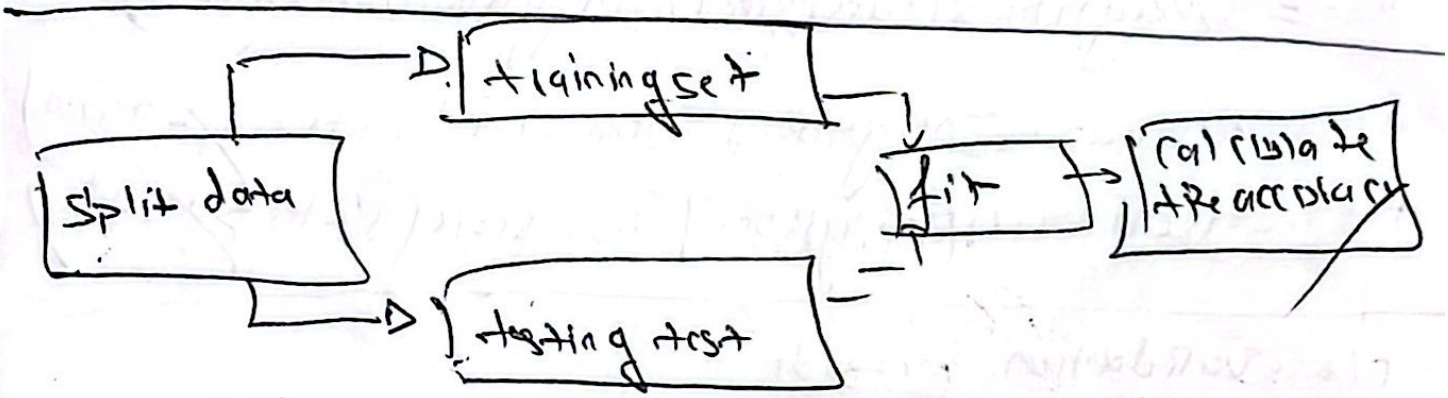
• KNN by finding Jimbo

```
from sklearn.neighbors import KNeighborsClassifier

X = [ ].values.

y = [ ].values.

print (X.shape, y.shape)

knn = KNeighborClassifier (n_neighbors = 15)

knn.fit (X, y)

Prediction = knn.predict (X_new)

print ('Prediction: {}'.format (prediction))
```




```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split (
    X, y, test_size = 0.3, random_state = 21, stratify = y)

knn = KNeighborsClassifier (n_neighbors = 6)

knn.fit (X_train, y_train)

print (knn.score (X_test, y_test))
```

↳ accuracy

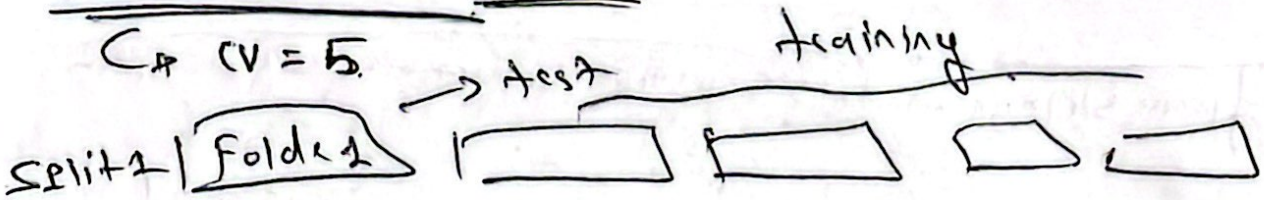
- Underfitting
  - Happens when machine learning task complexity

- overfeting
  - to complex.

```
train_accuracies = {}
test_accuracies = {}
neighbors = np.arange(1, 26)
for neighbor in neighbors:
knn = KNeighborsClassifier(n_neighbors=neighbor)
train_accuracies[neighbor] = knn.score(X_train, y_train)
test_accuracies[neighbor] knn.score(X_test, y_test)
```

Cross validation method

C> cv = 5.



Split 5

C> K fold

```
from sklearn.model_selection import cross_val_score, KFold
kf = KFold(n_split=6, shuffle=True, random_state=42)
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=kf)
```

⊕ Regularized regression

↳ a way to escape overfitting

↳ Regularization: Penalize large coefficient

① Ridge regression

loss function = OLS loss function + $\alpha \sum_{i=1}^{n} a_i^2$

penalize coefficients

Hyperparameter

↳ parameter that used to optimize model

↳ cross validation

↳ $\alpha = 0 = OLS$

↳ α 가 high → underfitting

```
from sklearn.linear_model import Ridge
scores = []
for alpha in [0.1, 1.0, 10.0, 100, 1000.0]:
    ridge = Ridge(alpha = alpha)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_test)
```

(3)

scores.append(ridge.score(X_test, Y_test))

Print(scores)

Lasso regression

Loss func = OLS loss func + $\alpha * \sum_{i=1}^{n} |a_i|$

- Shrink the coefficients of less important features to zero
- Non-zero => selected by lasso

Logistics regression → (0, 1) classification

- P > 0.5 → 1
- P < 0.5 → 0

```
from sklearn.linear_model import LogisticRegression
logreg = LogsticsRegression()

X_train, X_test, Y_train, Y_test = train_test_split
(X, Y, test_size = 0.3, random_state = 42)
logreg = fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
```

Predicting probabilities

```
Y_pred_probs = logreg.predict_proba(X_test)[:, 1]
Print(Y_pred_probs[0])
```

DL

Simple

+ Input layer X

+ Hiden layer

+ output layer y Step (1)

$X = [x_1 \text{ —— } x_n]$ → Input vectors

$W = [w_1, \text{ —— } w_n]$ → Weight vector

b → Bais

z → weighted sum

→ activation function

$a = f(z)$

Sigmoid ⟶ (0, 1) → classification

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

↳ Good for probability

Cs for Hiden layers (ReLu)

$$f(z) = \max(0, z)$$

Cs tanhelize

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

Cs softmax → multy class ⇒ conv) → Proba

$$f(z_i) = \frac{e^{z_i}}{\sum e^{z_j}}$$

(1)

Forward Propagation

→ multi-layer

Step (2)

$$Z^{(l)} = W^{(l)} A^{(l-1)} + b^{(l)}$$

$$A^{(l)} = f(Z^{(l)})$$

$$\hat{y} = f(W^{(L)} A^{(L-1)} + b^{(l)})$$

↳ loss function (Error)

1) MSE

$$L = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

2) Cross-Entropy

$$L = -\sum y_i \log(\hat{y}_i)$$

↳ Backpropagation

minimize loss

↳ Gradient calculation

Step (3)

each (w_{ij}) → weight

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial A^{L}} \cdot \frac{\partial A^{(L)}}{\partial Z^{(l)}} \frac{\partial Z^{(l)}}{\partial w_{ij}}$$

(SGD) → Stochastic Gradient Decent

$$w := w - \alpha \frac{\partial L}{\partial w}$$

α : learning rate

Step (4)

Repeat

(3)

1. Supervised Learning (Labeled Data)

👉 Used when we have **input-output pairs** and want to train a model to map inputs to outputs.

Use Case	Machine Learning Algorithm	Deep Learning Model
Spam Detection (Email)	Logistic Regression, Naïve Bayes	Recurrent Neural Network (RNN)
Fraud Detection (Banking)	Decision Trees, Random Forest	Deep Neural Network (DNN)
Image Classification	SVM, Random Forest	Convolutional Neural Network (CNN)
Sentiment Analysis	Naïve Bayes, SVM	Long Short-Term Memory (LSTM)
Stock Price Prediction	Linear Regression	LSTM, Transformer Networks

2. Unsupervised Learning (No Labeled Data)

👉 Used when we want to find **patterns and structures** in data without predefined labels.

Use Case	Machine Learning Algorithm	Deep Learning Model
Customer Segmentation	K-Means, DBSCAN	Autoencoders
Anomaly Detection	Isolation Forest, One-Class SVM	Variational Autoencoder (VAE)
Topic Modeling (Text)	Latent Dirichlet Allocation	Transformer-based Models (BERT)
Image Clustering	K-Means, PCA	CNN-based Feature Extraction

3. Reinforcement Learning (Agent-Based Learning)

👉 Used in **decision-making** scenarios where an agent learns by interacting with an environment.

Use Case	Algorithm
Game AI (Chess, Go)	Deep Q-Learning (DQN), AlphaGo
Robotics (Path Planning)	Policy Gradient, PPO, A3C
Self-Driving Cars	Deep Deterministic Policy Gradient (DDPG)

Choosing the Right Algorithm for ML & DL

- **Small data?** Use **Machine Learning** (Decision Trees, SVM, Logistic Regression).
- **Big data?** Use **Deep Learning** (CNNs, RNNs, Transformers).
- **Text-related?** Use **NLP models** (LSTMs, BERT, GPT).
- **Real-time decision-making?** Use **Reinforcement Learning**.