

Lecture 3: Supervised Learning

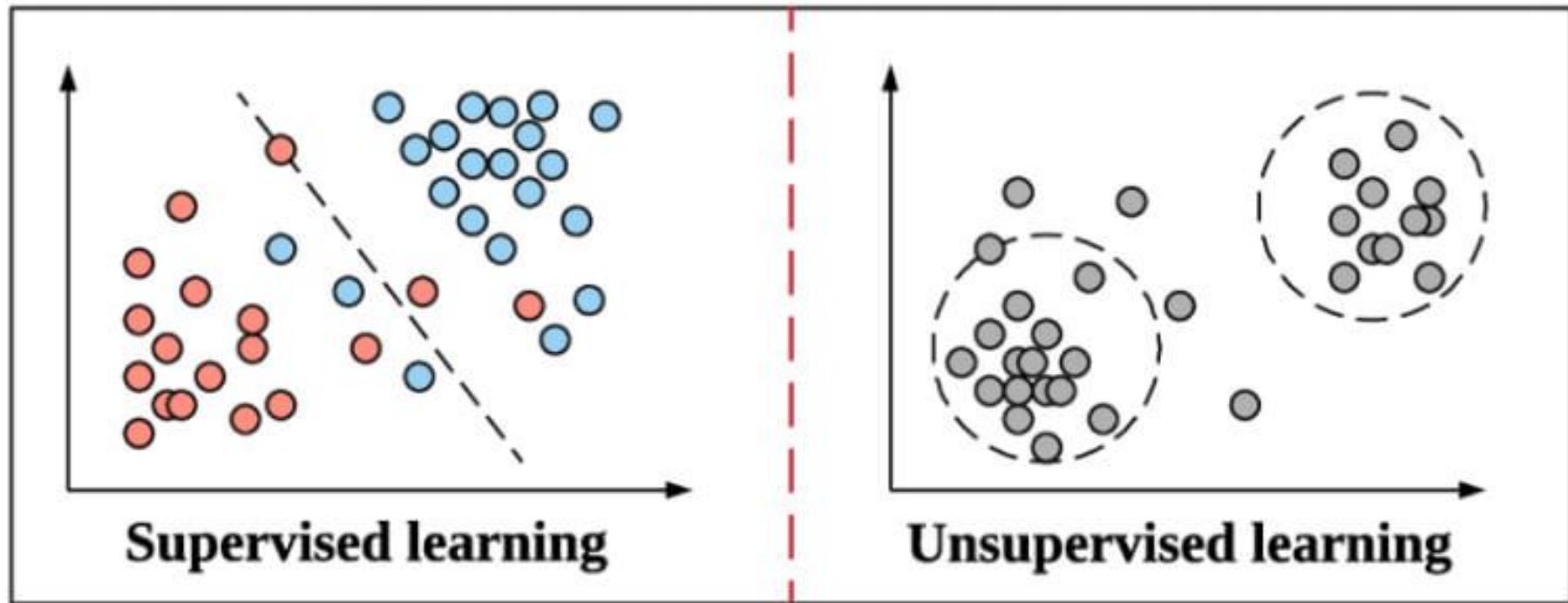
https://github.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2025s1

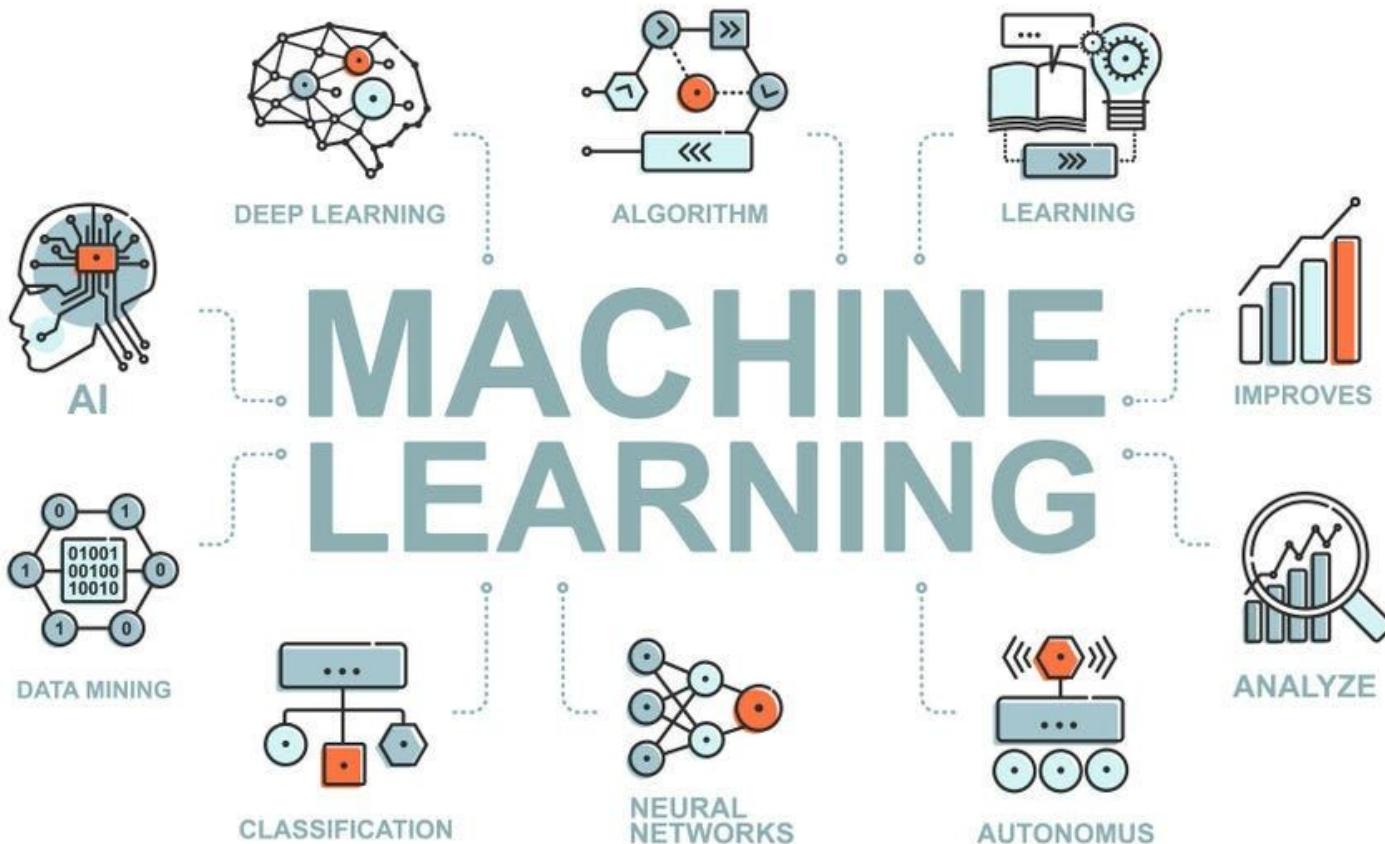
Teerapong Panboonyuen
<https://kaopanboonyuen.github.io>

Introduction to Supervised Learning

What is Supervised Learning?

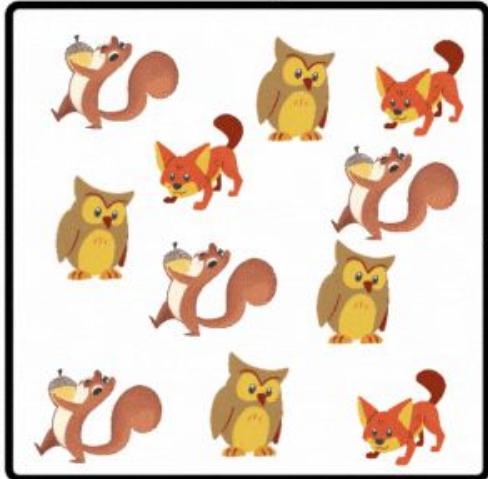
- **Definition:** Supervised learning is a machine learning technique where the model learns from labeled data to make predictions.
- **Goal:** The model tries to learn the relationship between input features and the output label (target variable).



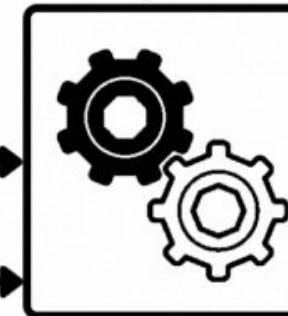


Supervised Learning

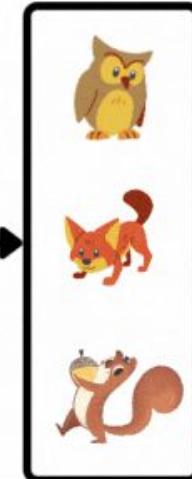
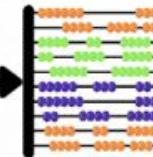
Labeled Training Data



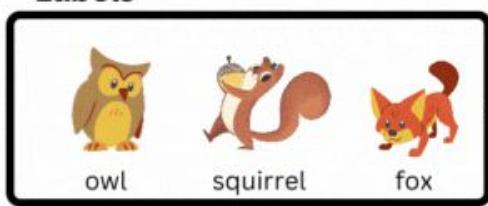
Training



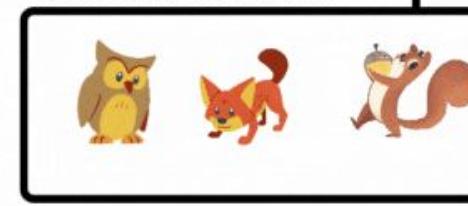
Prediction



Labels



Unlabeld Test Data



Datamapu

Introduction to Supervised Learning

Example in Practice:

- **Predicting Cancer Diagnosis:** Given a set of features such as age, gender, and medical history, we predict whether a patient has cancer (binary classification: "Cancer" or "No Cancer").

Key Terms:

- **Features (X):** The input variables (e.g., age, gender, test results).
- **Target (y):** The label we are trying to predict (e.g., cancer diagnosis).

REAL
ENGINEERING

MACHINE LEARNING VS. CANCER



REAL
ENGINEERING

MACHINE LEARNING VS. CANCER



Age

BMI

of
pregnancies

Pre- or post-
menopausal

GCPII C1561T

RFC1 G80A

cSHMT C1420T

Folate

B2

B6

B12

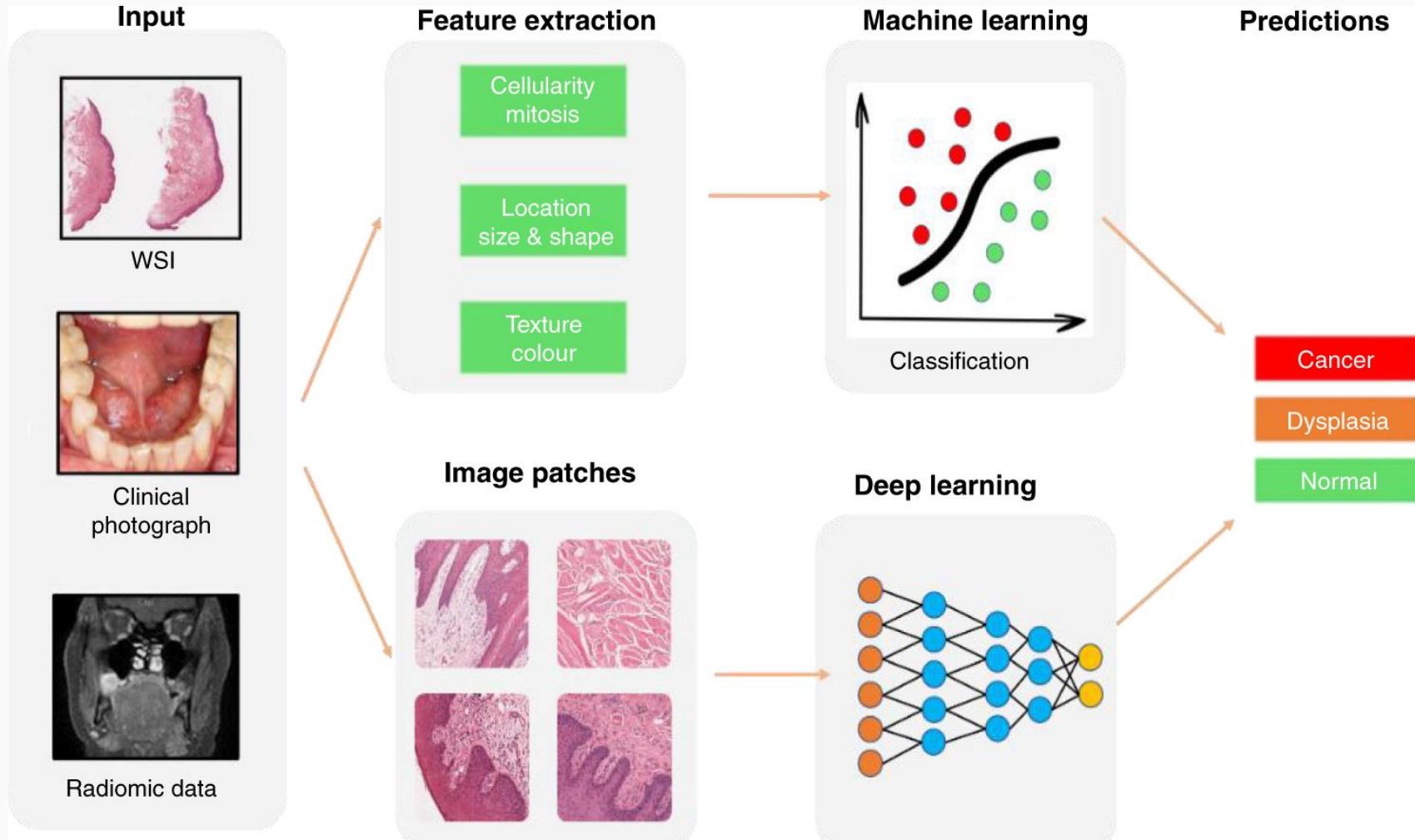


Input Variables

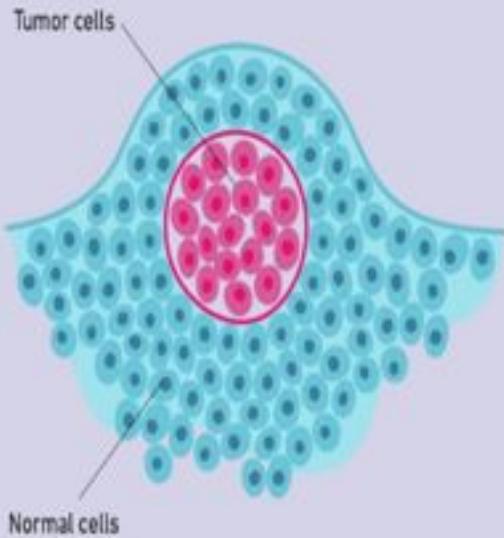


0 = no
1 = yes

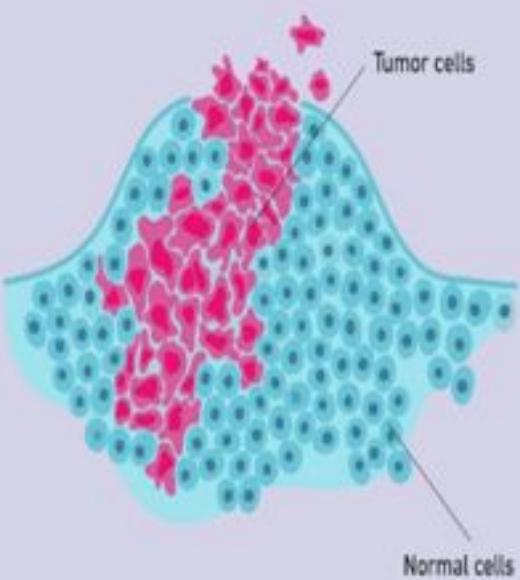
**Output
Variable**



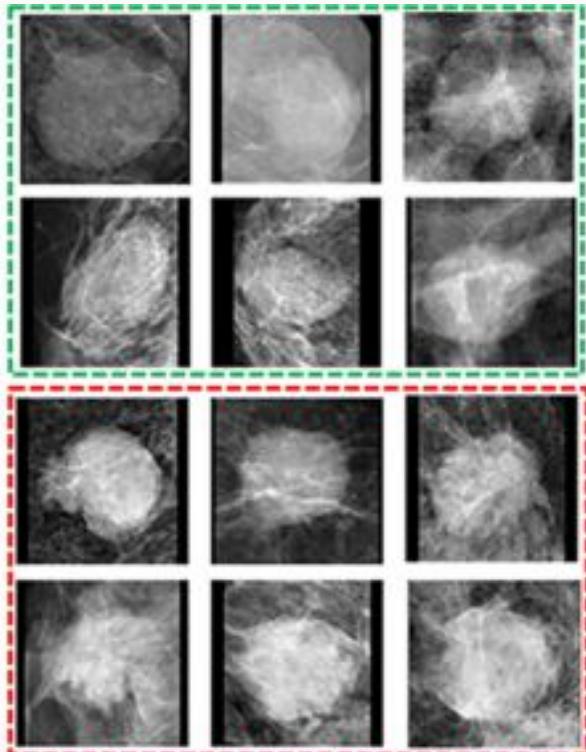
Benign tumor



Malignant tumor



(a)

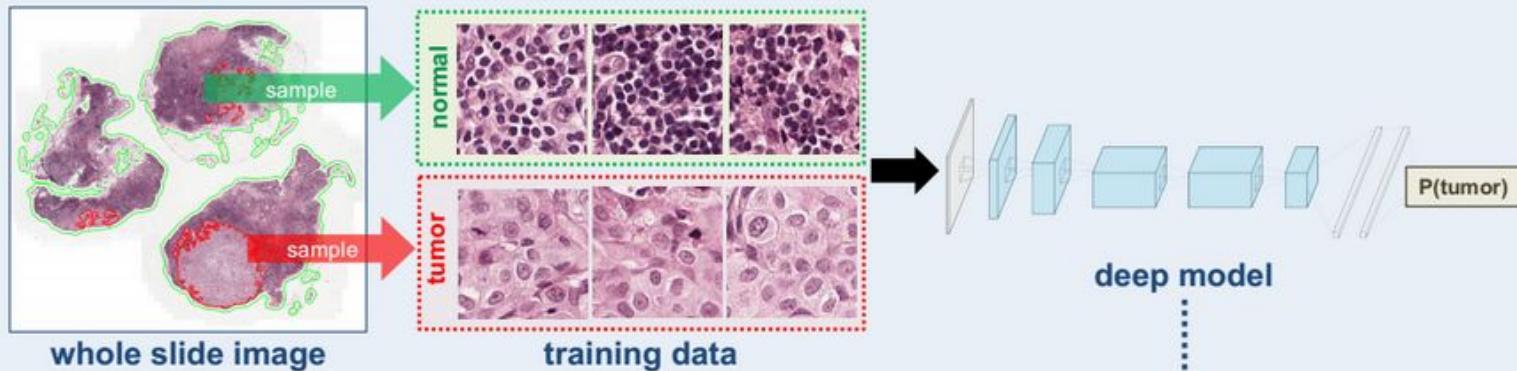


(b)

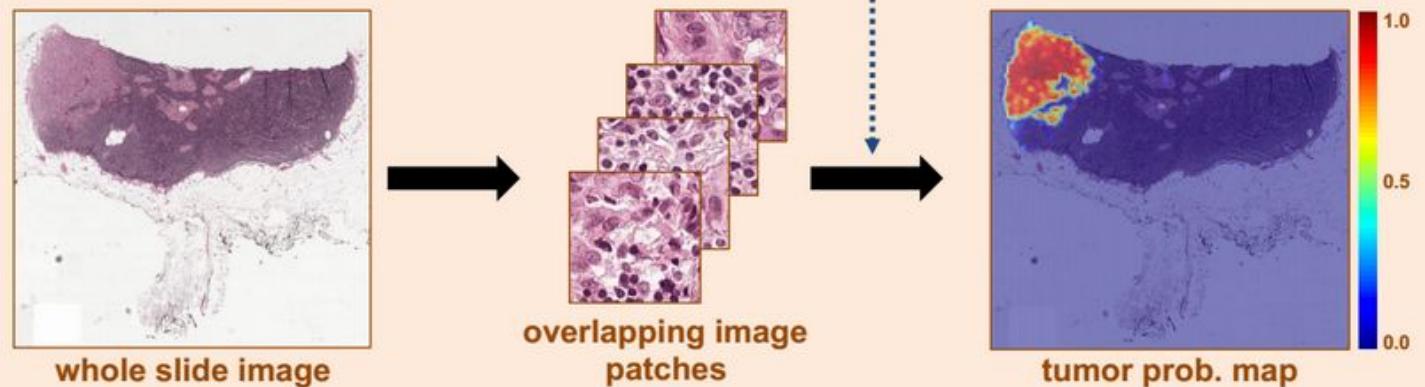
Benign

Malignant

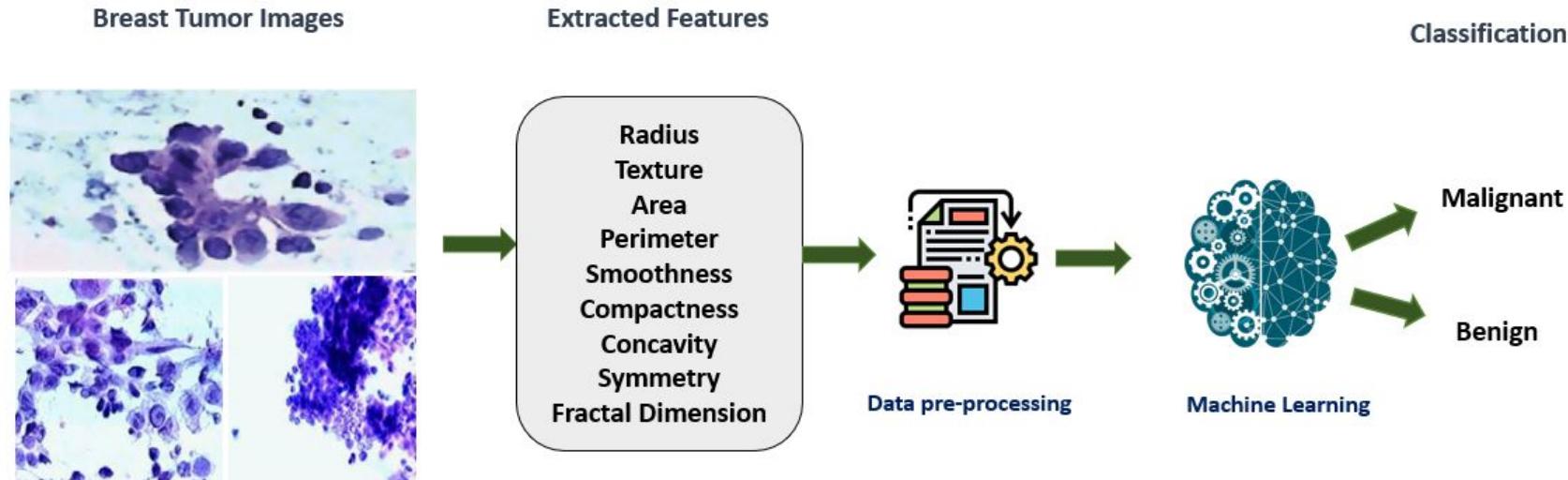
Train



Test



Breast Cancer Detection using Machine Learning



Overview of Scikit-learn for Supervised Learning

Why Scikit-learn?

- **Simple:** Scikit-learn provides an easy-to-use API for machine learning.
- **Wide Range of Algorithms:** Supports many algorithms for classification, regression, clustering, and more.
- **Efficient:** Offers optimized algorithms for quick model building.

Machine Learning Workflow in Scikit-learn

- 1. Load and Prepare Data**
- 2. Preprocess Data** (e.g., encoding, scaling, etc.)
- 3. Split Data** into training and test sets
- 4. Choose a Model** (e.g., Decision Tree, Logistic Regression)
- 5. Train the Model** using the training set
- 6. Evaluate the Model** using metrics (e.g., accuracy, precision, recall)
- 7. Fine-tune and Optimize** the model (e.g., hyperparameter tuning)
- 8. Deploy** for predictions on new data

Exploratory Data Analysis (EDA)

Before training a model, understanding the dataset is essential.

EDA helps uncover patterns, detect outliers, and understand the relationships between variables.

Basic Steps in EDA:

1. **Check Dataset Shape:** Understand the size and structure of your data.
2. **Summary Statistics:** Get a statistical summary for numerical features.
3. **Count Categories:** Understand the distribution of categorical variables.
4. **Visualize:** Use plots to identify patterns in the data.

Code Example:

python

 Copy  Edit

```
# Display basic information about the dataset
print(df.info())
print(df.describe())
```

Data Preprocessing: Handling Categorical Variables

Why is Preprocessing Important?

- **Machine Learning models require numerical data.**
- We need to convert categorical variables into numerical formats (e.g., Male/Female → 0/1).

One-hot Encoding:

- **Definition:** Convert each category in a feature into a separate binary column.
- **Example:** "Gender" (Male, Female) → Create columns for "Male" and "Female".

Code Example:

python

Copy Edit

```
# One-hot encode categorical variables
df = pd.get_dummies(df, columns=['Gender', 'Symptom'], dummy_na=True)
```

Splitting Data into Training and Testing Sets

To evaluate the model's performance, we split the data into two sets:

- **Training Set:** Used to train the model.
- **Testing Set:** Used to evaluate how well the model generalizes to unseen data.

Why Split the Data?

- The model is trained on the training data, and its accuracy is tested on the test data to simulate real-world scenarios.

Code Example:

python

 Copy  Edit

```
from sklearn.model_selection import train_test_split

X = df.drop(columns=['Diagnosis']) # Features
y = df['Diagnosis'] # Target label

# Split data with a random seed for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Choosing and Training Supervised Models

Different Supervised Learning Models:

1. **Decision Trees:** A flowchart-like model used for classification and regression.
2. **Random Forests:** An ensemble of decision trees to improve accuracy.
3. **Logistic Regression:** A statistical model used for binary classification.
4. **Neural Networks:** A complex model inspired by the human brain, used for more intricate data patterns.

Training the Models:

- Fit each model using the training data.
- Evaluate performance on the test set.

Code Example:

python

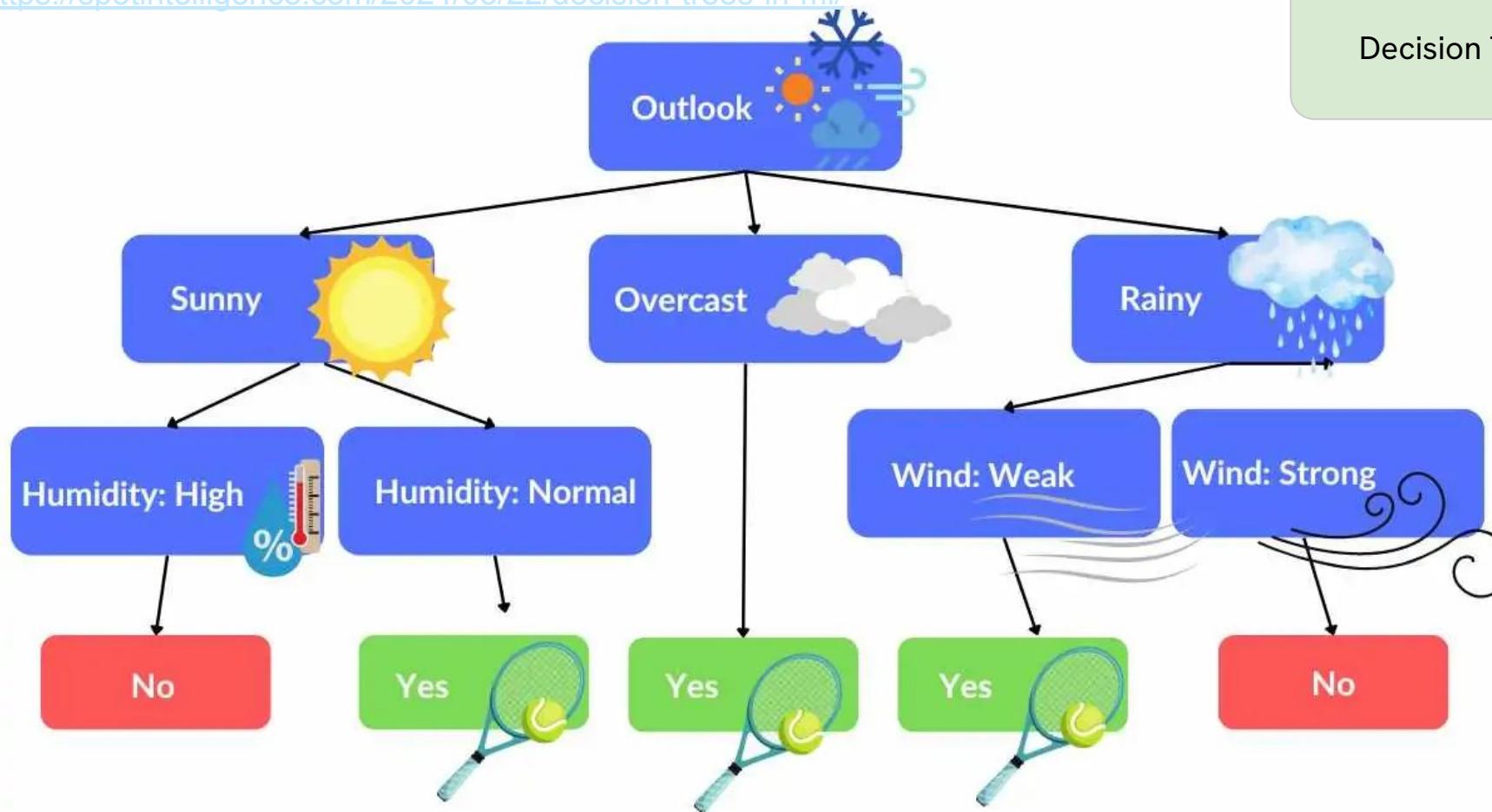
Copy Edit

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Neural Network": MLPClassifier(hidden_layer_sizes=(32, 16), max_iter=1000, random_st
}

# Train and evaluate models
results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results.append([name, acc])
```

Decision Tree



<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>



Install User Guide API Examples Community More ▾

sklearn.covariance

sklearn.cross_decomposition

sklearn.datasets

sklearn.decomposition

sklearn.discriminant_analysis

sklearn.dummy

sklearn.ensemble

sklearn.exceptions

sklearn.experimental

sklearn.feature_extraction

sklearn.feature_selection

sklearn.frozen

sklearn.gaussian_process

sklearn.impute

sklearn.inspection

sklearn.isotonic

sklearn.kernel_approximation

sklearn.kernel_ridge

sklearn.linear_model

sklearn.manifold

sklearn.metrics

sklearn.mixture

sklearn.model_selection

sklearn.multiclass

sklearn.multioutput

sklearn.tree

sklearn.tree.DecisionTreeClassifier

API Reference > sklearn.tree > DecisionTreeClassifier

DecisionTreeClassifier

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
```

[source]

A decision tree classifier.

Read more in the [User Guide](#).

Parameters:

criterion : {"gini", "entropy", "log_loss"}, default="gini"

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see [Mathematical formulation](#).

splitter : {"best", "random"}, default="best"

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : int, default=None

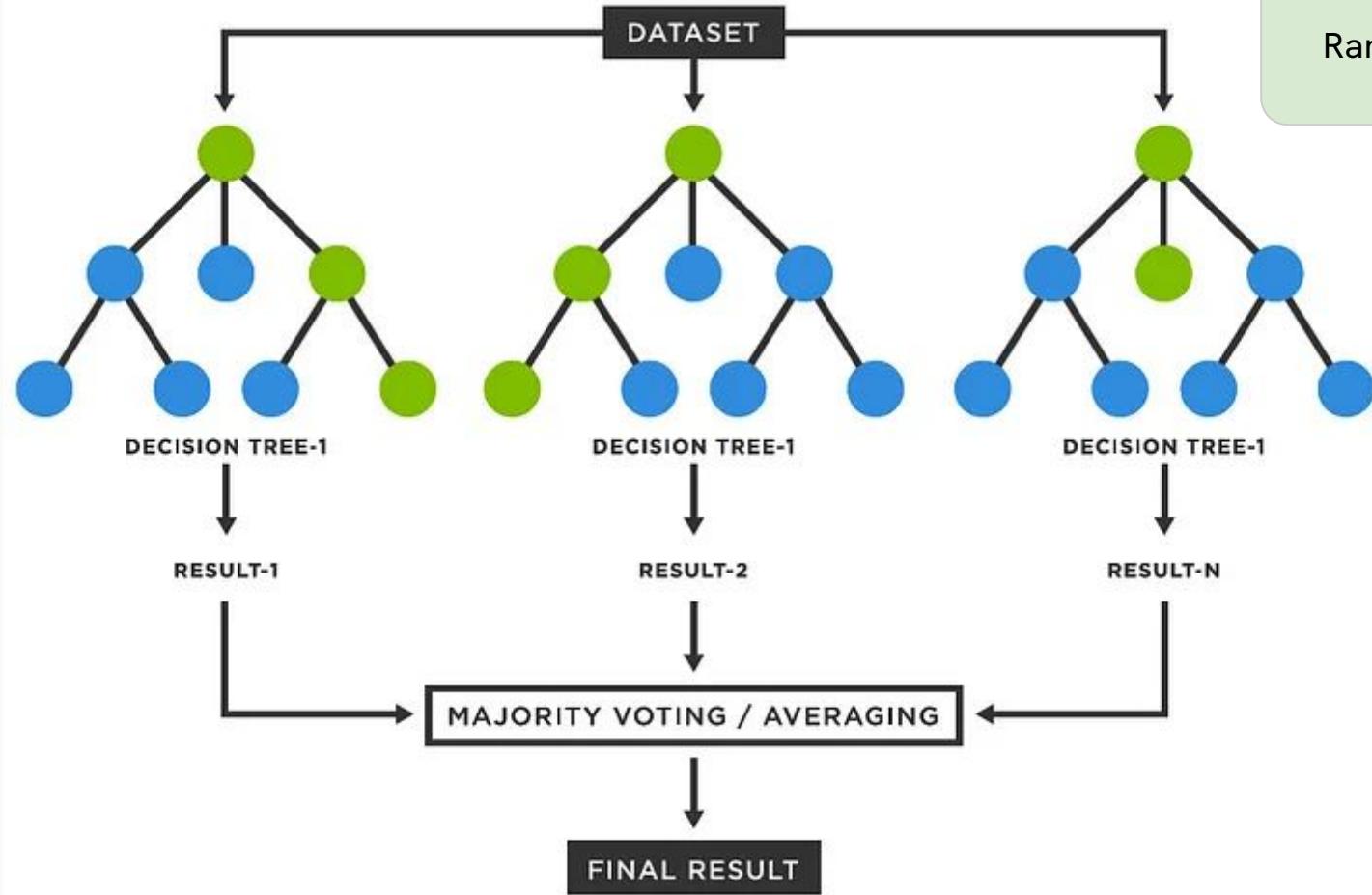
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split : int or float, default=2

The minimum number of samples required to split an internal node:

- If int, then consider min_samples_split as the minimum number.

Random Forest





Install User Guide API Examples Community More ▾

sklearn.covariance

sklearn.cross_decomposition

sklearn.datasets

sklearn.decomposition

sklearn.discriminant_analysis

sklearn.dummy

sklearn.ensemble

- AdaBoostClassifier
- AdaBoostRegressor
- BaggingClassifier
- BaggingRegressor
- ExtraTreesClassifier
- ExtraTreesRegressor
- GradientBoostingClassifier
- GradientBoostingRegressor
- HistGradientBoostingClassifier
- HistGradientBoostingRegressor
- IsolationForest
- RandomForestClassifier**
- RandomForestRegressor
- RandomTreesEmbedding
- StackingClassifier
- StackingRegressor
- VotingClassifier
- VotingRegressor

Home > API Reference > sklearn.ensemble > RandomForestClassifier

RandomForestClassifier

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
```

[\[source\]](#)

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. Trees in the forest use the best split strategy, i.e. equivalent to passing `splitter="best"` to the underlying [DecisionTreeClassifier](#). The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

For a comparison between tree-based ensemble models see the example [Comparing Random Forests and Histogram Gradient Boosting models](#).

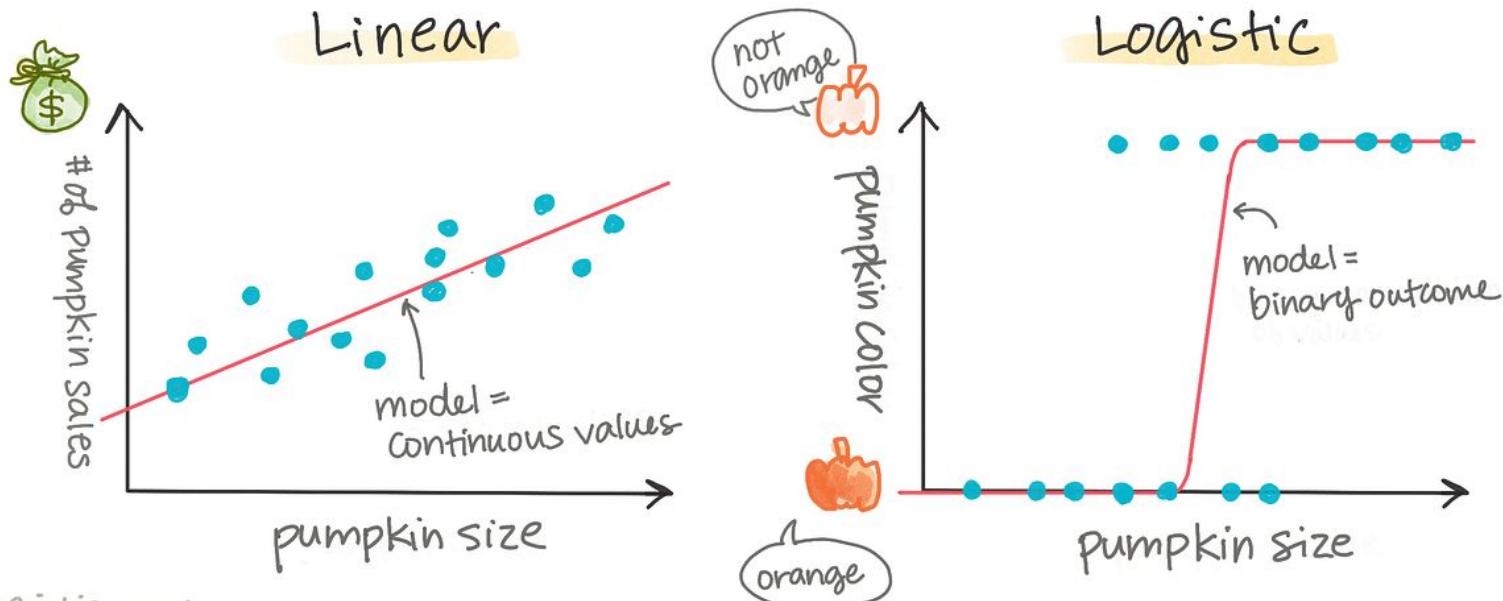
This estimator has native support for missing values (NaNs). During training, the tree grower learns at each split point whether samples with missing values should go to the left or right child, based on the potential gain. When predicting, samples with missing values are assigned to the left or right child consequently. If no missing values were encountered for a given feature during training, then samples with missing values are mapped to whichever child has the most samples.

Read more in the [User Guide](#).

Parameters:

n_estimators : int, default=100

LINEAR vs. LOGISTIC REGRESSION



@girliemac

sklearn

Install User Guide API Examples Community More ▾

sklearn.covariance

sklearn.cross_decomposition

sklearn.datasets

sklearn.decomposition

sklearn.discriminant_analysis

sklearn.dummy

sklearn.ensemble

sklearn.exceptions

sklearn.experimental

sklearn.feature_extraction

sklearn.feature_selection

sklearn.frozen

sklearn.gaussian_process

sklearn.impute

sklearn.inspection

sklearn.isotonic

sklearn.kernel_approximation

sklearn.kernel_ridge

sklearn.linear_model

LogisticRegression

LogisticRegressionCV

PassiveAggressiveClassifier

Perceptron

RidgeClassifier

RidgeClassifierCV

sklearn.linear_model.LogisticRegression

API Reference > sklearn.linear_model > LogisticRegression

LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None) [source]
```

Logistic Regression (aka logit, MaxEnt) classifier.

This class implements regularized logistic regression using the ‘liblinear’ library, ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ solvers. **Note that regularization is applied by default.** It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The ‘newton-cg’, ‘sag’, and ‘lbfgs’ solvers support only L2 regularization with primal formulation, or no regularization. The ‘liblinear’ solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty. The Elastic-Net regularization is only supported by the ‘saga’ solver.

For `multiclass` problems, all solvers but ‘liblinear’ optimize the (penalized) multinomial loss. ‘liblinear’ only handle binary classification but can be extended to handle multiclass by using `OneVsRestClassifier`.

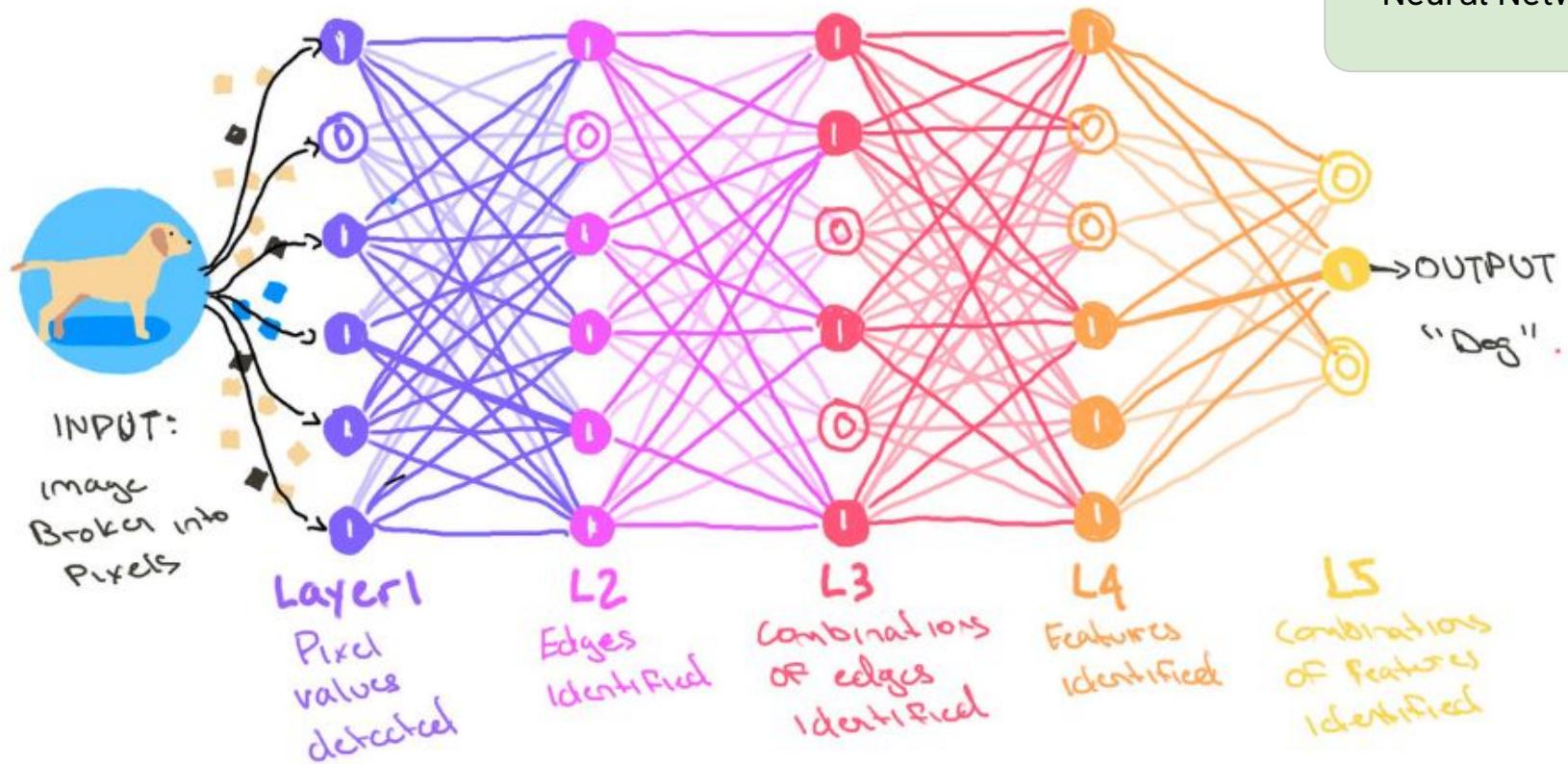
Read more in the [User Guide](#).

Parameters:

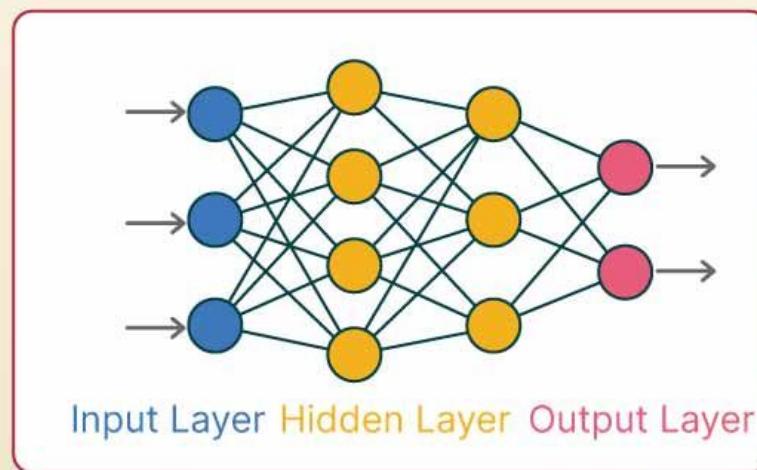
`penalty : {'l1', 'l2', 'elasticnet', None}, default='l2'`

Specify the norm of the penalty:

- `None` : no penalty is added;
- `'l2'` : add a L2 penalty term and it is the default choice;



Multilayer Perceptron (MLP) Neural Networks



MULTILAYER PERCEPTRON

Supervised Learning

scikit learn Install User Guide API Examples Community More ▾

sklearn.covariance
sklearn.cross_decomposition
sklearn.datasets
sklearn.decomposition
sklearn.discriminant_analysis
sklearn.dummy

sklearn.neural_network.MLPClassifier

MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

[\[source\]](#)

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

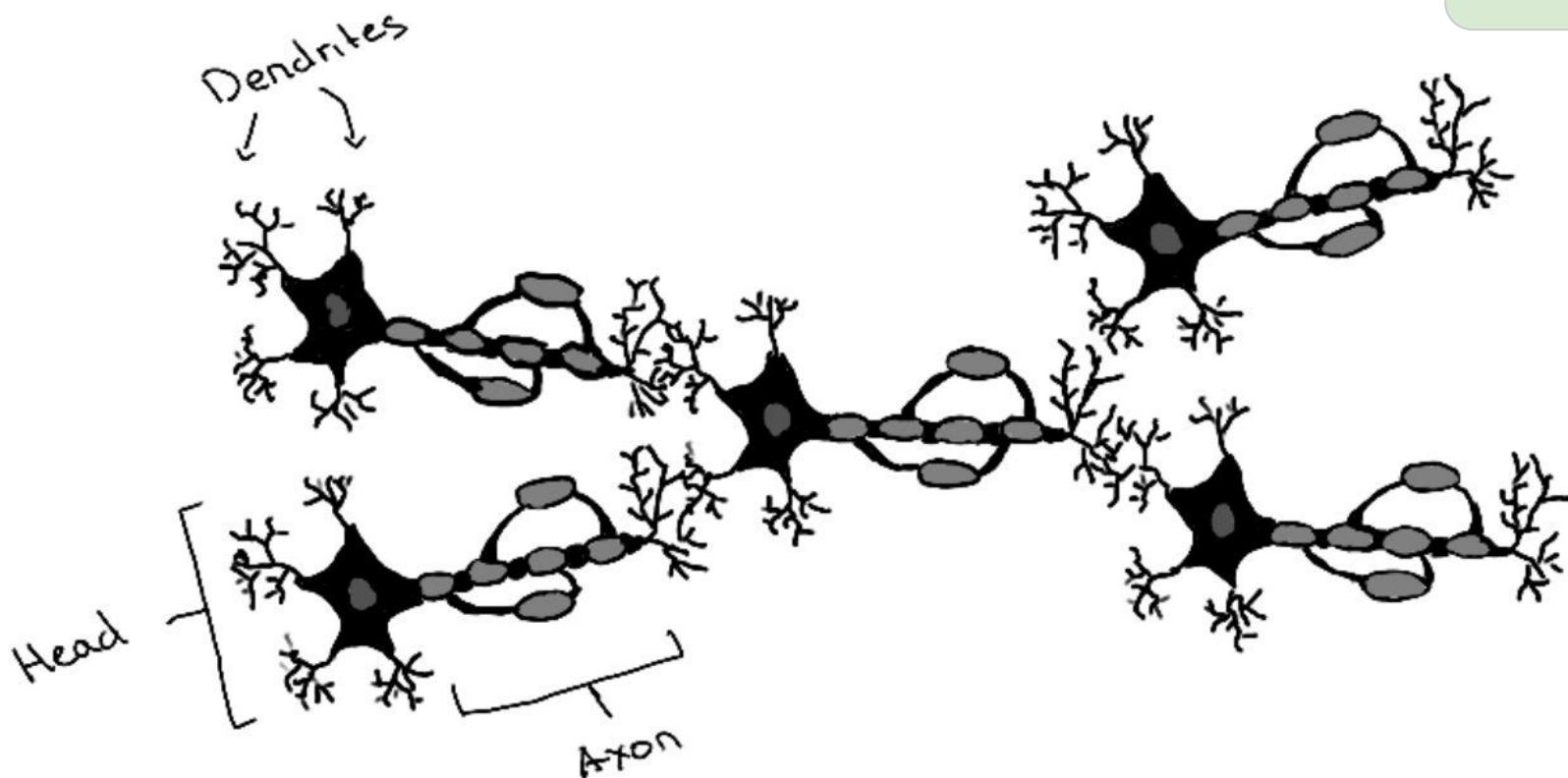
Added in version 0.18.

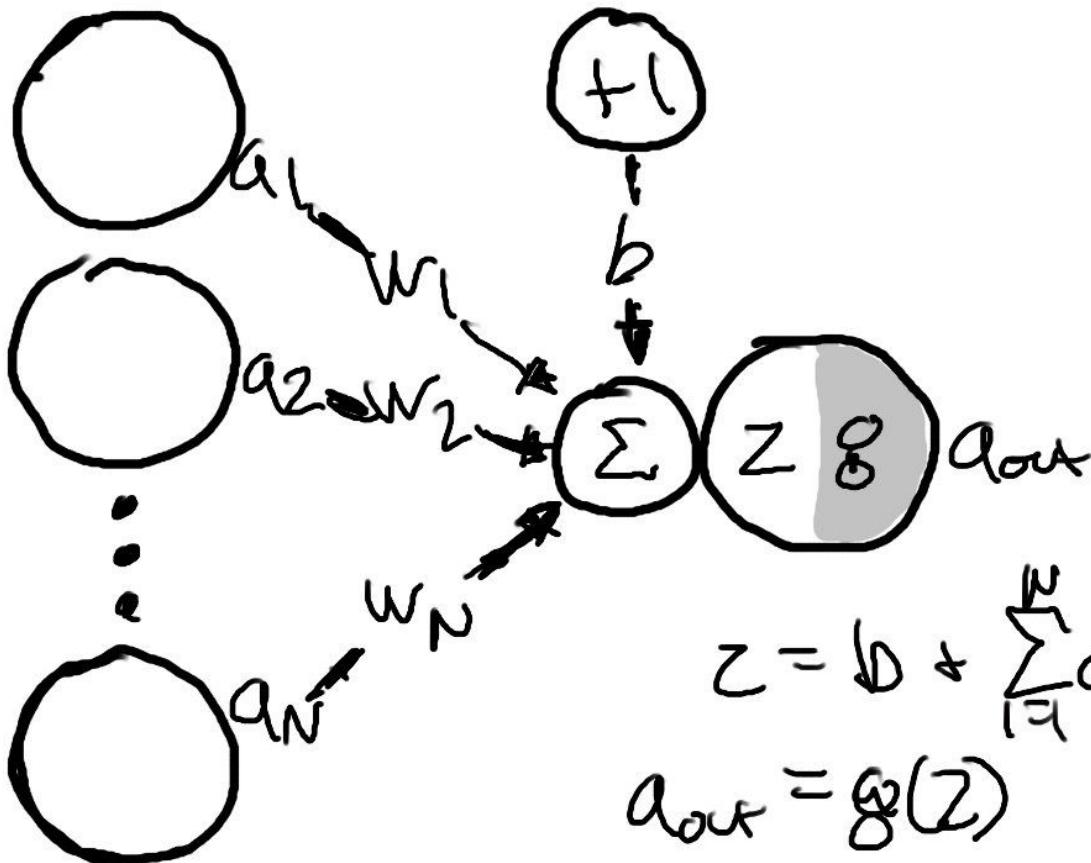
Parameters:

hidden_layer_sizes : array-like of shape(n_layers - 2,), default=(100,)
The ith element represents the number of neurons in the ith hidden layer.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns $f(x) = x$
- 'logistic', the logistic sigmoid function, returns $f(x) = 1 / (1 + \exp(-x))$.
- 'tanh', the hyperbolic tan function, returns $f(x) = \tanh(x)$.
- 'relu', the rectified linear unit function, returns $f(x) = \max(0, x)$





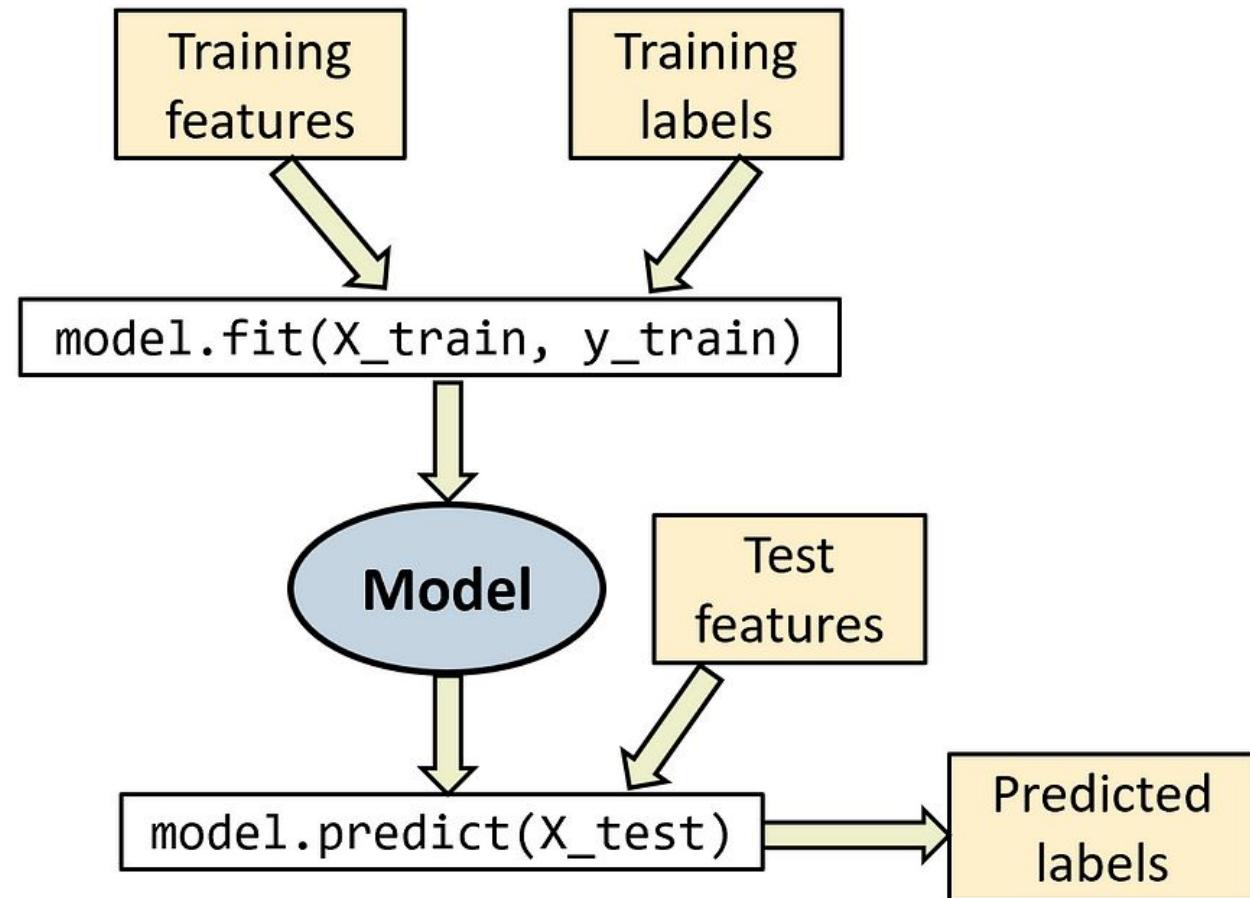
Model Evaluation: Accuracy, Precision, Recall, F1-Score

Model Evaluation Metrics:

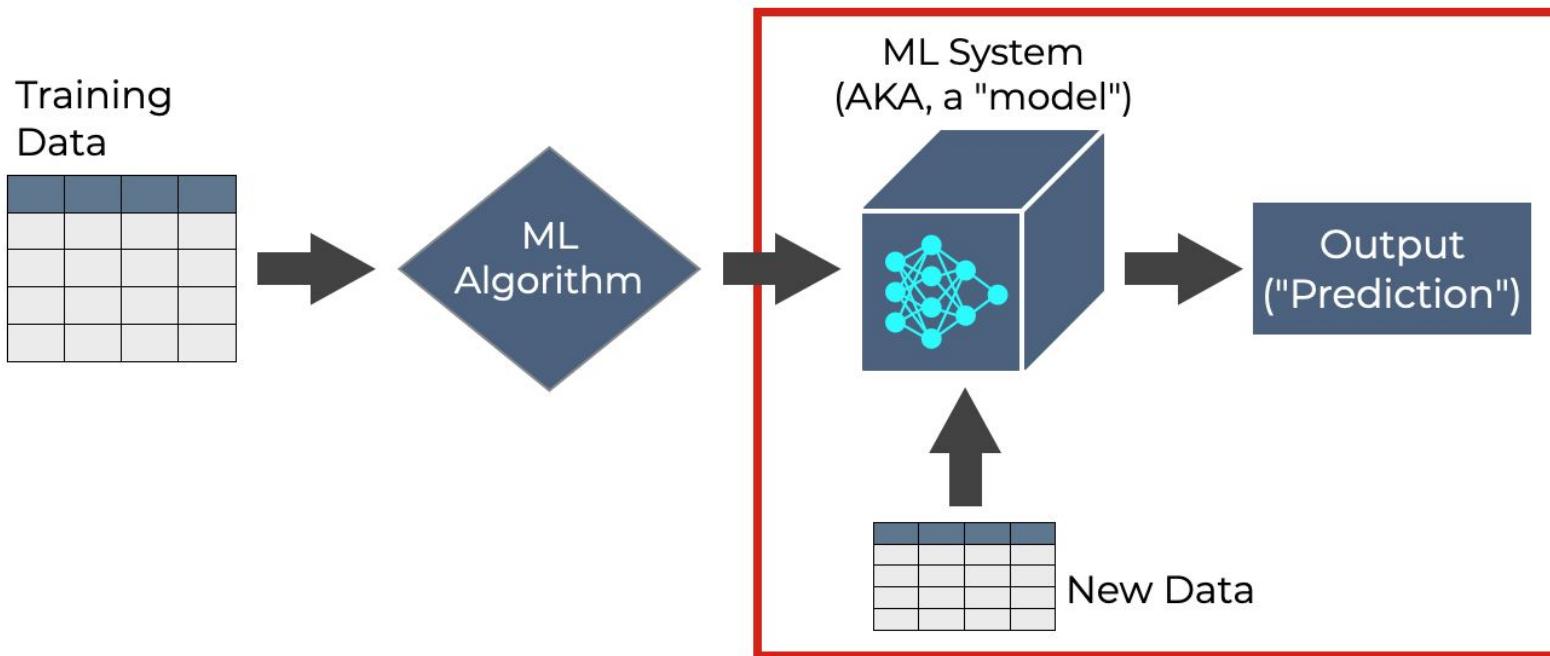
1. **Accuracy:** Percentage of correctly predicted labels.
2. **Precision:** How many of the predicted positives were actually positive?
3. **Recall:** How many actual positives were predicted correctly?
4. **F1-Score:** A balance between precision and recall (important for imbalanced datasets).

Why These Metrics Matter?

- **Accuracy** is not always reliable (e.g., in imbalanced datasets).
- **Precision** and **Recall** are useful when we care about the consequences of false positives or false negatives (e.g., diagnosing cancer).



WITH SOME MODELS, WE CAN MAKE A "PREDICTION" ON THE BASIS OF NEW INPUT VALUES



Code Example:

python

 Copy  Edit

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    pr = precision_score(y_test, y_pred)
    rc = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f'{name} - Accuracy: {acc}, Precision: {pr}, Recall: {rc}, F1 Score: {f1}'")
```

Visualizing Model Performance: Confusion Matrix

A **Confusion Matrix** is a summary of how well the model is performing. It shows the count of:

- **True Positives (TP)**: Correctly predicted positive cases.
- **False Positives (FP)**: Incorrectly predicted positive cases.
- **True Negatives (TN)**: Correctly predicted negative cases.
- **False Negatives (FN)**: Incorrectly predicted negative cases.

Visualizing the confusion matrix allows us to intuitively understand model performance.

Code Example:

python

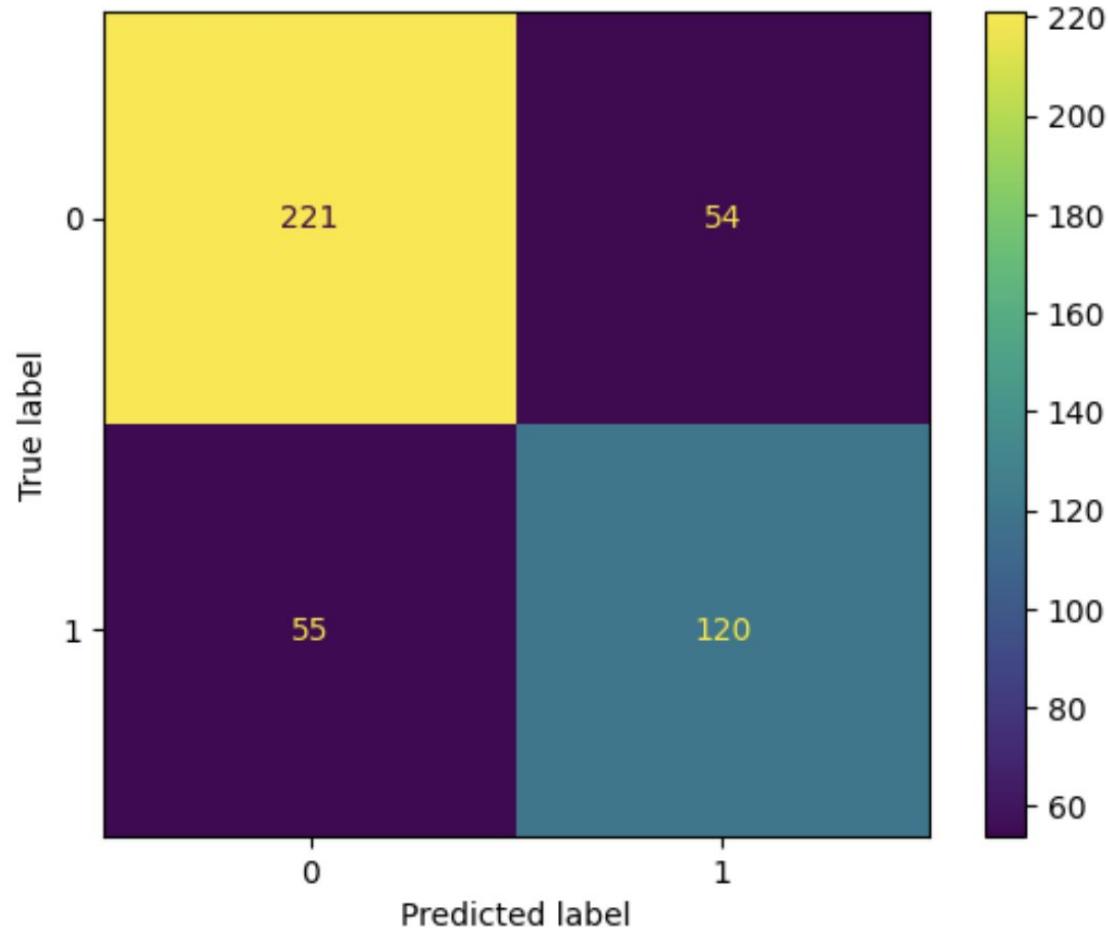
 Copy  Edit

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

for name, model in models.items():
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    plt.title(f"{name} Confusion Matrix")
    plt.show()
```

Decision Tree Confusion Matrix

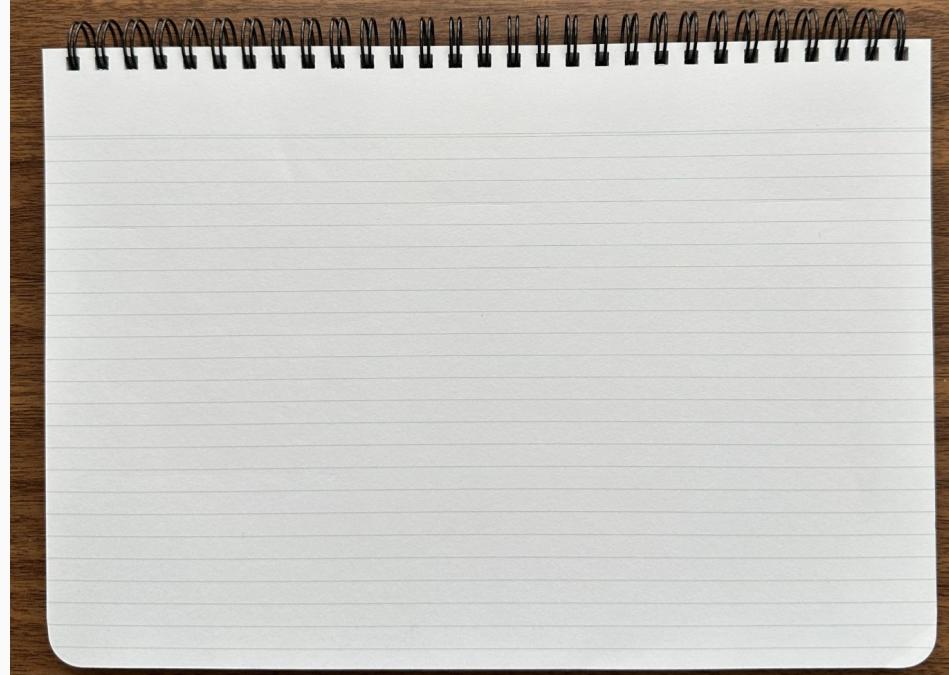


		POSITIVE	NEGATIVE
ACTUAL VALUES	POSITIVE	TP	FN
	NEGATIVE	FP	TN

$$\text{Precision} = \frac{TP}{TP + FP} \quad \text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$



Comparing Model Performance: Bar Plot

Once we evaluate the models using various metrics, we can **compare their performance** visually.

Using F1-Score for Comparison:

- **F1-Score** provides a balanced measure for imbalanced datasets.

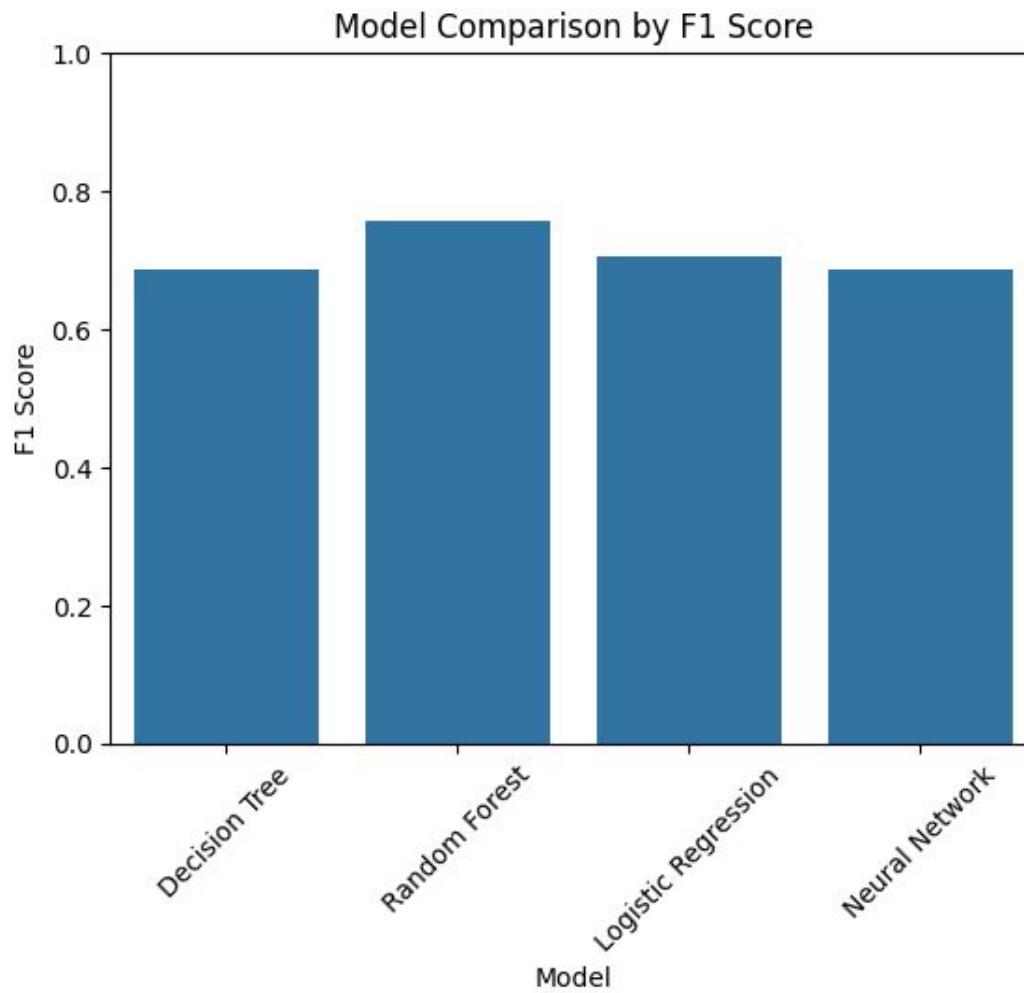
Code Example:

```
python

import seaborn as sns
import matplotlib.pyplot as plt

# Create a DataFrame of results
results_df = pd.DataFrame(results, columns=["Model", "F1 Score"])

# Visualize the comparison
sns.barplot(data=results_df, x='Model', y='F1 Score')
plt.title("Model Comparison by F1 Score")
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.show()
```



ROC Curve and AUC (Area Under the Curve)

The **ROC curve** helps visualize the trade-off between the True Positive Rate (Recall) and the False Positive Rate. The **AUC** score measures the overall ability of the model to distinguish between classes.

AUC Interpretation:

- **0.5**: No discrimination (random guessing).
- **1**: Perfect discrimination (ideal model).

Code Example:

python

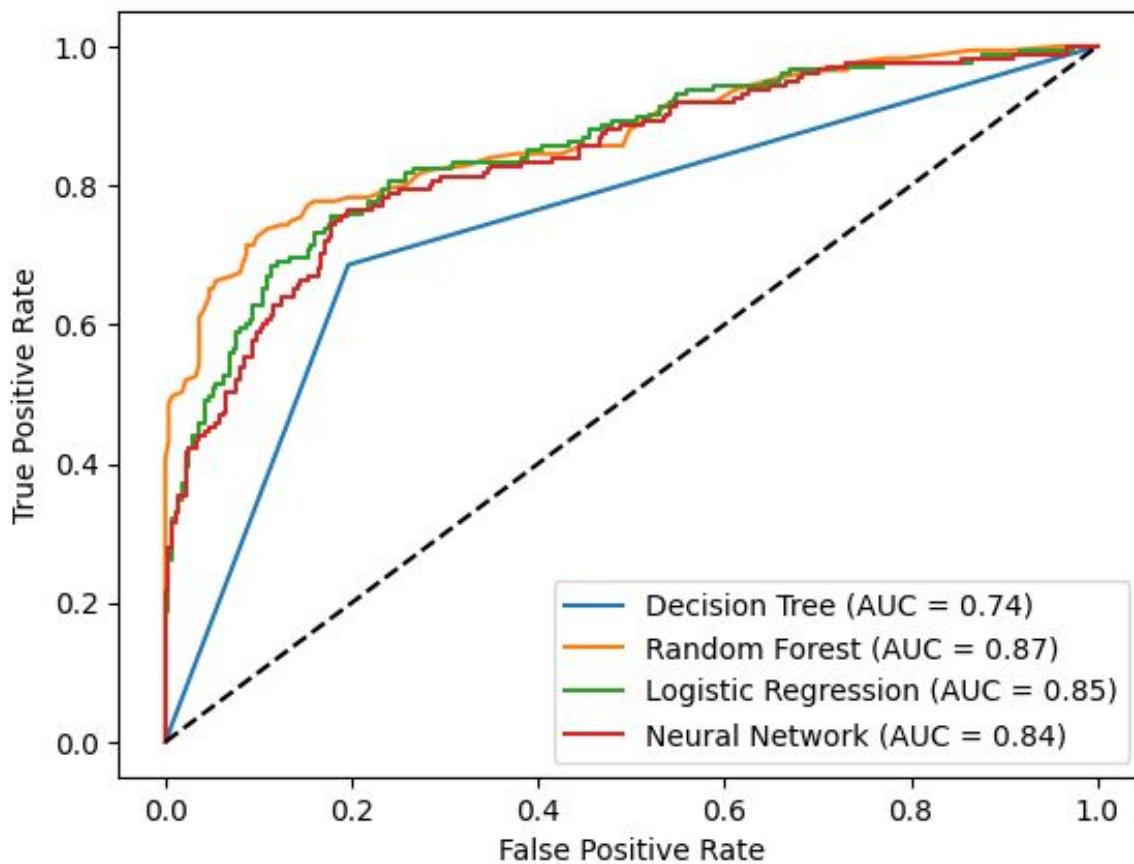
 Copy  Edit

```
from sklearn.metrics import roc_curve, roc_auc_score

for name, model in models.items():
    y_prob = model.predict_proba(X_test)[:, 1] # For probabilistic models
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc:.2f})')

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

ROC Curve



Conclusion: Model Evaluation and Next Steps

- **Evaluate multiple models** to choose the best one for your task.
- **Consider hyperparameter tuning** (e.g., using GridSearchCV) to optimize model performance.
- **Continue learning** about more advanced techniques (e.g., ensemble methods, deep learning).



CANCER CAN BE MANAGED.
THINK DIABETES! WITHOUT
INSULIN, DIABETICS WOULD
BE DEAD IN A WEEK.



Q&A Session:

- Any questions regarding the concepts covered today?
- Do you feel comfortable working with the models and evaluation metrics?