Machine Learning

| Real Life Application

Prof. Dr. Mostafa Elhosseini

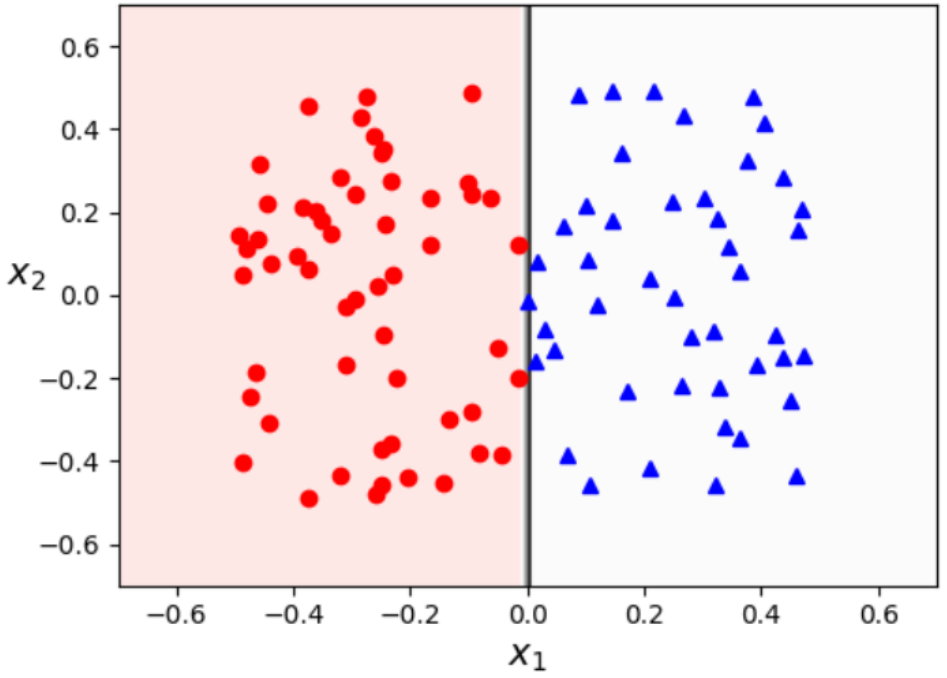
https://youtube.com/ElhosseiniAcademy

Agenda

- Sensitivity to Axis Orientation
- Decision Trees Have a High Variance
- Real-life application
  - Predicting graduate admissions.

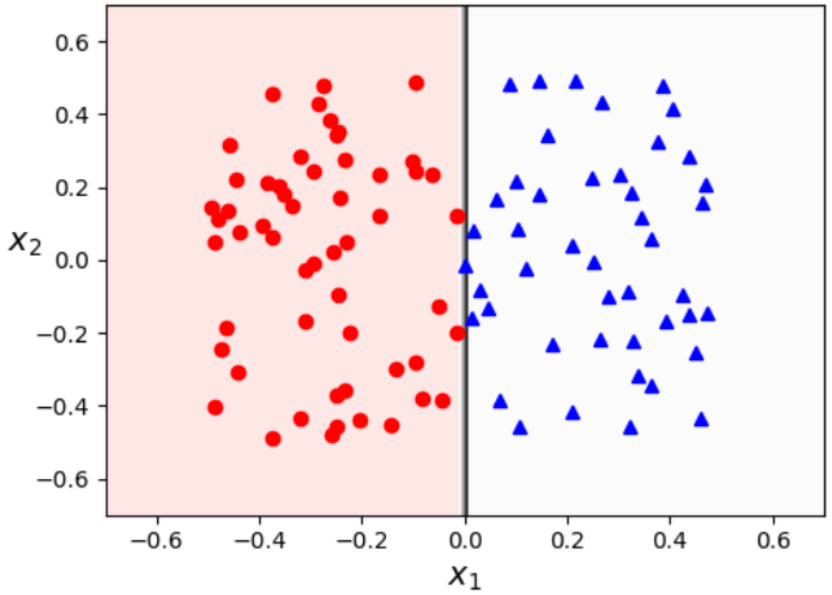
Sensitivity to Axis Orientation

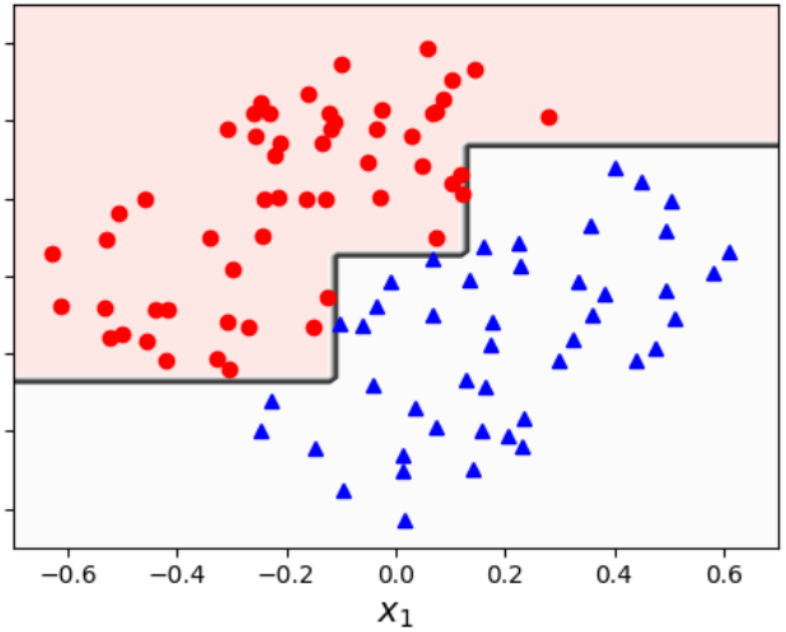
- Decision Trees love **orthogonal** decision boundaries (all splits are **perpendicular** to an axis), which makes them **sensitive** to the data's orientation.
- Linearly separable dataset



Sensitivity to Axis Orientation

- the dataset is **rotated by 45°**, the decision boundary looks **unnecessarily convoluted**





- Although both decision trees fit the training set perfectly, it is very likely that the model on the right will not generalize well.

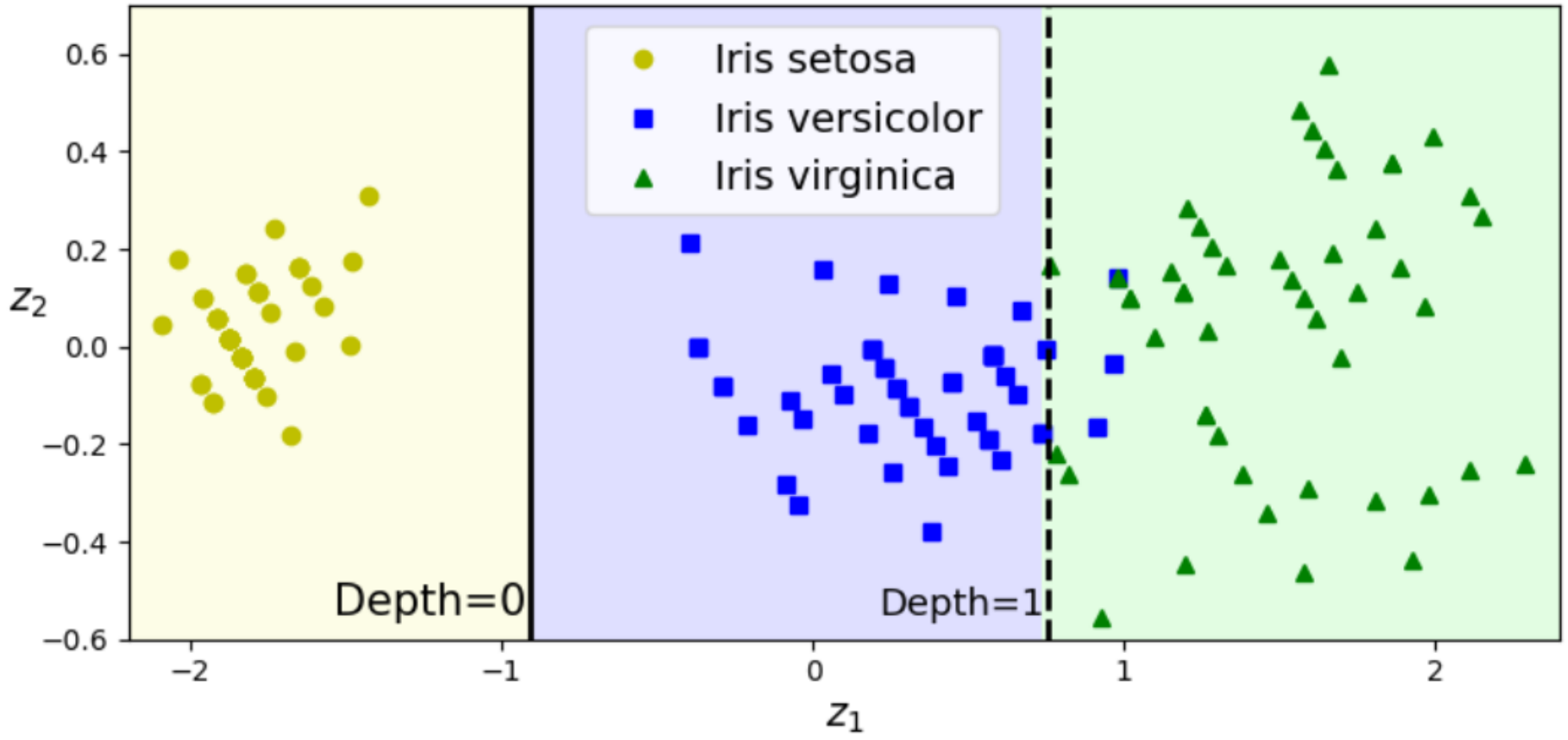
Sensitivity to Axis Orientation

- One way to limit this problem is to scale the data, then apply a **principal component analysis** (PCA) transformation
- It rotates the data in a way that reduces the correlation between the features, which often (not always) makes things easier for trees.

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

pca_pipeline = make_pipeline(StandardScaler(), PCA())
X_iris_rotated = pca_pipeline.fit_transform(X_iris)
tree_clf_pca = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf_pca.fit(X_iris_rotated, y_iris)
```

Sensitivity to Axis Orientation



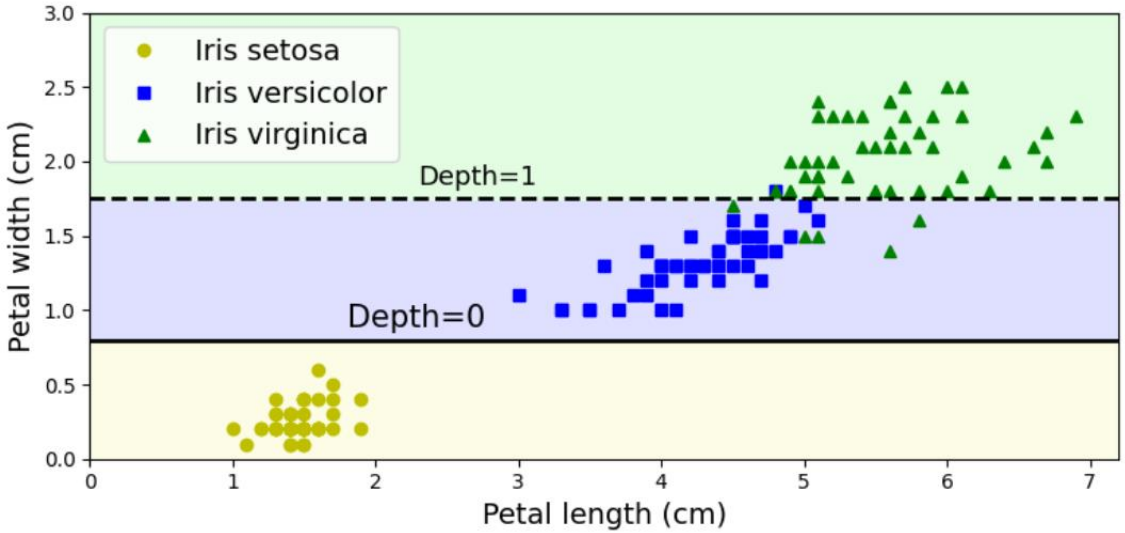
Decision Trees Have a High Variance

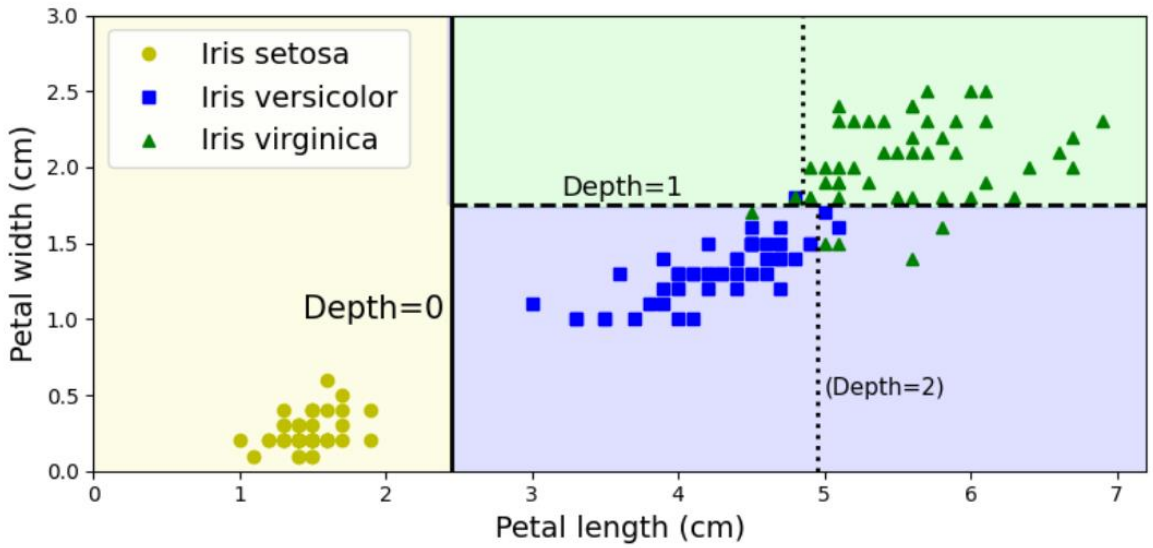
- Small changes to the hyperparameters or to the data may produce very different models (making their predictions unstable).

- In fact, since the **training algorithm** used by Scikit-Learn is **stochastic**—it **randomly selects the set of features** to evaluate at each node—even retraining the same decision tree on the exact same data may produce a very different model

Decision Trees Have a High Variance

- Small changes to the hyperparameters or to the data may produce very different models (making their predictions unstable).





Decision Trees Have a High Variance

- Adding a few data points or tweaking a hyperparameter (like the tree depth) may lead to a very different tree structure, which impacts the model's predictions

- In fact, since the **training algorithm** used by Scikit-Learn is **stochastic**—it **randomly selects the set of features** to evaluate at each node—even retraining the same decision tree on the exact same data may produce a very different model

Decision Trees Have a High Variance

- Luckily, by **averaging predictions over many trees**, it's possible to reduce variance significantly. Such an ensemble of trees is called a random forest, and it's one of the most powerful types of models available today
- Random forests are ensembles of multiple decision trees, where each tree is trained on different random subsets of the data and features.

Real-life application

- This dataset is commonly used in machine learning for **predicting graduate admissions**.
- It typically includes features such as **GRE scores**, **TOEFL scores**, undergraduate GPA, research experience, and more, aiming to predict the Chance of Admission to a graduate program.

Real-life application

- Use decision trees to build a model that predicts admission to graduate schools (Kaggle) .
- Dataset: Admission_Predict.csv
  - GRE score: a number out of 340
  - TOEFL score: a number out of 120
  - University rating: a number from 1 to 5
  - Statement of purpose strength (SOP): a number from 1 to 5
  - Undergraduate grade point average (CGPA): a number from 1 to 10
  - Letter of recommendation strength (LOR): a number from 1 to 5
  - Research experience: Boolean variable (0 or 1)

Real-life application

- Loading the Dataset

```
dataset = pd.read_csv('Admission_Predict.csv')
dataset = dataset.drop('Serial No.', axis=1)

# Display the first few rows
print("First 5 Rows of the Dataset:")
print(dataset.head())
```

```
First 5 Rows of the Dataset:
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research  \
0        337          118                  4  4.5   4.5  9.65         1
1        324          107                  4  4.0   4.5  8.87         1
2        316          104                  3  3.0   3.5  8.00         1
3        322          110                  3  3.5   2.5  8.67         1
4        314          103                  2  2.0   3.0  8.21         0

   Chance of Admit
0             0.92
1             0.76
2             0.72
3             0.80
4             0.65
```

Real-life application

- Understanding the Dataset

```
# Dataset Information
print("Dataset Information:")
print(dataset.info())

# Statistical Summary
print("\nStatistical Summary:")
print(dataset.describe())

# Check for Missing Values
print("\nMissing Values:")
print(dataset.isnull().sum())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   GRE Score          400 non-null    int64
 1   TOEFL Score        400 non-null    int64
 2   University Rating  400 non-null    int64
 3   SOP                400 non-null    float64
 4   LOR                400 non-null    float64
 5   CGPA               400 non-null    float64
 6   Research           400 non-null    int64
 7   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(4)
memory usage: 25.1 KB
None
```

Real-life application

- The **labels** on the dataset are the **chance of admission**, which is a number between 0 and 1. To have **binary labels**, we'll consider every student with a chance of **0.75 or higher as "admitted,"** and any other student as "not admitted."

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Admitted
337	118	4	4.5	4.5	9.65	1	True
324	107	4	4.0	4.5	8.87	1	True
316	104	3	3.0	3.5	8.00	1	False
322	110	3	3.5	2.5	8.67	1	True
314	103	2	2.0	3.0	8.21	0	False

Real-life application

- Features

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
337	118	4	4.5	4.5	9.65	1
324	107	4	4.0	4.5	8.87	1
316	104	3	3.0	3.5	8.00	1
322	110	3	3.5	2.5	8.67	1
314	103	2	2.0	3.0	8.21	0

- Labels

Admitted
True
True
False
True
False

Real-life application

- Training a decision tree

```
dt = DecisionTreeClassifier()
dt.fit(features, labels)
dt.predict(features[0:5])
```

GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Admitted
337	118	4	4.5	4.5	9.65	1	True
324	107	4	4.0	4.5	8.87	1	True
316	104	3	3.0	3.5	8.00	1	False
322	110	3	3.5	2.5	8.67	1	True
314	103	2	2.0	3.0	8.21	0	False

```
array([ True,  True, False,  True, False])
```

- The decision tree we just trained massively overfits
  - Score = 100%
  - Tree depth = 10

```
accuracy = dt.score(features, labels)
print(f"Accuracy of the Decision Tree Classifier: {accuracy:.2f}")
```

Real-life application

Hyperparameters Fine-tuning

- max_depth
  - The maximum number of levels (depth) the tree can grow. Limiting depth prevents the tree from becoming too complex and overfitting the data.
  - Default: None (nodes are expanded until all leaves are pure or contain fewer than min_samples_split samples).

Real-life application

Hyperparameters Fine-tuning

- min_samples_split
  - The minimum number of samples required to split an internal node. Increasing this number can prevent the tree from creating nodes that capture noise in the data.
  - Default: 2

Real-life application

Hyperparameters Fine-tuning

- min_samples_leaf
  - The minimum number of samples required to be at a leaf node. Ensuring a minimum number of samples at each leaf helps in smoothing the model and preventing overfitting.
  - Default: 1

Real-life application

Hyperparameters Fine-tuning

- max_features
  - The number of features to consider when looking for the best split. Reducing the number can decrease the variance and improve generalization.
  - Options: "auto", "sqrt", "log2", or an integer/floating number specifying the number of features.
  - Default: None (all features are considered).

Real-life application

Hyperparameters Fine-tuning

- criterion
  - The function to measure the quality of a split.
  - Options: "gini" for Gini impurity or "entropy" for information gain.
  - Default: "gini"