

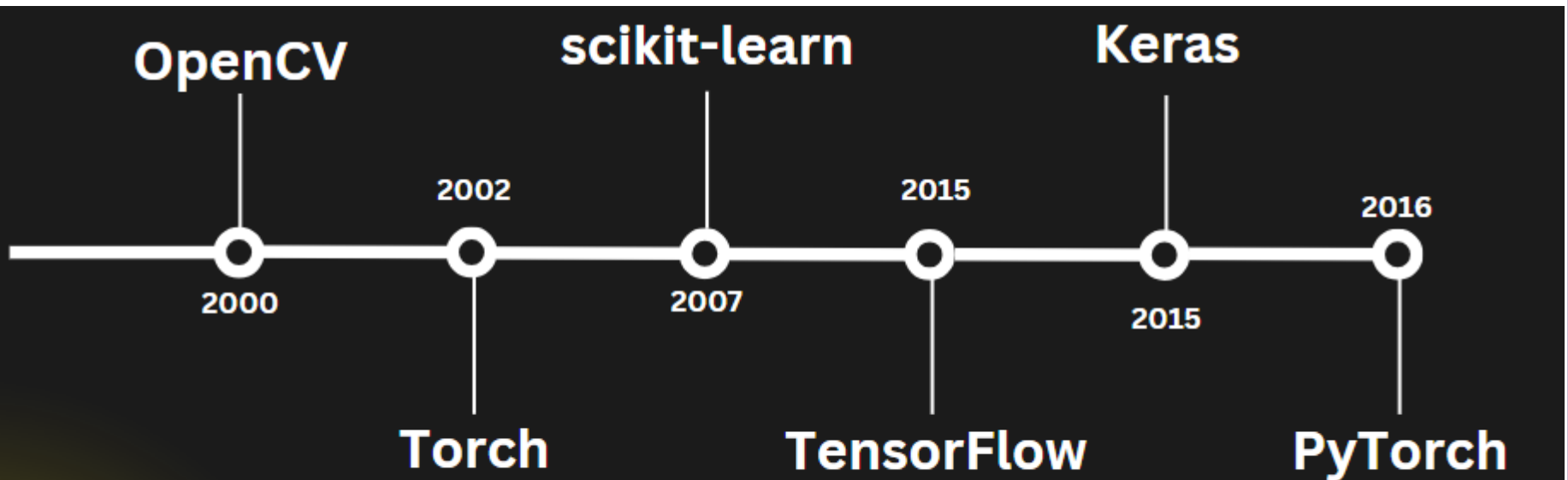
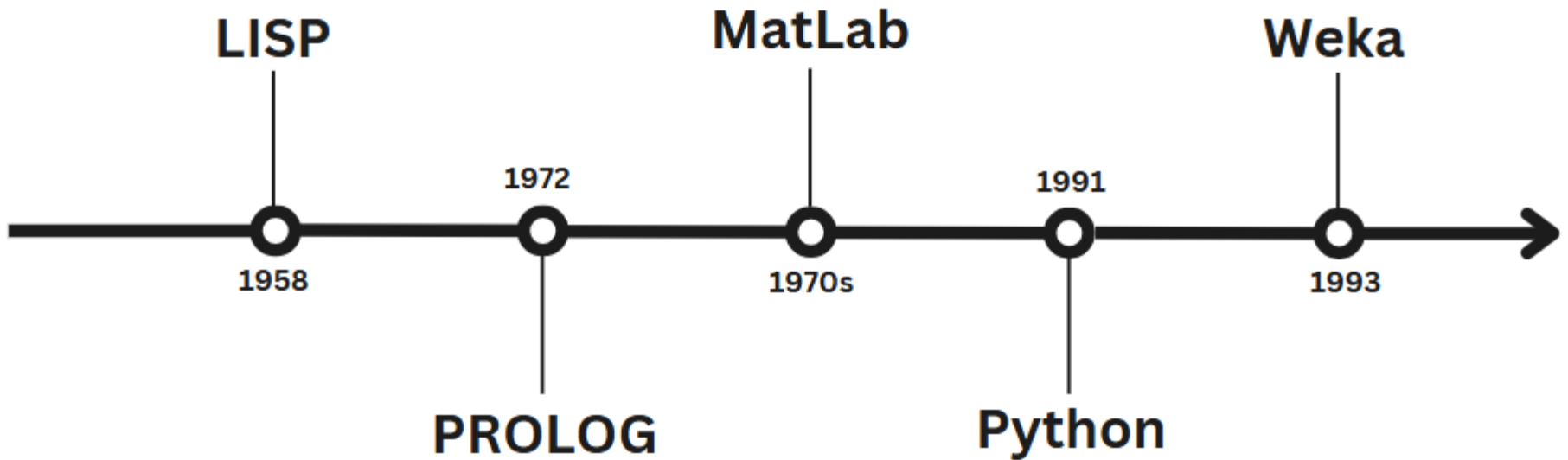


فصل چہارم شناسایی الگو پیادہ سازی

IMPLEMENTATION

محمد جواد فدائی اسلام

تاریخچه



SCIKIT-LEARN



- **scikit-learn** is a free and open-source **machine learning library** for the Python programming language.
- It supports various classification, regression and clustering algorithms.
- It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

SCIKIT-LEARN

DATA REPRESENTATION

- Numpy arrays
- Pandas DataFrames
- Sparse matrices (from SciPy)

```
np.array([ [[1,2],[3,4]],  
          [[5,6],[7,8]] ])
```

	5	6
1	2	8
3	4	

Columns

Name *Team* *Number* *Position* *Age*

0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Rows

Data

MLPCCLASSIFIER

- Multi-layer Perceptron classifier.
- This model optimizes the **log-loss function** using LBFGS or stochastic gradient descent.

MLPClassifier- PARAMETERS

hidden_layer_sizes*array-like of shape(n_layers - 2,), default=(100,)*

The *i*th element represents the number of neurons in the *i*th 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 $f(x) = \max(0, x)$

solver*{‘lbfgs’, ‘sgd’, ‘adam’}, default=‘adam’*

The solver for weight optimization.

- ‘lbfgs’ is an optimizer in the family of quasi-Newton methods.
- ‘sgd’ refers to stochastic gradient descent.
- ‘adam’ refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba. For a comparison between Adam optimizer and SGD, see [Compare Stochastic learning strategies for MLPClassifier](#).

Note: The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better.



MLPCLASSIFIER- PARAMETERS ...

batch_size*int, default='auto'*

Size of minibatches for stochastic optimizers.

learning_rate{*'constant', 'invscaling', 'adaptive'*},
default='constant'

Learning rate schedule for weight updates.

- 'constant' is a constant learning rate given by 'learning_rate_init'.
- 'invscaling' gradually decreases the learning rate at each time step 't'.
- 'adaptive' keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing. Each time two consecutive epochs fail to decrease training loss by at least tol, or fail to increase validation score by at least tol if 'early_stopping' is on, the current learning rate is divided by 5.
- Only used when solver='sgd'.

learning_rate_init *float, default=0.001*

The initial learning rate used. It controls the step-size in updating the weights. Only used when solver='sgd' or 'adam'.

MLPCLASSIFIER- PARAMETERS ...

max_iter*int, default=200*

Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.

shuffle*bool, default=True*

Whether to shuffle samples in each iteration. Only used when solver='sgd' or 'adam'.

tol*float, default=1e-4*

Tolerance for the optimization. When the loss or score is not improving by at least tol for n_iter_no_change consecutive iterations, unless learning_rate is set to 'adaptive', convergence is considered to be reached and training stops.

verbose*bool, default=False*

Whether to print progress messages to stdout.

momentum*float, default=0.9*

Momentum for gradient descent update. Should be between 0 and 1. Only used when solver='sgd'.

MLPCLASSIFIER- EXAMPLE

```
>>> from sklearn.neural_network import MLPClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> X, y = make_classification(n_samples=100,
random_state=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X,
y, stratify=y, ... random_state=1)
>>> clf = MLPClassifier(hidden_layer_sizes=(150,100,50),
max_iter=300, activation='relu', solver='adam',
random_state=1)
>>> clf.fit(X_train, y_train)
>>> clf.predict_proba(X_test[:1])
array([[0.038..., 0.961...]])
>>> clf.predict(X_test[:5, :])
array([1, 0, 1, 0, 1])
>>> clf.score(X_test, y_test) 0.8...
```

MLPCLASSIFIER- METHOD

fit(X, y)

Fit the model to data matrix X and target(s) y.

predict(X)

Predict using the multi-layer perceptron classifier.

score(X, y, sample_weight=None)

Return the mean accuracy on the given test data and labels.

sklearn.svm

Support vector machine algorithms.

User guide. See the [Support Vector Machines](#) section for further details.

<u>LinearSVC</u>	Linear Support Vector Classification.
<u>LinearSVR</u>	Linear Support Vector Regression.
<u>NuSVC</u>	Nu-Support Vector Classification.
<u>NuSVR</u>	Nu Support Vector Regression.
<u>OneClassSVM</u>	Unsupervised Outlier Detection.
<u>SVC</u>	C-Support Vector Classification.
<u>SVR</u>	Epsilon-Support Vector Regression.
<u>l1_min_c</u>	Return the lowest bound for C.

SVM

<https://scikit-learn.org/dev/modules/svm.html#svm>

- **Support vector machines (SVMs)** are a set of supervised learning methods used for:
 - Classification
 - Multi-class classification
 - Scores and probabilities
 - Unbalanced problems
 - regression
 - outliers detection.

MLP WITH TENSORFLOW 2.0 AND KERAS

```
# Imports
import tensorflow
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

# Configuration options
feature_vector_length = 784
num_classes = 10

# Load the data
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

# Reshape the data - MLPs do not understand such things as '2D'.
# Reshape to 28 x 28 pixels = 784 features
X_train = X_train.reshape(X_train.shape[0], feature_vector_length)
X_test = X_test.reshape(X_test.shape[0], feature_vector_length)
```

MLP WITH TENSORFLOW 2.0 AND KERAS ...

```
# Convert into greyscale
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# Convert target classes to categorical ones
Y_train = to_categorical(Y_train, num_classes)
Y_test = to_categorical(Y_test, num_classes)

# Set the input shape
input_shape = (feature_vector_length,)
print(f'Feature shape: {input_shape}')
```

MLP WITH TENSORFLOW 2.0 AND KERAS ...

```
# Create the model
model = Sequential()
model.add(Dense(350, input_shape=input_shape, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Configure the model and start training
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, Y_train, epochs=10, batch_size=250, verbose=1, validation_split=0.2)

# Test the model after training
test_results = model.evaluate(X_test, Y_test, verbose=1)
print(f'Test results - Loss: {test_results[0]} - Accuracy: {test_results[1]}%')
```