOR



IRIS VIRGINICA

The leaf node provides a predicted classification as well as the probability of the prediction





Probability of  being an

Iris-setosa: $\frac{0}{54} = 0\%$

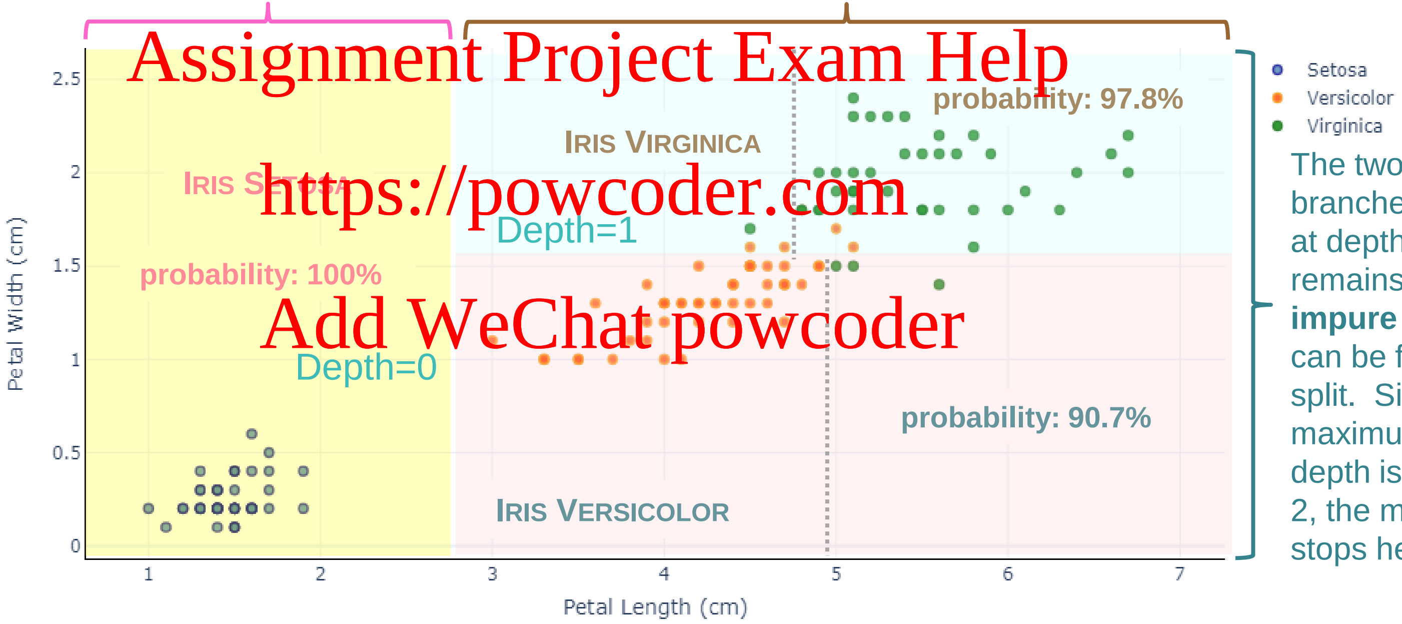
Iris-versicolor: $\frac{49}{54} = 90.7\%$

Iris-virginica: $\frac{5}{54} = 9.3\%$

The growth of the tree is determined by the purity of the branches

This region is **pure** and cannot be split anymore

If the maximum depth were set at 3, the two impure branches would be further split along the **dotted** lines generating purer branches



The two branches split at depth 1 remains **impure** and can be further split. Since the maximum depth is set at 2, the model stops here

The best split for a decision tree node is chosen by minimizing the impurity of the branches

- Gini Impurity is the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution

- It can be interpreted as a probabilistic measure stating the homogeneity of the random labelling of a dataset according to the class distribution

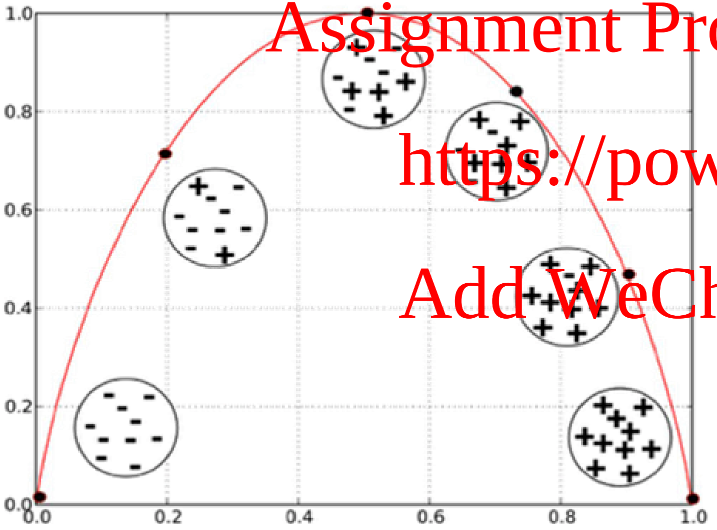
$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

  - G_i is the Gini impurity score for node i
  - n is the number of categories
  - $p_{i,k}$ is the ratio of k instances among all the data covered at node i

- The best split is chosen by maximizing the Gini Gain, which is calculated by subtracting the weighted impurities of the branches from the original impurity.

Gini impurity is a measure of "information", "surprise", or "uncertainty" inherent in the features' possible outcomes

Gini Impurity



In the building of a decision tree, the growth of the tree is to reduce uncertainty by maximizing the homogeneity within each branches

Decision trees are learned by recursive partitioning via selecting splitting features and branching by feature values

- **Growth Termination** - the growth of a tree node is determined by the observations corresponding to the node (i.e. the observation set) and will terminate when
  - All observations in the observation set are of the same class – a **pure** partition
  - **Number of observations** in the observation set is less than a specified **minimum**
  - **Depth** of the current node is greater than a specified **maximum**
  - The **improvement of class impurity** for the best available split of the node's observation set is less than a specified **minimum** (i.e. when splitting adds little value to the prediction)

- **Node Prediction** – class of the most frequent feature in the observations set is assigned as the prediction of the node

- **Split Selection** – for each feature, identify a list of candidate split values and then calculate the impurity measure for each split, weight each impurity measure with the relative size of each split, sum the weighted impurity measures, and select the feature with the lowest sum of weighted impurity measures and the corresponding split value

Petal Length	1.6	2.45	4.8	6.9
Setosa	44	6	0	0
Versicolor	0	0	46	4
Virginica	0	0	3	47
All	**44**	**6**	**49**	**51**
Node Size	**150**			
(Setosa/**All**)^2	1.0	1.0	0.0	0.0
(Versicolor/**All**)^2	0.0	0.0	0.881	0.006
(Virginica/**All**)^2	0.0	0.0	0.004	0.849
Gini impurity	**0.0**	**0.0**	**0.115**	**0.145**
All/Node Size	0.293	0.040	0.327	0.340
Weighted Gini	0.0	0.0	0.038	0.049
Selection Measure	**0.087**			

Sepal Length	5.1	5.8	6.5	7.9
Setosa	36	14	0	0
Versicolor	4	20	18	8
Virginica	1	5	22	22
All	**41**	**39**	**40**	**30**
Node Size	**150**			
(Setosa/All)^2	0.771	0.129	0.0	0.0
(Versicolor/All)^2	0.010	0.263	0.203	0.071
(Virginica/All)^2	0.001	0.010	0.303	0.538
Gini impurity	**0.219**	**0.592**	**0.495**	**0.391**
All/Node Size	0.273	0.260	0.267	0.20
Weighted Gini	0.060	0.154	0.132	0.078
Selection Measure	**0.424**			

Petal Width	0.3	1.4	1.75	2.5
Setosa	41	9	0	0
Versicolor	0	35	14	1
Virginica	0	1	4	45
All	**41**	**45**	**18**	**46**
Node Size	**150**			
(Setosa/All)^2	1.0	0.040	0.0	0.0
(Versicolor/All)^2	0.0	0.605	0.605	0.0
(Virginica/All)^2	0.0	0.0	0.049	0.957
Gini impurity	**0.0**	**0.355**	**0.345**	**0.043**
All/Node Size	0.273	0.300	0.120	0.307
Weighted Gini	0.0	0.100	0.041	0.013
Selection Measure	**0.161**			

Sepal Width	2.8	3.0	3.3	4.4
Setosa	1	7	11	31
Versicolor	27	15	7	1
Virginica	19	14	12	5
All	**47**	**36**	**30**	**37**
Node Size	**150**			
(Setosa/All)^2	0.0	0.038	0.134	0.702
(Versicolor/All)^2	0.330	0.174	0.054	0.001
(Virginica/All)^2	0.163	0.151	0.160	0.018
Gini impurity	**0.506**	**0.637**	**0.651**	**0.279**
All/Node Size	0.313	0.240	0.200	0.247
Weighted Gini	0.159	0.153	0.130	0.069
Selection Measure	**0.511**			

Selection Measure

- One for each candidate feature
- Based on the size & Gini impurity score of all the splits of the feature
- The feature with the **lowest** impurity measure is selected
- The split of the selected feature with the least Gini impurity will become the node's splitting value

The selection measure with the least value determines from which node the decision tree will grow for the next level

- The selection of a splitting criterion is based the sum of the weighted gini impurity of each factor

$$I(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} \cdot I(S_i)$$

  - The dataset covered by the node is to be partitioned into a number of data subsets S_i based on some proposed splitting criteria
  - $I(S_i)$ is the gini impurity index of the proposed data subset S_i
  - $|S|$ represents the cardinality of the original dataset S
  - $|S_i|$ represents the cardinality of the proposed data subset S_i

- The splitting criteria with the least $I(S)$ is to be selected to grow the next level of branches for the decision tree

Python: Decision Tree Classifier

load relevant modules

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```


instantiate a decision tree classifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
evaluate the model using cross validation
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
specify the cross validation to be a 3 folds cross validation
specify the "accuracy" metric to be the model evaluation metric
https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter


```
dtc = DecisionTreeClassifier()
cross_val_score(dtc, data.drop('target',axis=1), data['target'], cv=3, scoring='accuracy')
```

```
array([0.98, 0.94, 0.98])
```

Python: Decision Tree Classifier – Using Only Two Features

load relevant modules

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
```


project two features, petal length and petal width, to predict the Iris species
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
evaluate the model using cross validation
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html


```
X = iris.data[:,2:]
y = iris.target
```


instantiate a decision tree classifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
dtc = DecisionTreeClassifier(max_depth=2)
```

Python: Decision Tree Classifier – Performance

```
# Perform cross-validation by dividing the training data into 3 datasets
# Use accuracy as the performance score
```

```
scores = cross_val_score(dtc, data.drop('target', axis=1), data['target'],
                         cv=3,
                         scoring='accuracy')
```



```
# Display the score for each cross validation
# Display the average score for the 3 cross validation
```

```
print(scores)
print(scores.mean())
```

```
[0.96 0.92 0.92]
0.9333333333333332
```

```
# Fit the classifier with the training dataset
```

```
dtc.fit(X, y)
```

Python: Decision Tree Classifier – Show the Decision Tree

import drawing libraries

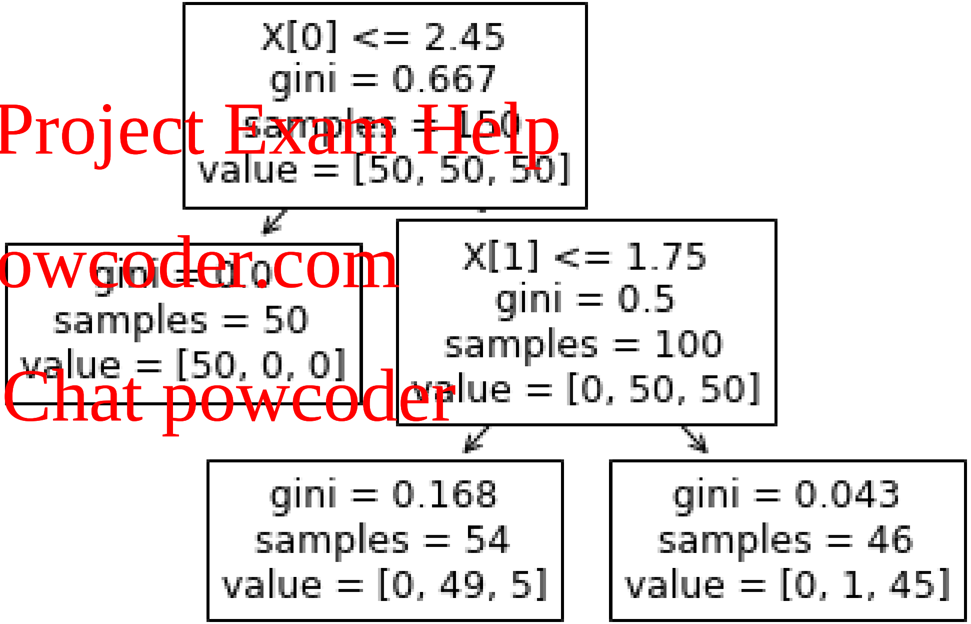
```
from sklearn import tree
from matplotlib.pyplot as plt
```

show attributes related to the dataset

```
print(iris.feature_names[2:])
print(iris.target_names)
print(data.columns)
print(data.target.nunique())
print(data.target.unique())
data.shape
```

display the decision tree

```
tree.plot_tree(dtc);
```

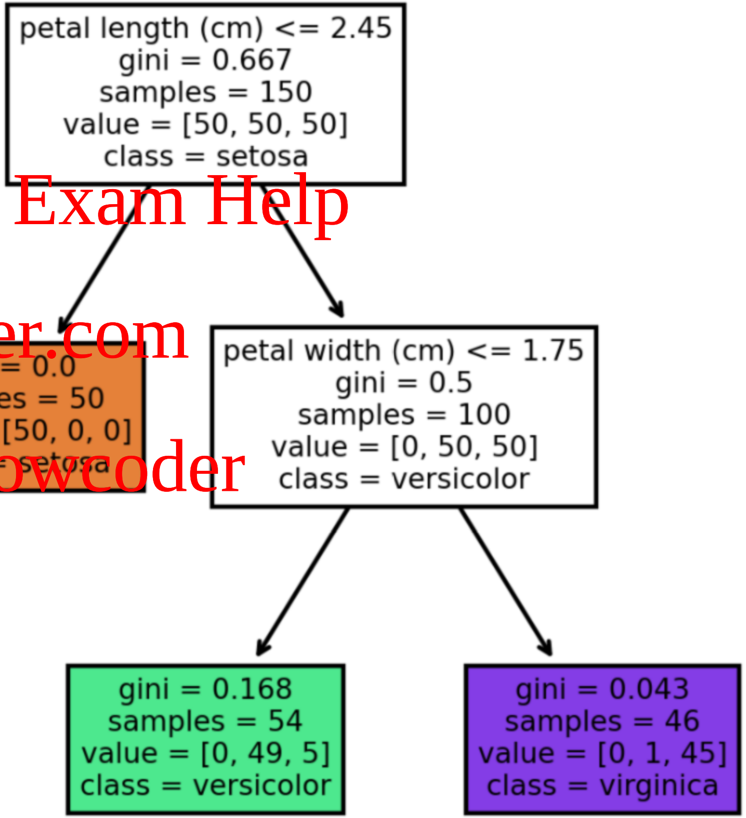


Python: Decision Tree Classifier – Show Majority Classes

```
# ['petal length (cm)','petal width (cm)']
fn = iris.feature_names[2:]

# ['setosa', 'versicolor', 'virginica']
cn = iris.target_names

# paint nodes to indicate majority class
fig, axes = plt.subplots(
                nrows=1, ncols=1,
                figsize=(4,4), dpi=300)
tree.plot_tree(dtc,
               feature_names=fn,
               class_names=cn,
               filled=True);
```



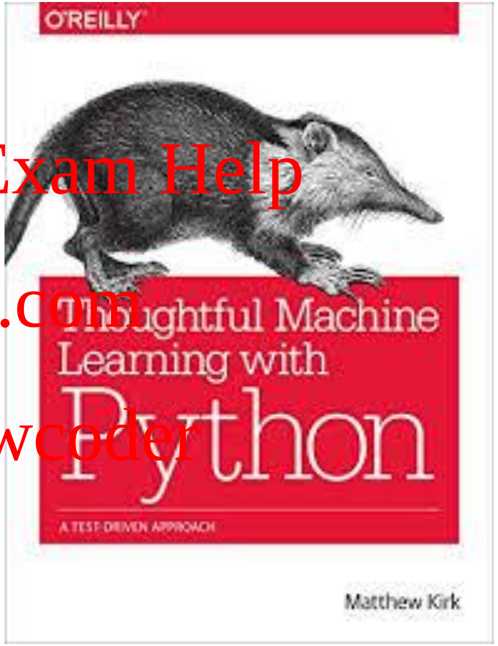
Decision Trees are greedy, top-down, recursive partitioning techniques

	Property	Description
1	Feature Data Types	Both categorical and numerical data.
2	Target Data Types	Categorical - class membership.
3	Key Principles	Feature is selected based on having a value that can provide the purest branches based on the impurity measure. Recursively partition nodes to branches until the leaf nodes are pure or the depth of tree is reached. Optimisation is performed at node level and not at the tree level.
4	Hyperparameters	Impurity measure (Gini impurity measure or Entropy measure). Depth of tree. Regularization parameters (e.g. minimum number of observations at each node, maximum number of leaf nodes) are used to reduce the risk of overfitting as the training is highly unconstrained.
5	Data Assumptions	Non-parametric. Very little data preparation. No feature scaling or centering is required.
6	Performance	Implicit feature selection. Can be unstable because small variance in the data.
7	Accuracy	Greedy approach relies on local optimisation and cannot guarantee to return the globally optimal decision tree (which is NP-complete).
8	Explainability	Comprehensive explanation.

References

References

"Hands-On Machine Learning with Scikit-Learn and TensorFlow", Aurelien Geron, O'Reilly Media, Inc., 2017



"Thoughtful Machine Learning with Python", Matthew Kirk, O'Reilly Media, Inc., 2017

References

- "Introduction to Machine Learning" (https://tomaszgolan.github.io/introduction_to_machine_learning/)
- Iris dataset (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html)
- "Exploratory Data Analysis of IRIS Data Set Using Python", Venkata Sai Raddy Avuluri, 2019 (https://medium.com/@avulurivenkatasaireddy/exploratory-data-analysis-of-iris-data-set-using-python-823e54110d2d)
- "Profiling and Timing Code", Jake VanderPlas, 2016 (https://jakevdp.github.io/PythonDataScienceHandbook/01.07-timing-and-profiling.html)
- "Construction of Decision Tree: Attribute Selection Measures", R. Aruna Devi & K. Mirmala, International Journal of Advancements in Research & Technology, Vol. 2, Issue 4, 2013 (http://www.ijoart.org/docs/Construction-of-Decision-Tree--Attribute-Selection-Measures.pdf)

Thank You