Module 3

Thursday, July 03, 2025 1:17 AM

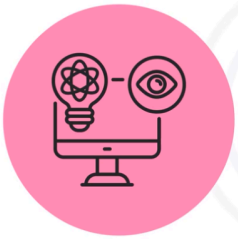
What is classification?

Supervised ML method

Uses fully trained models to predict labels on new data

Labels form a categorical variable with discrete values

What is supervised learning?



Understands data in context when answering a question

Ensures accuracy in predictions

Model adjusts the data to fit the algorithm and classifies it accordingly

Applications of classification

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	Yes
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	Yes
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	No
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	No
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	?

Churn prediction: If customer will discontinue service

Customer segmentation: Predict the category of a customer

Advertising: Predict if a customer will respond to a campaign

Use cases of classification

Loan default prediction

age	ed	employ	address	income	debtinc	creddebt	othdebt	default
41	3	17	12	176	9.3	11.359	5.009	1
27	1	10	6	31	17.3	1.362	4.001	0
40	1	15	14	55	5.5	0.856	2.169	0
41	1	15	14	120	2.9	2.659	0.821	0
24	2	2	0	28	17.3	1.787	3.057	1
41	2	5	5	25	10.2	0.393	2.157	0
39	1	20	9	67	30.6	3.834	16.668	0
43	1	12	11	38	3.6	0.129	1.239	0
24	1	3	4	19	24.4	1.358	3.278	1
36	1	0	13	25	19.7	2.778	2.147	0

Model Training

age	ed	employ	address	income	debtinc	creddebt	othdebt	default
37	2	16	10	130	9.3	10.23	3.21	

Trained Classifier

Prediction

Use cases of classification

Multiclass drug prescription

Age	Sex	BP	Cholesterol	Na	K	Drug
23	F	HIGH	HIGH	0.793	0.031	drugY
47	M	LOW	HIGH	0.739	0.056	drugC
47	M	LOW	HIGH	0.697	0.069	drugC
28	F	NORMAL	HIGH	0.564	0.072	drugX
61	F	LOW	HIGH	0.559	0.031	drugY
22	F	NORMAL	HIGH	0.677	0.079	drugX
49	F	NORMAL	HIGH	0.79	0.049	drugY
41	M	LOW	HIGH	0.767	0.069	drugC
60	M	NORMAL	HIGH	0.777	0.051	drugY
43	M	LOW	NORMAL	0.526	0.027	drugY

Categorical Variable

Modeling

Classifier

Age	Sex	BP	Cholesterol	Na	K	Drug
36	F	LOW	HIGH	0.697	0.069	DrugX

Prediction

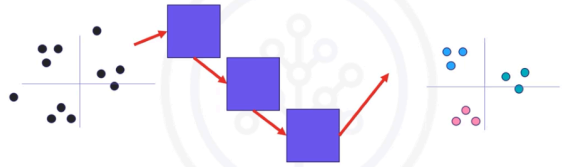
Predicted Labels

Classification algorithms

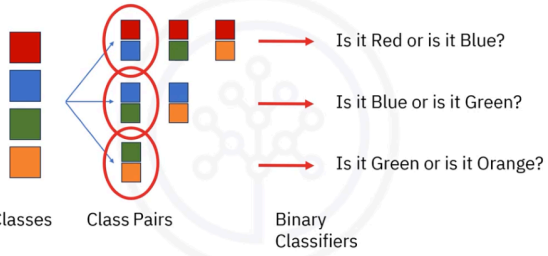
One-versus-all strategy

- **Binary classifier**: One for each class label
- Assigned a single label that defines target class
- **Task**: Binary prediction for every data point for a one-versus-the-rest classifier
- **K-classes** = K binary classifiers

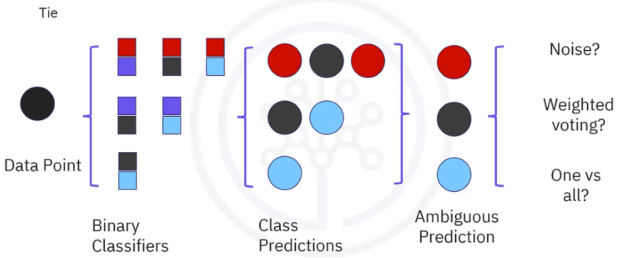
One-versus-all strategy



One-versus-one strategy

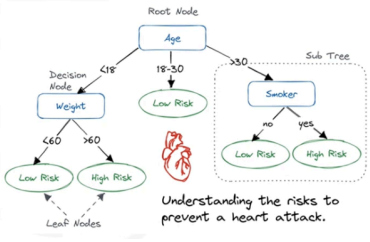


One-versus-one strategy



Decision tree

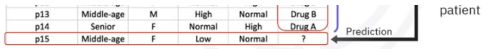
- Each internal node corresponds to a test
- Each branch corresponds to the result of the test
- Each terminal, or leaf node, assigns its data to a class



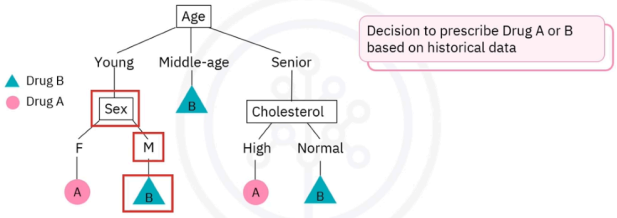
How to build a decision tree?

Patient ID	Age	Sex	BP	Cholesterol	Drug
p1	Young	F	High	Normal	Drug A
p2	Young	F	High	High	Drug A
p3	Middle-age	F	High	Normal	Drug B
p4	Senior	F	Normal	Normal	Drug B
p5	Senior	M	Low	Normal	Drug B
p6	Senior	M	Low	High	Drug A
p7	Middle-age	M	Low	High	Drug B
p8	Young	F	Normal	Normal	Drug A
p9	Young	M	Low	Normal	Drug B
p10	Senior	M	Normal	Normal	Drug B
p11	Young	M	Normal	High	Drug B
p12	Middle-age	F	Normal	High	Drug B

- Consider features of a data set
- Example in medical study: Age, sex, blood pressure, and cholesterol
- Use training part of data set to build decision tree
- Use decision tree to predict class of unknown

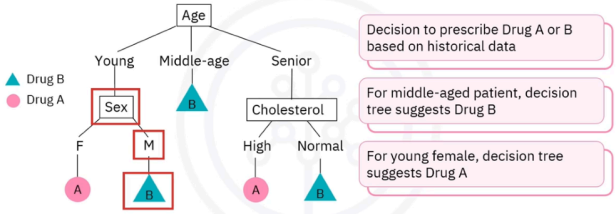


Patient classifier example



Decision to prescribe Drug A or B based on historical data

Patient classifier example

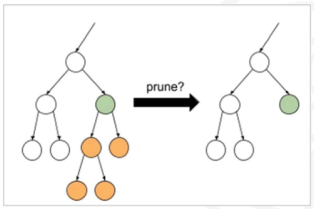


Decision to prescribe Drug A or B based on historical data

For middle-aged patient, decision tree suggests Drug B

For young female, decision tree suggests Drug A

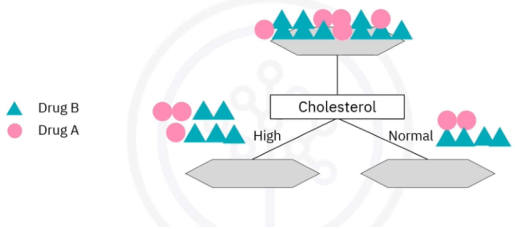
Tree pruning



Stop growing the tree when:

- Maximum tree depth is reached
- Minimum number of data points in a node has been exceeded
- Minimum number of samples in a leaf has been exceeded
- Decision tree has reached maximum number of leaf nodes

Pruning decision tree example



Module 3 Summary and Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

- Classification is a supervised machine learning method used to predict labels on new data with applications in churn prediction, customer segmentation, loan default prediction, and multiclass drug prescriptions.
- Binary classifiers can be extended to multiclass classification using one-versus-all or one-versus-one strategies.
- A decision tree classifies data by testing features at each node, branching based on test results, and assigning classes at leaf nodes.
- Decision tree training involves selecting features that best split the data and pruning the tree to avoid overfitting.
- Information gain and Gini impurity are used to measure the quality of splits in decision trees.
- Regression trees are similar to decision trees but predict continuous values by recursively splitting data to maximize information gain.
- Mean Squared Error (MSE) is used to measure split quality in regression trees.
- K-Nearest Neighbors (k-NN) is a supervised algorithm used for classification and regression by assigning labels based on the closest labeled data points.

- To optimize k-NN, test various k values and measure accuracy, considering class distribution and feature relevance.
- Support Vector Machines (SVM) build classifiers by finding a hyperplane that maximizes the margin between two classes, effective in high-dimensional spaces but sensitive to noise and large datasets.
- The bias-variance tradeoff affects model accuracy, and methods such as bagging, boosting, and random forests help manage bias and variance to improve model performance.
- Random forests use bagging to train multiple decision trees on bootstrapped data, improving accuracy by reducing variance.

Cheat Sheet: Building Supervised Learning Models

Common supervised learning models

Process Name	Brief Description	Code Syntax

One vs One classifier (using logistic regression)	**Process:** This method trains one classifier for each pair of classes.
	Key hyperparameters:
	- `estimator`: Base classifier (e.g., logistic regression)
	Pros: Can work well for small datasets.
	Cons: Computationally expensive for large datasets.
	Common applications: Multiclass classification problems where the number of classes is relatively small.
		`from sklearn.multiclass import OneVsOneClassifier`
		`from sklearn.linear_model import LogisticRegression`
		`model = OneVsOneClassifier(LogisticRegression())`

One vs All classifier (using logistic regression)	**Process:** Trains one classifier per class, where each classifier distinguishes between one class and the rest.
	Key hyperparameters:
	- `estimator`: Base classifier (e.g., Logistic Regression)
	- `multi_class`: Strategy to handle multiclass classification (`ovr`)
	Pros: Simpler and more scalable than One vs One.
	Cons: Less accurate for highly imbalanced classes.
	Common applications: Common in multiclass classification problems such as image classification.
		`from sklearn.multiclass import OneVsRestClassifier`
		`from sklearn.linear_model import LogisticRegression`
		`model = OneVsRestClassifier(LogisticRegression())`
		
		or
		
		`from sklearn.linear_model import LogisticRegression`
		`model_ova = LogisticRegression(multi_class='ovr')`

Decision tree classifier	**Process:** A tree-based classifier that splits data into smaller subsets based on feature values. **Key hyperparameters:** - `max_depth`: Maximum depth of the tree **Pros:** Easy to interpret and	`from sklearn.tree import DecisionTreeClassifier` `model = DecisionTreeClassifier(max_depth=5)`

	visualize. **Cons:** Prone to overfitting if not pruned properly. **Common applications:** Classification tasks, such as credit risk assessment.	
Decision tree regressor	**Process:** Similar to the decision tree classifier, but used for regression tasks to predict continuous values. **Key hyperparameters:** - `max_depth`: Maximum depth of the tree **Pros:** Easy to interpret, handles nonlinear data. **Cons:** Can overfit and perform poorly on noisy data. **Common applications:** Regression tasks, such as predicting housing prices.	`from sklearn.tree import DecisionTreeRegressor` `model = DecisionTreeRegressor(max_depth=5)`
Linear SVM classifier	**Process:** A linear classifier that finds the optimal hyperplane separating classes with a maximum margin. **Key hyperparameters:** - `C`: Regularization parameter - `kernel`: Type of kernel function (`linear`, `poly`, `rbf`, etc.) - `gamma`: Kernel coefficient (only for `rbf`, `poly`, etc.) **Pros:** Effective for high-dimensional spaces. **Cons:** Not ideal for nonlinear problems without kernel tricks. **Common applications:** Text classification and image recognition.	`from sklearn.svm import SVC` `model = SVC(kernel='linear', C=1.0)`
K-nearest neighbors classifier	**Process:** Classifies data based on the majority class of its nearest neighbors. **Key hyperparameters:** - `n_neighbors`: Number of neighbors to use - `weights`: Weight function used in prediction (`uniform` or `distance`) - `algorithm`: Algorithm used to compute the nearest neighbors (`auto`, `ball_tree`, `kd_tree`, `brute`) **Pros:** Simple and effective for small datasets. **Cons:** Computationally expensive as the dataset grows. **Common applications:** Recommendation systems, image recognition.	`from sklearn.neighbors import KNeighborsClassifier` `model = KNeighborsClassifier(n_neighbors=5, weights='uniform')`
Random Forest regressor	**Process:** An ensemble method using multiple decision trees to improve accuracy and reduce overfitting. **Key hyperparameters:** - `n_estimators`: Number of trees in the forest - `max_depth`: Maximum depth of each tree **Pros:** Less prone to overfitting than individual decision trees. **Cons:** Model complexity increases with the number of trees. **Common applications:** Regression tasks such as predicting sales or stock prices.	`from sklearn.ensemble import RandomForestRegressor` `model = RandomForestRegressor(n_estimators=100, max_depth=5)`

XGBoost regressor	**Process:** A gradient boosting method that builds trees sequentially to correct errors from previous trees. **Key hyperparameters:** - `n_estimators`: Number of boosting rounds - `learning_rate`: Step size to improve accuracy - `max_depth`: Maximum depth of each tree **Pros:** High accuracy and works well with large datasets. **Cons:** Computationally intensive, complex to tune. **Common applications:** Predictive modeling, especially in Kaggle competitions.	`import xgboost as xgb` `model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5)`

Associated functions used

Method Name	Brief Description	Code Syntax

OneHotEncoder	Transforms categorical features into a one-hot encoded matrix.	`from sklearn.preprocessing import OneHotEncoder`
		`encoder = OneHotEncoder(sparse=False)`
		`encoded_data = encoder.fit_transform(categorical_data)`

accuracy_score	Computes the accuracy of a classifier by comparing predicted and true labels.	`from sklearn.metrics import accuracy_score` `accuracy = accuracy_score(y_true, y_pred)`

LabelEncoder	Encodes labels (target variable) into numeric format.	`from sklearn.preprocessing import LabelEncoder`
		`encoder = LabelEncoder()`
		`encoded_labels = encoder.fit_transform(labels)`

plot_tree	Plots a decision tree model for visualization.	`from sklearn.tree import plot_tree plot_tree(model, max_depth=3, filled=True)`
normalize	Scales each feature to have zero mean and unit variance (standardization).	`from sklearn.preprocessing import normalize normalized_data = normalize(data, norm='l2')`
compute_sample_weight	Computes sample weights for imbalanced datasets.	`from sklearn.utils.class_weight import compute_sample_weight weights = compute_sample_weight(class_weight='balanced', y=y)`
roc_auc_score	Computes the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) for binary classification models.	`from sklearn.metrics import roc_auc_score auc = roc_auc_score(y_true, y_score)`