Artificial Intelligence
Chapter 3 - Supervised Learning Algorithms

Dr. Adnan Ismail Al-Sulaifanie

Department of Electrical and Computer Engineering
University of Duhok, Iraq

Outline

- ➤ Linear regression algorithms.
- ➤ Polynomial regression
- ➤ Support vector machine
- ➤ K-nearest neighbors.
- ➤ Decision tree
- ➤ Random Forest
- ➤ Hyper-parameters tuning

References

1. Jason Brownlee, **Data Preparation for Machine Learning**, 2020
2. Ashwin Pajankar, **Hands-on Machine Learning with Python**, 2022
3. Benjamin Johnston, **Applied Supervised Learning with Python**, 2019
4. scikit-learn user guide

Contents

1 Supervised Learning Algorithms

➤ Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time.

➤ Supervised learning can be separated into regression and classification algorithms.

➤ Common supervised learning algorithms:

1. Linear Regression models – OLS, Ridge, Lasso, ...
2. Polynomial Regression
3. Logistic Regression
4. Support Vector Machine (SVM)
5. Decision Trees (DT)
6. Random Forest (RF)
7. K-nearest neighbours (kNN)
8. ...

2 Linear Regression

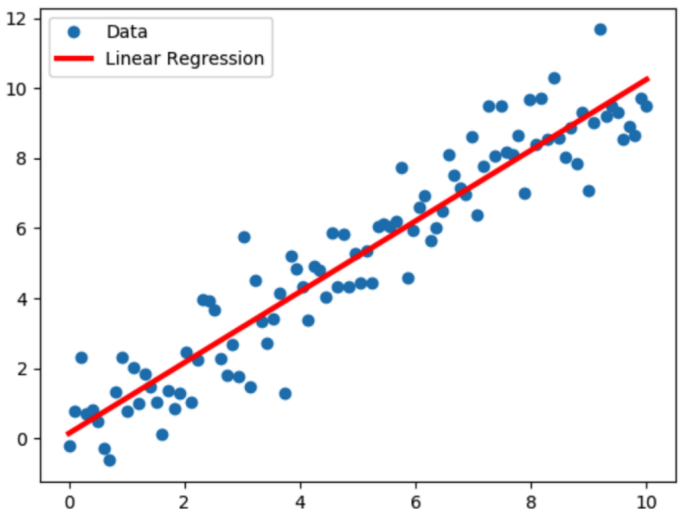
2.1 Ordinary Least Squares (OLS)

➤ Linear regression assumes a linear connection between X and y. It employs a regression line

$z = \boldsymbol{W}.\boldsymbol{X} = w_0 + w_1\, x_1 + w_2\, x_2 + ... + w_f\, x_f$

$yp = \varphi(z) = z$ linear activation function

➤ A gradient descent is used to solve this problem by minimizing $||wX - y||_2^2$



- The linear regression model can be simple (single feature) or complex (multiple features).
- Model's complexity is defined by the number of features, which determines the number of model parameters.
- Linear regression procedure:

1. Load or create dataset

   ```
   from sklearn import datasets, linear_model
   d = datasets.load_diabetes()
   X = d.data
   y = d.target
   ```

   or create your own dataset

   ```
   X, y,_ = datasets.make_regression(n_samples = 500, n_features = 2, n_informative = 1, noise = 10, random_state = 42)
   ```

2. Scale features if it is necessary

   ```
   from sklearn.preprocessing import StandardScaler
   scaler = StandardScaler()
   X = scaler.fit_transform(X)
   ```

3. If the number of features is large you can apply feature selection or dimensionality reduction
4. Split dataset

   ```
   from sklearn.model_selection import train_test_split
   xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.3, random_state = 0)
   ```

5. Build, fit and evaluate the model.

   ```
   from sklearn.linear_model import LinearRegression
   model = LinearRegression()
   model.fit(xtrain, ytrain)
   ```

6. Predict output using test set

   ```
   yp = model.predict(xtest)
   ```

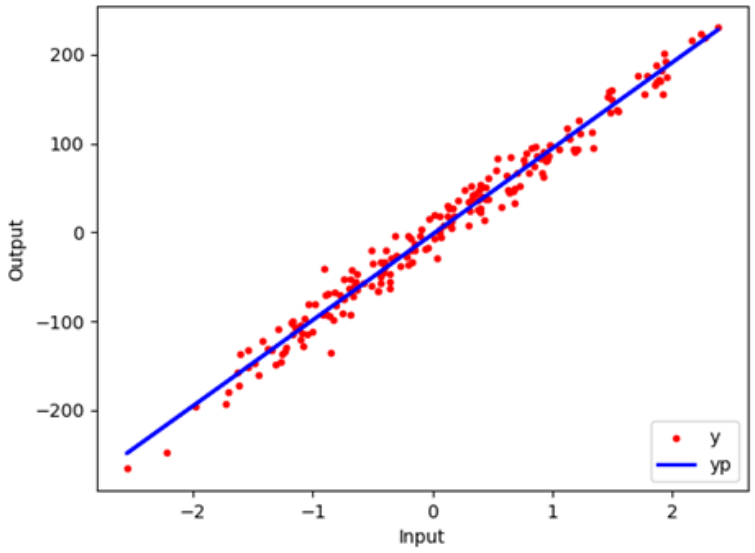
7. Evaluate performance of the model.

   ```
   from sklearn import metrics
   mse = metrics.mean_squared_error(ytest,yp)
   abs_err = metrics.mean_absolute_error(ytest, yp)
   ```

Example 1 - the following code shows an implementation of simple linear regression.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error
```

```
X, y, _ = datasets.make_regression(n_samples=200, n_features=1, n_informative=1, noise=15,
coef=True, random_state=0)

model = linear_model.LinearRegression()
model.fit(X, y)
yp = model.predict(X)
mse = mean_squared_error(y, yp)

print('W = ', model.coef_, ' W0 = ', model.intercept_)
print('MSE = ', mse)

plt.scatter(X, y, color="red", marker=".", label="y")
plt.plot(X, yp, color='blue', linewidth = 2, label = 'yp')
plt.legend(loc="lower right")
plt.xlabel("Input")
plt.ylabel("Output")
plt.show()
```



Example 2 - the diabetes dataset contains 10 features for each person. Design ML prediction model using linear regression.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y = True)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3, random_state=42)

model = LinearRegression()
```

```
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
mse = mean_squared_error(ytest, yp)

print('MSE = ', mse)
print('x-size = ', X.data.shape, ' xtrain-size = ', xtrain.shape, ' xtest-size = ', xtest.shape)
print('W = ', model.coef_)
print('W0 = ', model.intercept_)
```

```
MSE = 2821.750981001311
x-size = (442, 10)    xtrain-size = (309, 10)    xtest-size = (133, 10)
W = [ 29.254 -261.706 546.3 388.398 -901.96 506.763 121.154 288.035 659.269 41.377]
W0 = 151.00821291456543
```

Homework 1 - for the following data, determine the coefficients of linear regression model

```
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

y = np.array([6 8 9 11])
```

2.2 Regularization

- **Multicollinearity Issue –** When features are highly correlated, Ordinary Least Squares (OLS) may produce inflated weights, making the model sensitive to small changes in input data and prone to overfitting.
- **Regularization Solution –** To address this, regularization can be applied to penalize large weights by adding a penalty term to the loss function. This helps in shrinking model parameters and discourages overfitting.
- Types of Regularization:
  1. L2 Regularization – Adds the sum of the squared weights to the loss function.
  2. L1 Regularization – Adds the sum of the absolute weights to the loss function.
- While regularization can improve model performance by reducing overfitting, it may not always enhance performance in every scenario.

2.2.1 L2 Regularization

- L2 regularization adds a penalty term to the cost function to keep the magnitude of weights small.
- **Effect –** It prevents any single weight from becoming too large, thereby reducing the impact of correlated features and making the model more robust.
- It doesn't reduce the weights to zero, so we will observe that they all have non-zero values.

- **Formula –** The penalty term is $\sum {w_i}^2$
- Ridge Regression uses L2 regularization to mitigate overfitting and multicollinearity.

  Cost Function – $L(y, yp) = ||WX - y||_2^2 + \alpha||w||_2$

  where α is the regularization parameter.
- Impact of α
  - **Small α –** Minimal regularization effect, similar to Ordinary Least Squares (OLS).
  - **Large α –** Increased shrinkage of coefficients, potentially leading to underfitting.
- Benefits of Ridge regression
  - Reduces the impact of multicollinearity.
  - Keeps all features in the model, assuming all are relevant.
  - Requires standardization of input features.
- Drawbacks of Ridge regression
  - Does not perform feature selection.
  - May underfit if α is too large.
- Ridge regression is effective in distributing the impact of correlated features, preventing the model from relying too heavily on any single variable.
- It is suitable when all features are assumed to be relevant.
- For feature selection, other techniques like Lasso regression may be more appropriate.

2.2.2 L1 Regularization

- **Mechanism –** Adds the absolute value of the weights $\alpha \sum |w|$to the loss function, shrinking model weights toward zero and potentially setting some to zero.
- **LASSO –** Least Absolute Shrinkage Selection Operator.
- LASSO uses L1 regularization. It incorporates a penalty term into the OLS objective function:

  Cost = $||WX - y||_2^2 + \alpha||w||_1$

  α controls the degree of regularization; larger values result in more features being shrunk to zero.
- It acts as a feature selector by reducing the weights of less important features to zero, effectively removing them from the model.
- Features require to be standardized.

➤ **Limitations –** Cannot solve the problem of correlated features. It fails to exclude one of them. ElasticNet can address this issue.

Example 3 - python code compare the performance of OLS, Ridge, and Lasso

```
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression, Ridge, Lasso
import numpy as np
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y=True)
scaler = StandardScaler().fit(X)
XS = scaler.transform(X)

model = LinearRegression()
model.fit(X,y)
yp = model.predict(X)
MSE = mean_squared_error(y, yp)

print('OLS')
print('========')
print('Linear Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
print('W = ', model.coef_)

model.fit(XS,y)
yp = model.predict(XS)
MSE = mean_squared_error(y, yp)
print('Normalized Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
print('W = ', model.coef_)

model = Ridge(alpha = 0.5)
model.fit(X,y)
yp = model.predict(X)
MSE = mean_squared_error(y, yp)

print('RIDGE')
print('========')
print('Linear Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
print('W = ', model.coef_)

model.fit(XS,y)
yp = model.predict(XS)
MSE = mean_squared_error(y, yp)
print('Normalized Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
```

```
print('W = ', model.coef_)

model = Lasso(alpha = 0.5)
model.fit(X,y)
yp = model.predict(X)
MSE = mean_squared_error(y, yp)

print('LASSO')
print('========')
print('Linear Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
print('W = ', model.coef_)

model.fit(XS,y)
yp = model.predict(XS)
MSE = mean_squared_error(y, yp)
print('Normalized Sum of weights = ', sum(model.coef_) ,' MSE = ', MSE)
print('W = ', model.coef_)
```

```
OLS
========
Linear — Sum of weights = 1375.9753353877677     MSE = 2859.6963475867506
W = [ -10.01 -239.82 519.85 324.38 -792.18 476.74 101.04 177.06 751.27 67.63]
Normalized — Sum of weights = 65.44847241969218     MSE = 2859.6963475867506
W = [ -0.48 -11.41 24.73 15.43 -37.68 22.68 4.81 8.42 35.73 3.22]

RIDGE
========
Linear — Sum of weights = 830.2121773364549     MSE = 3039.0615529336997
W = [ 20.14 -131.24 383.48 244.84 -15.19 -58.34 -174.84 121.98 328.5 110.89]
Normalized — Sum of weights = 62.42468404934038     MSE = 2860.0032441794497
W = [ -0.45 -11.37 24.75 15.4 -33.44 19.31 2.94 7.92 34.12 3.24]

LASSO
========
Linear — Sum of weights = 957.2496258584583     MSE = 3230.3583272339656
W = [ 0. -0. 471.04 136.5 -0. -0. -58.32 0. 408.02 0. ]
Normalized — Sum of weights = 44.29789662886187     MSE = 2876.3799168593073
W = [ -0. -10.29 24.98 14.67 -7.76 -0. -8.45 3.28 24.95 2.91]
```

Homework 2 - modify previous example for Ridge regression to select best value for α

2.3 Hybrid Regularization

➤ Combines both L1 and L2 regularization.

- Elastic Net
- It allows a balance between feature selection and model's simplicity.

2.4 Polynomial Regression

- Polynomial regression is a machine learning technique used to model non-linear relationships between outcome and predictor variables.
- Polynomial regression is a kind of linear regression in which the relationship between X and y is modeled as the nth degree of the polynomial. This is done to look for the best way of drawing a line using data points.
- PolynomialFeatures is a preprocessing technique that generates polynomial combinations of features, enabling algorithms to capture nonlinear relationships in the data. It takes the original feature matrix and transforms it into a new matrix containing all possible polynomial combinations of the features up to a specified degree.
- Polynomial Regression is implemented by adding higher-degree terms of the predictor variable to the equation, such as x^2, x^3, etc.

  $(x)^2 \longmapsto 1, x, x^2$

  $(x_1, x_2)^2 \longmapsto x_1, x_2, x_1 * x_2, {x_1}^2, {x_2}^2$

  The linear regression is expressed as shown below:

  $yp(w, x) = w_0 + w_1\ x_1 + +w_2\ x_2$

  If the PolynomialFeatures is applied, then

  $yp(w, x) = w_0 + w_1\ x_1 + w_2\ x_2 + w_3\ x_1\ x_2 + w_4\ x_1^2 + w_1\ x_2^2$

  The model still linear, to see this, imagine creating a new set of features

  $z = [x_1, x_2, x_1\ x_2, x_1^2, x_2^2]$

  With this re-labeling of the data, our problem can be written

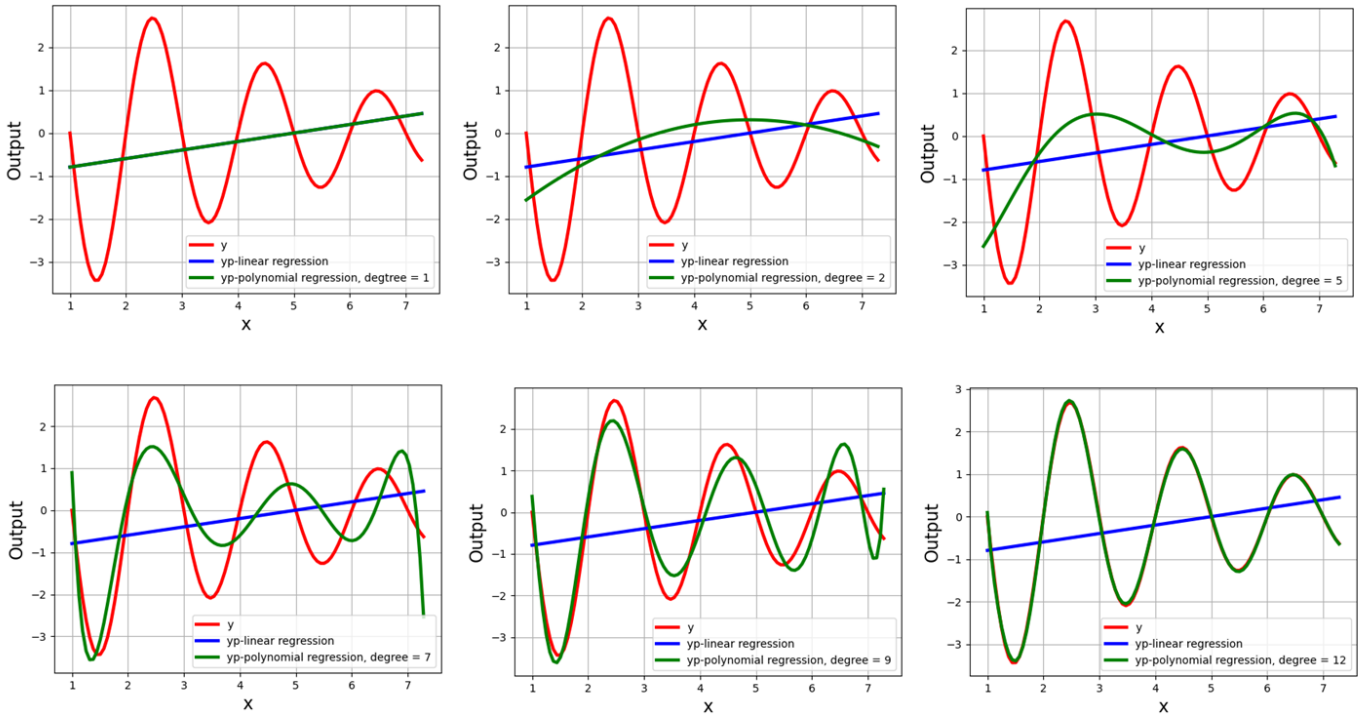
  $yp(w, z) = w_0 + w_1\ z_1 + w_2\ z_2 + w_3\ z_3 + w_4\ z_4 + w_5\ z_5$
- Polynomial regression can fit more complex data than simple linear regression, but it also has some drawbacks, such as overfitting and multicollinearity.

Example 4 - python code explains using polynomialFeatures to use LinearRegression for nonlinear relationship.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
```

```
x = 1 + np.linspace(0.0, 2 * np.pi, 100)
y = 5 * np.sin(x * np.pi) * np.exp(-x/4)

x = x.reshape(-1,1)
y = y.reshape(-1,1)

model = LinearRegression()
model.fit(x,y)
yp = model.predict(x)
mse = mean_squared_error(y, yp)
print('Degree = None MSE = ',mse)

deg = 1
poly = PolynomialFeatures(degree = deg)
xp = poly.fit_transform(x)
model.fit(xp,y)
yp2 = model.predict(xp)
mse = mean_squared_error(y, yp)
print('Degree = ',deg,' X-size = ', xp.shape' MSE = ',mse)

plt.plot(x, y, color="red", linewidth = 3, label="y")
plt.xlabel('x', fontsize = 16)
plt.ylabel("Output", fontsize = 16)
plt.plot(x, yp, color="blue", linewidth = 3, label = "yp-linear regression")
plt.plot(x, yp2, color="green", linewidth = 3, label = "yp-polynomial regression, degree = 1")

plt.legend(loc="lower right")
plt.grid() plt.show()
```

Degree	No. features	MSE
1	2	2.104
2	3	1.979
3	4	1.842
4	5	1.776
5	6	1.731
6	7	1.654
7	8	0.831
8	9	0.631
9	10	0.209
10	11	0.053
11	12	0.02
12	13	0.001



Homework 3 - complete the following code to design best regression model for the following datasets.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error

x = np.array([2,3,4,5,6,7,7,8,9,11,12])
y = np.array([18,16,15,17,20,23,25,28,31,30,29])
```

3 Logistic Regression

- ➤ Logistic regression is a statistical method used in machine learning for binary classification, predicting whether an instance belongs to one of two classes.
- ➤ Despite its name, it is a classification algorithm, not a regression algorithm.
- ➤ The output is a class probability, indicating the likelihood that an instance belongs to a particular class.
- ➤ Logistic regression uses a linear combination of input features

  $z = \boldsymbol{W}.\boldsymbol{X} = w_0 + w_1.x_1 + w_1.x_1 + ... + w_f.x_f$

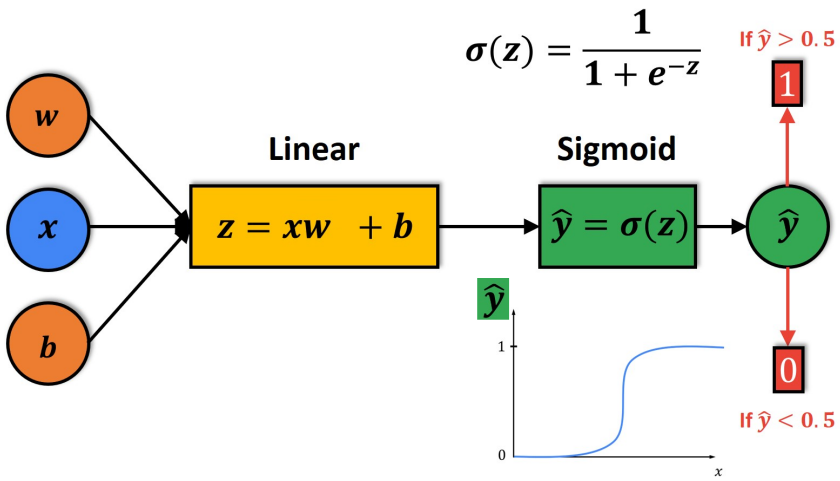
  This linear combination is passed through a sigmoid activation function to produce a probability

$$p = f(z) = \frac{1}{1 + e^{-z}}$$

A threshold (typically 0.5) is used to classify instances: if $p > 0.5$, the instance is classified as class 1; otherwise, it is class 0.

➤ The cost function used for optimization in logistic regression is Binary Cross-Entropy (BCE). BCE measures the performance of a classification model whose output is a probability value between 0 and 1.

$$\text{BCE} = \frac{-1}{N}\sum_{i=1}^{N}\left[(y_i * log(p_i)) + (1 - y_i) * log(1 - p_i)\right]$$



Example 5 - the following codes show how to implement logistic regression.

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

x = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

model = LogisticRegression(max_iter=1000)
model.fit(x, y)
yp = model.predict(x) score = accuracy_score(y,yp)
print('accuracy = ', score)
```

accuracy = 0.9166666666666666

Homework 4 - for previous codes, use confusion matrix and classification report to evaluate their performance.

Example 6 -

```
from sklearn.linear_model import LogisticRegression
```

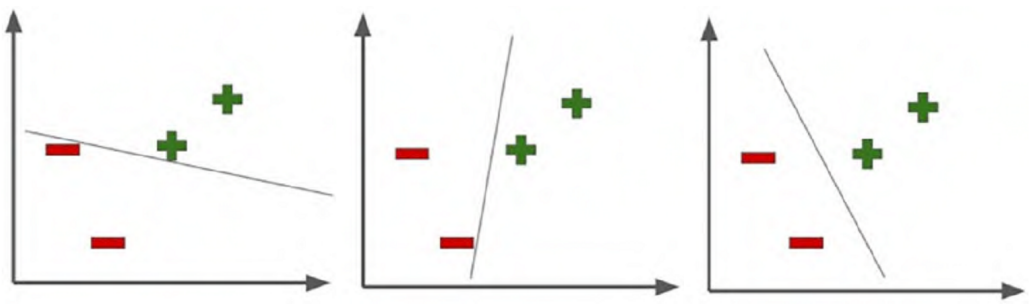
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_classes=2, random_state=42)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression(max_iter=1000)
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
score = accuracy_score(ytest,yp)
print('accuracy = ', score)
```

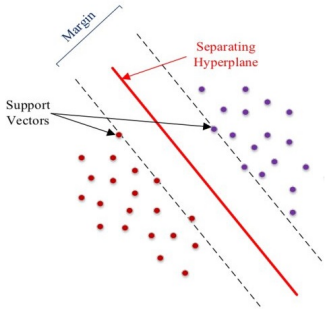
accuracy = 0.835

Homework 5 - show how you can improve the performance of previous model

4 Support Vector Machine (SVM)

- SVM is used to solve classification and regression problems. It is best suited for classification tasks.





- **Hyperplane – –**
  - Hyperplaneis a boundary that divides data into two classes.
  - Optimal hyperplane maximizes the margin between classes.
  - Dimension of the hyperplane depends on the number of features. In two-dimensional data, the hyperplane is a line. In three-dimensional space, it becomes a plane, and in higher dimensions, it extends accordingly.
- **Support Vectors – –** are the data points that are closest to the hyperplane and have a crucial role in determining the position and orientation of the hyperplane.

These data points influence the optimal placement of the hyperplane.

SVMs aim to find the hyperplane that maximizes the distance (margin) between classes, defined by support vectors.

- **Loss Function –** Uses hinge loss for penalty.
- **Margin –**
  - Margin is the distance between the support vectors and the hyperplane.
  - Margin Maximization – SVM aims to maximize this margin because a larger margin often leads to better generalization performance on unseen data.
- SVM finds application across various domains, including text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection.

4.1 SVM Regularization

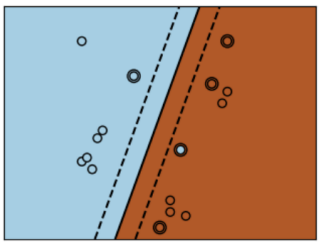
- Regularization in SVM is a technique used to prevent overfitting and improve the generalization of the model to unseen data.
- Regularization adds a penalty to the loss function.
- Regularization is controlled by a parameter often denoted as C.
- **C Parameter –** is the regularization parameter that controls the trade-off between the margin and the number of training errors.
- This parameter determines the trade-off between achieving a low training error and minimizing the model complexity.
- The parameter C controls the penalty for misclassified points.
- **Large C –**
  - The model tries to classify all training examples correctly.
  - This can result in a complex model with a small margin, which may overfit the training data.
- **Small C –**
  - A smaller C encourages a wider margin and allows some training points to be misclassified,
  - It leads to a simpler model with a larger margin.
  - This can improve the model's ability to generalize to new data
  - It may underfit if C is too small.

4.2 Nonlinear SVM

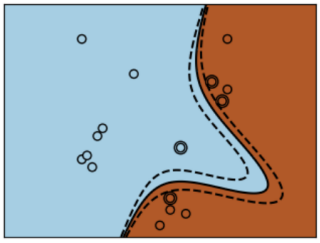
➤ Much of the data in real-world scenarios are not linearly separable, and that's where non-linear SVMs come into play.

➤ There are two main types of SVMs based on the nature of the decision boundary:

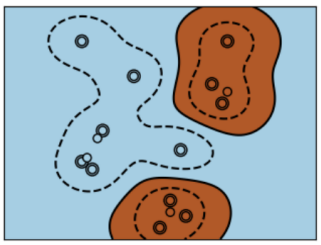
1. **Linear SVM – –**
   - ➢ It is used when the data are linearly separable.
     model = SVC(kernel = 'linear')
2. **Non-linear SVM – –**
   - ➢ It is used when the samples in dataset are not linearly separable
   - ➢ It uses kernel trick to implicitly map inputs into high-dimensional feature spaces, allowing them to perform non-linear classification.
   - ➢ The available options:
     (a) Polynomial kernel
     (b) Radial basis function kernel (also known as a Gaussian or RBF kernel)
     model = SVC(kernel='poly', degree = n)
     model = SVC(kernel='rbf')



svc = svm.SVC(kernel='linear')



svc = svm.SVC(kernel='poly', degree=3)



svc = svm.SVC(kernel='rbf')

Example 7 -

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score


breast = datasets.load_breast_cancer()
X = breast.data
y = breast.target
scaler = StandardScaler()
X = scaler.fit_transform(X)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = SVC(kernel = 'linear' , C = 1, gamma = 'scale')
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
train_accuracy = model.score(xtrain,ytrain)
test_accuracy = accuracy_score(ytest,yp)

print('Train acuuracy = ', train_accuracy * 100)
print('Test accuracy = ', test_accuracy * 100)
```

```
Train acuuracy = 98.68131868131869
Test accuracy = 95.6140350877193
```

Homework 6 - repeat previous code to fill the following table and answer the following questions:

C	Training accuracy	Test accuracy
0.1		
0.2		
0.4		
0.6		
0.8		
1.0		

1. How we can choose optimal value of C?
2. Do we need using other kernel functions?
3. What is the role of gamma parameter? List other options of this parameter.
4. What is the effect of standardization on the performance?

4.3 k-nearest neighbors (kNN)

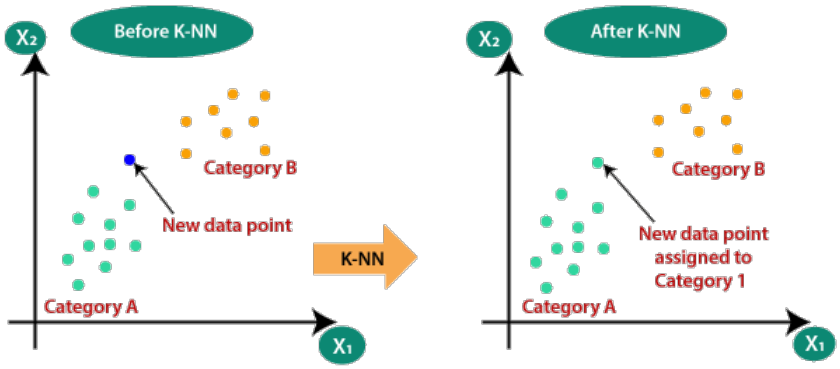
- ➤ The k-Nearest Neighbors (kNN) algorithm is a simple supervised learning technique used for both classification and regression.
- ➤ It is a lazy learner, storing the dataset and classifying new data points at the time of prediction.
- ➤ kNN is non-parametric and relies on the similarity between new and existing data points to categorize the new data.
- ➤ The algorithm works by finding the k closest neighbors to a new data point based on a distance metric, such as

  Euclidean distance $d(x, y) = \sqrt{\sum_{i=1}^{N_f}(x_i - y_i)^2}$

Manhattan distance $d(x, y) = \sum_{i=1}^{N_f} |x_i - y_i|$

KinKowski distance $d(x, y) = \left(\sum_{i=1}^{N_f} |x_i - y_i|^p \right)^{\frac{1}{p}}$

- The predicted output is determined by the majority vote (classification) or average of its k nearest neighbors (regression).
- **k** is a user-defined parameter representing the number of neighbors to consider. The value of k is a hyperparameter that can be tuned to improve the performance of the algorithm. It is recommended to choose an odd value for k to avoid ties in classification.



Example 8 - write python code to apply kNN algorithm on Iris dataset

```
from sklearn.neighbors import NearestNeighbors
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import numpy as np

iris = load_iris()
features = iris.data
target = iris.target

scaler = StandardScaler()
scaledFeatures = scaler.fit_transform(features)

model = NearestNeighbors(n_neighbors =5) model.fit(scaledFeatures)

xnew = np.array([5, 4, 1.5, 0.5]).reshape(1,-1)
scaled_xnew = scaler.transform(xnew)
distance, indices = model.kneighbors(scaled_xnew)

print('Indices = ', indices)
print('xnew = ', xnew)
print('Similar samples = ', features[indices])
print('Target = ', target[indices])
```

Indices = [[44 19 5 16 46]]
xnew = [[5. 4. 1.5 0.5]]

Similar samples = [[[5.1 3.8 1.9 0.4]
[5.1 3.8 1.5 0.3]
[5.4 3.9 1.7 0.4]
[5.4 3.9 1.3 0.4]
[5.1 3.8 1.6 0.2]]]

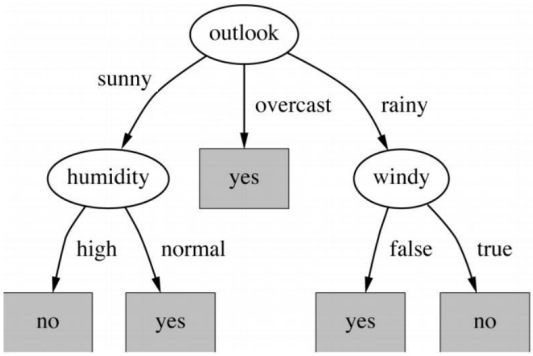
Target = [[0 0 0 0 0]]

Homework 7 - Answer the following questions

1. What is the effect of **n_neighbors** on the performance.
2. Estimate class of [6.9 3.1 6.0 2.1]
3. Show the effect of standardization on the performance.

5 Decision Tree (DT)

- ➤ DT is a supervised learning algorithm used for both classification and regression tasks.
- ➤ It is a flowchart-like structure where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.
- ➤ It's a tree-like model where an internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents the class label (for classification) or the predicted value (for regression).



- ➤ Decision trees are easy to understand, interpret, and visualize, making them a widely used tool in various domains.
- ➤ Splitting criterion

1. Information Gain (IG) = Entropy(S)- [(Weighted Avg) * Entropy(each feature)

2. Gini Index (GI) = $1 - \sum {p_i}^2$

➤ Procedure:

1. Select the Root Node – identify the best feature to split the dataset. The goal is to choose the feature that maximizes information gain.
2. Divide the dataset into subsets based on the chosen feature. Each subset represents a branch of the decision tree.
3. Create Branches – branch out to nodes representing the different values of the selected attribute.
4. For each branch, repeat the process by selecting the best attribute to split on until reaching leaf nodes.
5. Assign a class label to each leaf node based on the majority class of the instances in that node.

Example 9 - An athletic man engages in tennis every Sunday, depending on the current weather conditions. Develop a decision tree algorithm to formulate a predictive model for this activity

Outlook	Temperature	Humidity	Windy	Play Tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Entropy $E(P) = -\sum p_i * Log(p_i)$

Information gain of each feature $IG(S, feature) = E(S) - E(S, feature)$

$E(S) = E(yes, no) = E(\frac{9}{14}, \frac{5}{14}) = -\left[\frac{9}{14} Log_2(\frac{9}{14}) + \frac{5}{14} Log_2(\frac{5}{14})\right] = 0.94$

		yes	no	total
Outlook	Sunny	3	2	5

Overcast	4	0	4
Rain	2	3	5

Calculate weighted Entropy for each feature

$\boldsymbol{E(S, outlook) = \frac{5}{14}\ E(3,2) + \frac{4}{14}\ E(4,0) + \frac{5}{14}\ E(2,3)}$

$\boldsymbol{E(S, outlook) = \frac{5}{14}\left[\frac{-3}{5}\ Log_2(\frac{3}{5}) - \frac{2}{5}\ Log_2(\frac{2}{5})\right] + \frac{4}{14}\,[0] + \frac{5}{14}\left[\frac{-2}{5}\ Log_2(\frac{2}{5}) - \frac{3}{5}\ Log_2(\frac{3}{5})\right] = 0.693}$

$\boldsymbol{IG(S, outlook) = 0.94 - 0.693 = 0.247}$

		yes	no	total
Temperature	Hot	2	2	4
	Mild	4	2	6
	Cool	3	1	4

$\boldsymbol{E(S, temperature) = \frac{4}{14}\ E(2,2) + \frac{6}{14}\ E(4,2) + \frac{4}{14}\ E(3,1)}$

$\boldsymbol{E(S, temperature) = \frac{4}{14}\left[\frac{-2}{4}\ Log_2(\frac{2}{4}) - \frac{2}{4}\ Log_2(\frac{2}{4})\right] + \frac{6}{14}\left[\frac{-4}{6}\ Log_2(\frac{4}{6}) - \frac{2}{6}\ Log_2(\frac{2}{6})\right] +}$
$\boldsymbol{\frac{4}{14}\left[\frac{-3}{4}\ Log_2(\frac{3}{4}) - \frac{1}{4}\ Log_2(\frac{1}{4})\right] = 0.911}$

$\boldsymbol{IG(S, temperature) = 0.94 - 0.911 = 0.029}$

		yes	no	total
Humidity	High	3	4	7
	Normal	6	1	7

$\boldsymbol{E(S, humidity) = \frac{7}{14}\ E(3,4) + \frac{7}{14}\ E(6,1)}$

$\boldsymbol{E(S, humidity) = \frac{7}{14}\left[\frac{-3}{7}\ Log_2(\frac{3}{7}) - \frac{4}{7}\ Log_2(\frac{4}{7})\right] + \frac{7}{14}\left[\frac{-6}{7}\ Log_2(\frac{6}{7}) - \frac{1}{7}\ Log_2(\frac{1}{7})\right] = 0.788}$

$\boldsymbol{IG(S, humidity) = 0.94 - 0.788 = 0.152}$

		yes	no	total
Windy	Weak	6	2	8
	Strong	3	3	6

$\boldsymbol{E(S, windy) = \frac{8}{14}\ E(6,2) + \frac{6}{14}\ E(3,3)}$

$\boldsymbol{E(S, windy) = \frac{8}{14}\left[\frac{-6}{8}\ Log_2(\frac{6}{8}) - \frac{2}{8}\ Log_2(\frac{2}{8})\right] + \frac{6}{14}\left[\frac{-3}{6}\ Log_2(\frac{3}{6}) - \frac{3}{6}\ Log_2(\frac{3}{6})\right] = 0.892}$

$\boldsymbol{IG(S, windy) = 0.94 - 0.892 = 0.048}$

Now select the feature with largest IG. Here it is Outlook. So it forms the first node(root node) of the decision tree.

The root node has three branches.

Outlook	Temperature	Humidity	Windy	Play Tennis?
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

For each branch, we will compute the information gain of other features with respect to Outlook.

For sunny we compute IG for temperature, humidity, and wind. Also, we repeat the process for overcast and rain.

		yes	no	total
Temperature	hot	0	2	2
	cool	1	0	1
	mild	1	1	2

$$E(sunny) = -\frac{2}{5}\ Log_2(\frac{2}{5}) - \frac{3}{5}\ Log_2(\frac{3}{5}) = 0.97$$

$$E(sunny, temperature) = \frac{2}{5}\ E(0,2) + \frac{1}{5}\ E(1,0) + \frac{2}{5}\ E(1,1)$$

$$E(sunny, temperature) = \frac{2}{5} * [0] + \frac{1}{5} * [0] + \frac{2}{5} * \left[\frac{-2}{5}\ Log_2(\frac{1}{2}) - \frac{1}{2}\ Log_2(\frac{1}{2})\right] = 0.4$$

$$IG(sunny, temperature) = 0.97 - 0.4 = 0.57$$

$$IG(sunny, humidity) = 0.97 - 0 = 0.97$$

$$IG(sunny, windy) = 0.97 - 0.95 = 0.02$$

Select feature with largest IG, i.e. humidity. Then, test other features in branch.

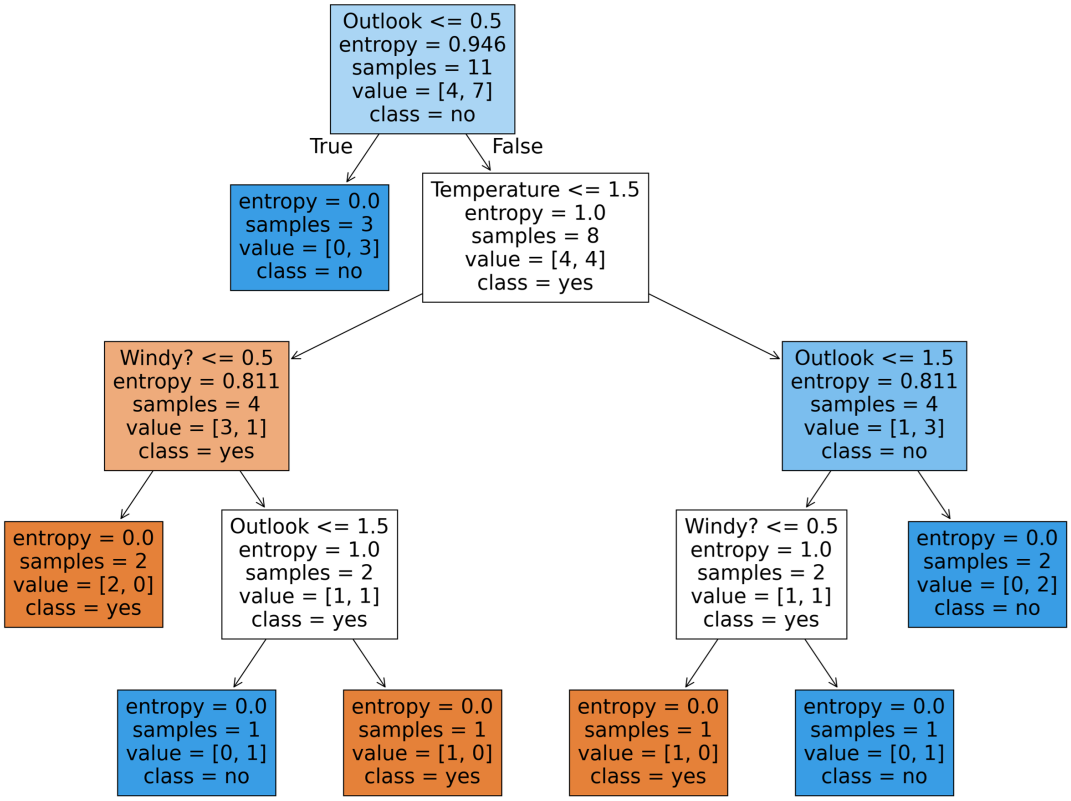
The process is repeated and stopped when the entropy ≈ 0

Example 10 -

import pandas as pd

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn import tree, preprocessing
import matplotlib.pyplot as plt

data =
'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain','Overcast', 'Sunny', 'Sunny', 'Rain',
'Sunny', 'Overcast', 'Overcast', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot',
'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'High'],

'Windy?': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak',
'Strong', 'Strong', 'Weak', 'Strong'],

'PlayTennis': ['No', 'No', 'Yes' , 'Yes', 'Yes', 'No','Yes', 'Yes' , 'No', 'Yes', 'Yes', 'Yes' , 'Yes', 'No']

df = pd.DataFrame(data)
le = preprocessing.LabelEncoder()
for column in df.columns: df[column] = le.fit_transform(df[column])


X = df.drop('PlayTennis', axis = 1)
y = df['PlayTennis']

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.20, random_state = 42)


model = DecisionTreeClassifier(criterion = 'entropy')
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
accuracy = sum(yp == ytest) / len(ytest)

print('Accuracy =', accuracy)

plt.rcParams['figure.dpi'] = 200
plt.rcParams['savefig.dpi'] = 200

plt.figure(figsize=(20,20))
tree.plot_tree(model, filled=True, feature_names= list(X.columns),class_names= ['yes', 'no'])
plt.show()
```

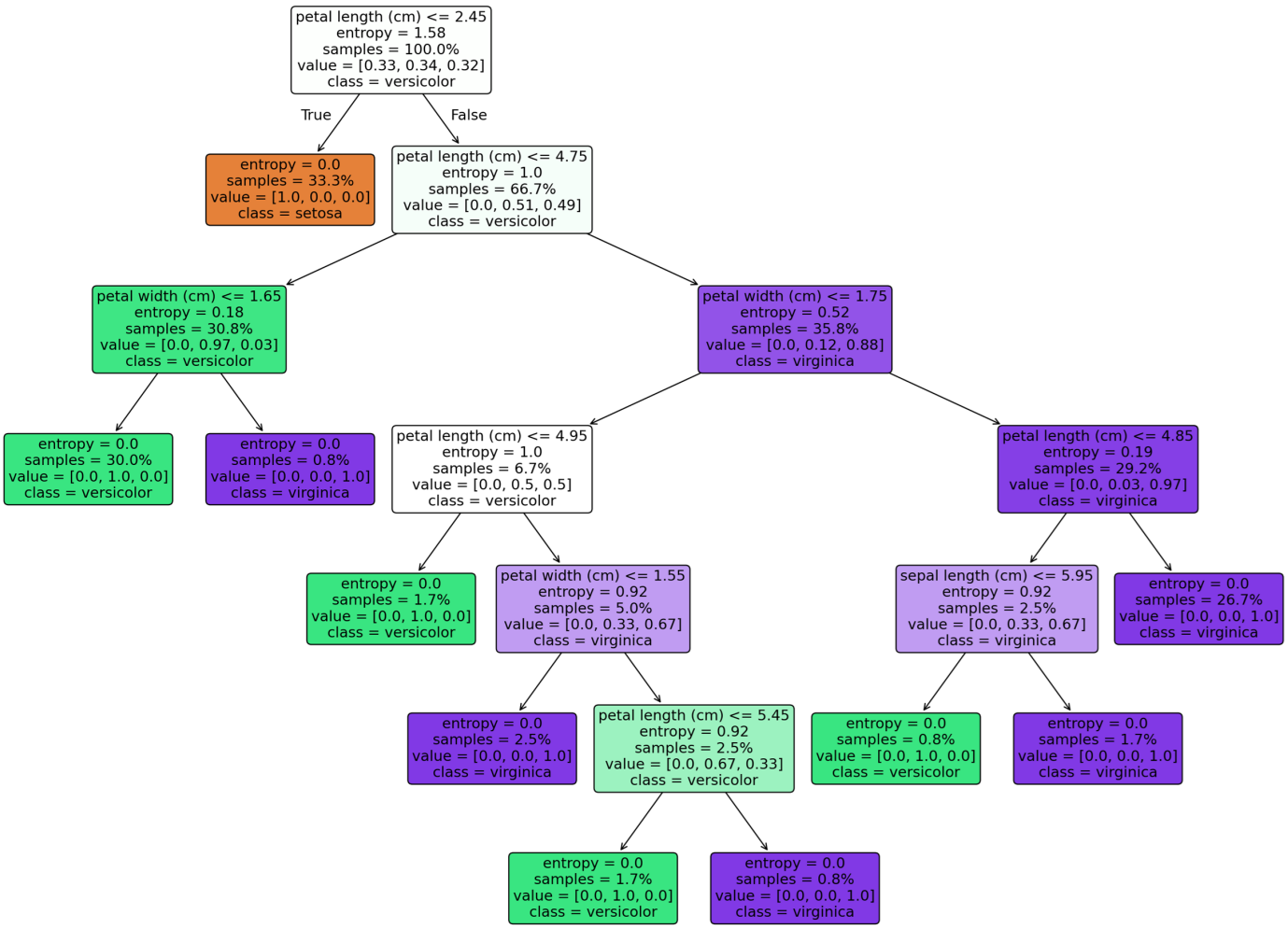
	Outlook	Temperature	Humidity	Windy?	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	Yes
8	Sunny	Cool	Normal	Weak	No
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No



Example 11 - write python code for decision tree classification of iris dataset.

```
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn import tree, preprocessing
import matplotlib.pyplot as plt

iris = load_iris()
```

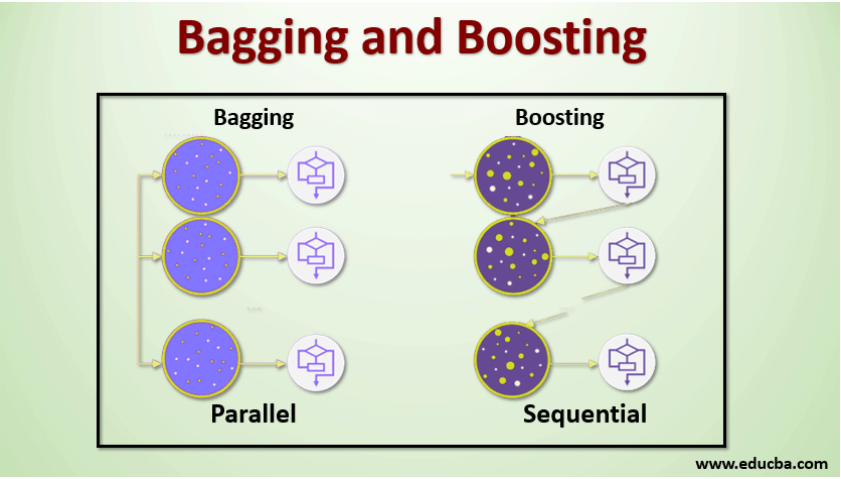
```
X = iris.data
y = iris.target
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.20, random_state = 42)


model = DecisionTreeClassifier(criterion = 'entropy')
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
accuracy = accuracy_score(ytest, yp)
print('Accuracy = ', accuracy)

plt.rcParams['figure.dpi'] = 200
plt.rcParams['savefig.dpi'] = 200
plt.figure(figsize=(20,15))
tree.plot_tree(model, filled = True, feature_names = iris.feature_names, class_names = iris.tar-
get_names, rounded = True, proportion = True, precision = 2)
plt.show()
```



6 Ensemble Learning

- Ensemble learning is a machine learning technique that combines multiple models to enhance overall performance and robustness.
- The primary goal is to create a more accurate and reliable model by leveraging the strengths of several individual models.
- Approaches:

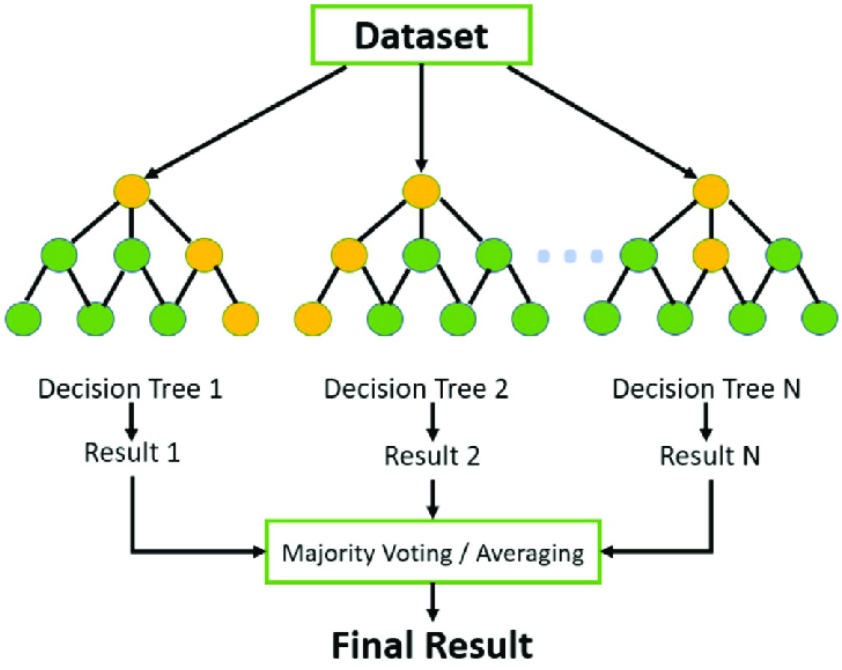
1. **Bagging –**
   - Multiple models are trained on different subsets of the data, and their predictions are combined.
   - Classification – each model votes for a class, and the final prediction is the class with the most votes.
   - Regression – the final prediction is the average of all individual model predictions. Also, weighted average can be used.
   - Models can be trained in parallel, making the process efficient.
   - Examples - random forest
2. **Boosting –**
   - Models are trained sequentially, with each new model attempting to correct the errors of the previous ones.
   - Each succeeding model depends on the performance of the previous model.
   - Examples – AdaBoost and Gradient Boosting.



6.1 Random Forest (RF)

- Random Forest is a popular machine learning algorithm used for both classification and regression tasks.
- It is an ensemble learning method

- RF uses number of decision trees, it builds multiple trees and combines their outputs.
- Each DT is trained on a different subset of the training data.
- For classification tasks, the final prediction is made by majority voting among the trees.
- For regression tasks, the final prediction is the average of the predictions from all trees.



- RF has ability to handle complex datasets and is less prone to overfitting compared to a single decision tree.
- It often achieves high accuracy and generalization across various tasks, even without extensive hyperparameter tuning.
- However, RF models can be computationally intensive and may require more resources.

Example 12 - write python code to build Random Forest classifier for iris dataset

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators = 10, random_state = 42)
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
```

```
print('Feature importance = ', model.feature_importances_)
accuracy = accuracy_score(ytest,yp)
print('Accuracy = ', accuracy*100)
```

```
Feature importance = [0.14908549 0.02575563 0.42600688 0.39915201]
Accuracy = 100.0
```

7 Hyperparameter Tuning

- Hyperparameters are user-defined settings that control the learning process. Unlike model parameters, they are not optimized during training and must be set beforehand.
- Hyperparameter tuning is the process of identifying the best hyperparameter values to maximize model performance.
- Examples of hyperparameters include:
  1. k in k-Nearest Neighbors (kNN)
  2. Learning rate for neural networks
  3. Depth of a decision tree
  4. Train-test split ratio
- Techniques for hyperparameter tuning include:
  1. **Grid Search –** Evaluates all combinations of predefined hyperparameters. It is exhaustive but can be computationally expensive for large hyperparameter spaces.
  2. **Random Search –** Samples hyperparameters randomly from a predefined distribution. It is less computationally intensive than grid search and often finds good hyperparameters with fewer evaluations.
  3. **Bayesian Optimization –** Uses probabilistic models to predict hyperparameter performance. It iteratively selects hyperparameters based on expected performance improvements, making it more efficient than random search.
- Libraries like scikit-learn provide functions for hyperparameter tuning, such as GridSearchCV and RandomizedSearchCV.
- Advanced methods, including Bayesian optimization and genetic algorithms, are used for more efficient exploration of the hyperparameter space.

Example 13 - write python code to optimize hyperparameters of Lasso using iris datasets.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

```
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()
X = iris.data
y = iris.target
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

svmmodel = SVC()
params = {'C': [0.1,1,10,100], 'kernel':['linear', 'rbf', 'poly'], 'gamma':['scale', 'auto', 0.1, 1, 10]}
model = GridSearchCV(estimator = svmmodel, param_grid = params, cv = 5)
model.fit(xtrain, ytrain)
accuracy = model.score(xtest,ytest)

print('Best parameters = ', model.best_params_)
print('Best accuracy = ', model.best_score_ * 100)
print('Test accuracy = ', accuracy * 100)
```

```
Best parameters = 'C': 0.1, 'gamma': 0.1, 'kernel': 'poly'
Best accuracy = 95.83333333333334
Test accuracy = 100.0
```

Example 14 - write python code to optimize α in Lasso and diabetes dataset.

```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes
from sklearn.metrics import mean_squared_error

X, y = load_diabetes(return_X_y = True)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

model = LassoCV(cv=5, random_state = 42)
model.fit(xtrain, ytrain)
yp = model.predict(xtest)
mse = mean_squared_error(ytest, yp)

print('Best alpha:', model.alpha_)
print('Mean Squared Error:', mse)
print('Coefficients:', model.coef_)
```

```
Best alpha = 0.07813983904476522
Mean Squared Error = 2800.2627096146075
Coefficients = [0. -168.17 554.13 311.63 -101.86 -0. -235.14 0. 460.59 36.92]
```

Homework 8 - modify previous code by incorporating data scaling and compare the results.

Example 15 - write python code to optimize hyperparameters of DecisionTreeClassifier using iris datasets.

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()
X = iris.data
y = iris.target
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

dtmodel = DecisionTreeClassifier()
params = {'max_depth':[3, 5, 7], 'min_samples_split':[2, 5, 10]}
model = GridSearchCV(estimator = dtmodel, param_grid = params, cv = 5)
model.fit(xtrain, ytrain)
accuracy = model.score(xtest,ytest)

print('Best parameters = ', model.best_params_)
print('Best accuracy = ', model.best_score_ * 100)
print('Test accuracy = ', accuracy * 100)
```

```
Best parameters = 'max_depth': 7, 'min_samples_split': 2
Best accuracy = 94.16666666666667
Test accuracy = 100.0
```

Example 16 - write python code to optimize regularization parameter of LogisticRegression.

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_classes=2, random_state=0)
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2, random_state=42)

lrmodel = LogisticRegression()
params = {'C': np.logspace(-3,3,7)}

model = GridSearchCV(estimator = lrmodel, param_grid = params, cv = 5)
model.fit(xtrain, ytrain)

print('Best parameters = ', model.best_params_)
print('Best accuracy = ', model.best_score_ * 100)
```

Best parameters = ’C’: 0.1
Best accuracy = 90.25000000000001

Homework 9 - Add the following lines at the end of previous code and explain the results

print(’Detailed results = ’, model.cv_results_)