ML- WEEK - 8

1. Implement Decision Tree Classifier (ID3) on IRIS dataset and tabulate the results. (reference material for Decision Tree Classifier using ID3 was already shared)
2. Implement SVM classifier on IRIS dataset and tabulate the results
3. Compare the performance of Decision Tree with SVM classifiers on IRIS dataset.
4. Trying different techniques (can try on synthetic data) to avoid overfitting and improve generalizability of classifier models:
   - Applying different Regularization techniques: L1 (Lasso), L2 (Ridge), Dropout, and Early Stopping etc., (see references including user guide for scikit learn).
   - Visualize the training and test/validation errors using *learning_curve*, *LearningCurveDisplay* of scikit learn library. (see references including user guide for scikit learn).

****

Ex: code snippet for generating synthetic dataset for classification

```
from sklearn.datasets import make_classification

# make_classification: Complex dataset with noisy features
X_class, y_class = make_classification(
    n_samples=100, n_features=2, n_informative=2,
    n_redundant=0, n_clusters_per_class=1,
    class_sep=1.0, random_state=42
)
#Alternatively
X, y = make_classification(n_samples=100, n_features=15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Note:

There are two ways to generate synthetic data: make_classification(), make_blobs()

- make_classification is for testing classifiers; make_blobs is for testing clustering algorithms like K-Means.
- make_classification allows defining n_informative, n_redundant, and n_repeated features. make_blobs simply generates data around centers.
- make_classification can produce non-convex, overlapping, and complex boundaries, whereas make_blobs produces spherical (isotropic) clusters

****

Some useful code snippets for Exercise.No. 4

```
from sklearn.model_selection import learning_curve
# The learning_curve function returns training set sizes, training scores, and
cross-validation scores (requires matplotlib to plot curves)
train_sizes, train_scores, test_scores = learning_curve(
    estimator,
    X,
    y,
    cv=5,  # 5-fold cross-validation
    scoring='accuracy',
    n_jobs=-1, # Use all available CPU cores
```

```
    train_sizes=np.linspace(0.1, 1.0, 10) # 10 different sizes from 10% to
100% of data
)
#Alternatively using LearningCurveDisplay (avoids the need for matplotlib)

from sklearn.model_selection import LearningCurveDisplay
display = LearningCurveDisplay.from_estimator(
    estimator, X, y, ax=ax, cv=5, n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 5), scoring='accuracy'
)
**************************************************************************
```

Some reference for Exercise No. 4

Key Indicators of Overfitting:

- High training accuracy but low validation/test accuracy.
- Validation loss increases while training loss decreases

Some common techniques to avoid overfitting:

1. Data-Level Techniques

- **Increase Data Size:** Gather more data to help the model learn general patterns rather than memorizing noise.
- **Data Augmentation:** Artificially increase the training set size by applying transformations (e.g., rotation, flipping, or adding noise).

2. Model & Training Techniques

- **Regularization (L1/L2):** Add a penalty term to the loss function to penalize large coefficients (weights), with L2 (Ridge) often preferred.
- **Early Stopping:** Monitor validation performance during training and halt training when validation error begins to increase, preventing the model from over-learning.
- **Reduce Complexity:** Simplify the model by reducing the number of layers or neurons in neural networks, or limiting tree depth in decision trees.
- **Dropout:** In neural networks, randomly deactivate neurons during training, forcing the network to learn more robust features

3. Evaluation & Feature Techniques

Cross-Validation: Use Use K-fold cross-validation to ensure the model performs consistently across different data splits.

- **Feature Selection:** Remove irrelevant or redundant features to simplify the data representation.
- **Ensemble Methods:** Combine predictions from multiple models (e.g., Random Forest, Bagging) to improve generalization