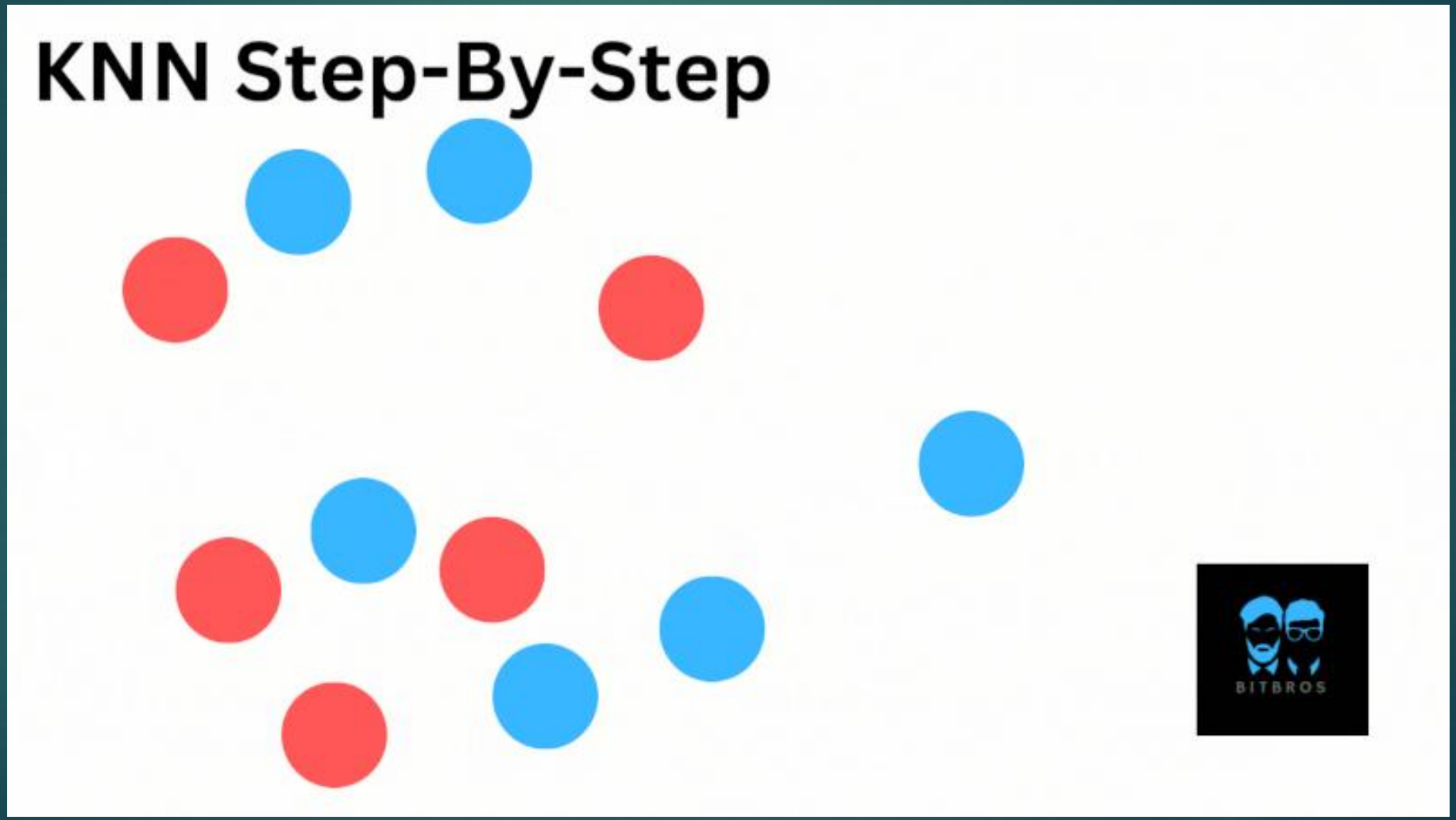
KNN

- Tell me your friend's group, I will tell where you belong.
- Store all training data
- For new Data Point:
  - Compute distance with all data point (Euclidean, Manhattan, Mankowski)
  - Pick k nearest Neighbor
  - Assign it to most common class
- Advantages:
  - Simple and Intuitive
  - No Assumption on Data Distribution
- Disadvantages:
  - Slow for large dataset. (distance computed for every point)
  - Sensitive to unscaled feature
  - Struggles in high dimension

KNN



Hyperparameter

- Parameter → Learned from data (i.e. weights in linear regression)
- Hyperparameter → Set before training. Controls how model learn

1. k (number of neighbors)
   - Small k → Sensitive to Noise (Overfitting)
   - Large → Smooth decision boundary (underfitting)
2. Distance metric
   - Euclidean → Default
   - Manhattan
   - Mankowski
   - Cosine
3. Weights
   - uniform → Each Neighbor has equal weight
   - Distance → Closer Neighbor More important

knn classifier

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

# Display matrix
sns.heatmap(cm, annot=True, fmt="d", xticklabels=iris.target_names, yticklabels=iris.target_names,
cmap="Blues")
plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.title("Confusion Matrix (Iris Dataset)"); plt.show()

print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Error

MSE (RMSE)

- Mean Squared Error
- Average Squared difference between predicted vs actual value
- (sum{(yp – ya) ** 2}) / n
- Sensitive to large error

R – Squared

- Coefficient of determination
- 1 – sum((y-yp_bar) ** 2) / sum((y – ybar) ** 2)
- 1 ➔ Perfect prediction, 0 ➔ No better than mean, -1 ➔ Worse than mean

MAE

- Mean absolute error
- Average of absolute error between predicted vs actual value
- If median is used instead of mean then its Median Absolute error

Confusion Matrix

- Table that summarizes how well a classification model performs by comparing Actual vs Predicted label

- For binary classification

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Significance:

TP ➔ Model correctly predicted positive value

TN ➔ Model correctly predicted negative value

FP ➔ Model predicted positive but it was wrong (Type – I)

FN ➔ Model predicted negative but it was wrong (Type – II)

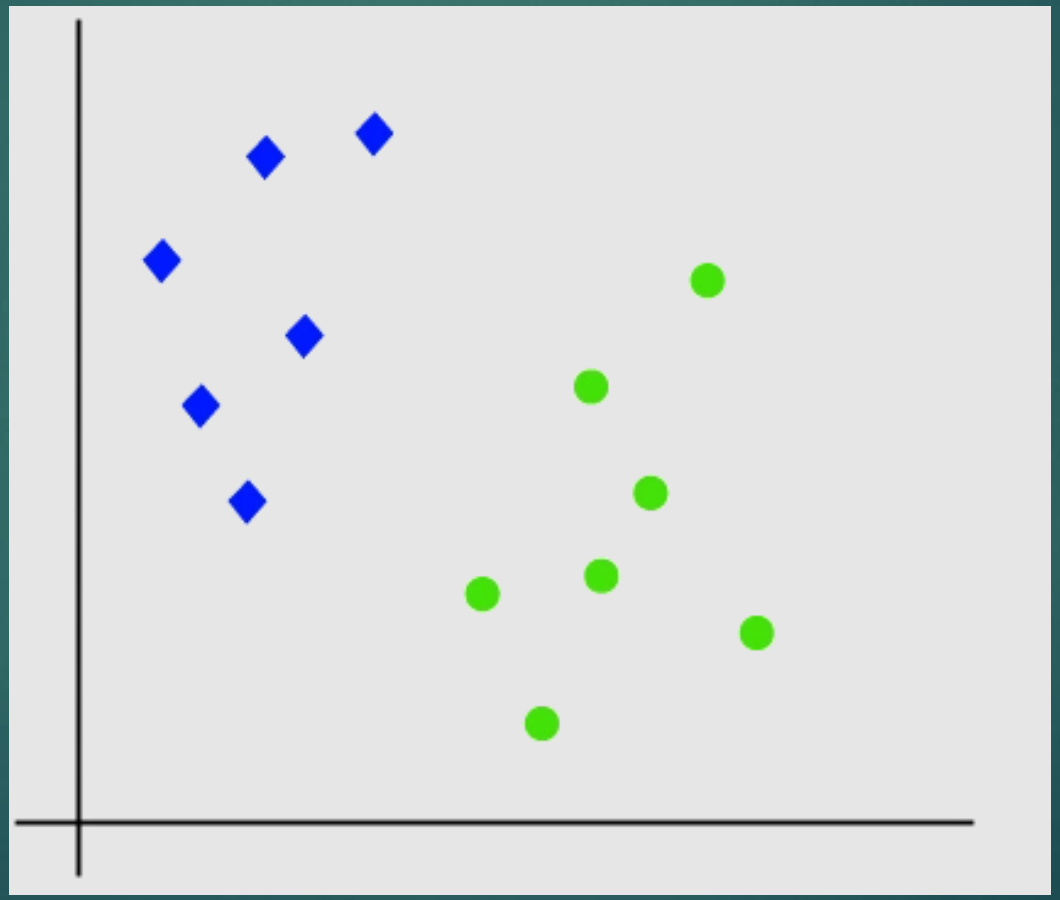
Other Metrics

- Accuracy
  - Overall Correctness
  - (TP + TN) / (TP + TN + FP + FN)
- Precision
  - Out of predicted positive how many were correct ?
  - TP / (TP + FP)
- Recall (Sensitivity)
  - Out of actual positive how many did we catch ?
  - TP / (TP + FN)
- Specificity
  - TN / (TN + FP)
- F1 Score
  - 2 * precision * recall / (precision + recall)

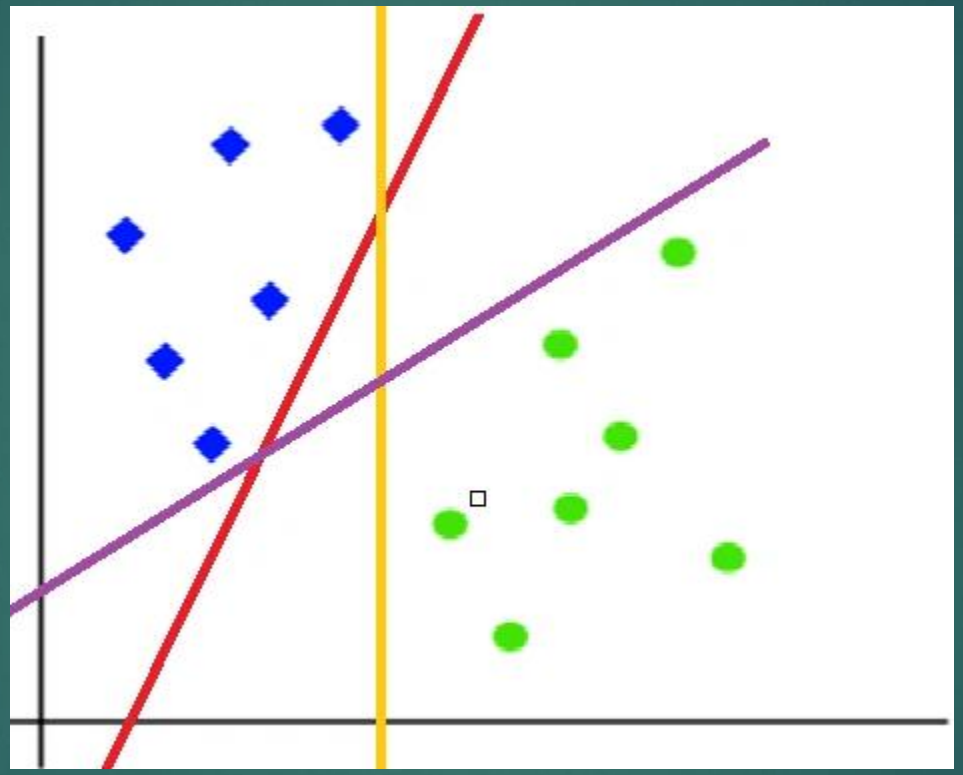
Why it matters

- Shows where the model is making error beyond the accuracy
- Accuracy itself is useless for imbalance class.
- Example: Disease test where 5% of patient are infected. Predicting everyone healthy gives 95% accuracy but misses all patients
- Helps in problem specific optimization:
  - Medical Diagnostic: Reduce **FN** is critical (Don't miss sick patient) [Recall]
  - Spam Detection: Reducing **FP** is critical (Don't block genuine mail) [Precision]
  - Credit Fraud: **FN** & **FP** critical (No loss vs customer card not block) [F1 Score]
  - Rain Prediction: **FN** (Farmer may lose crop) or **FP** (Trip Cancellation)
  - Security System for terrorist fact detection: **FN** critical

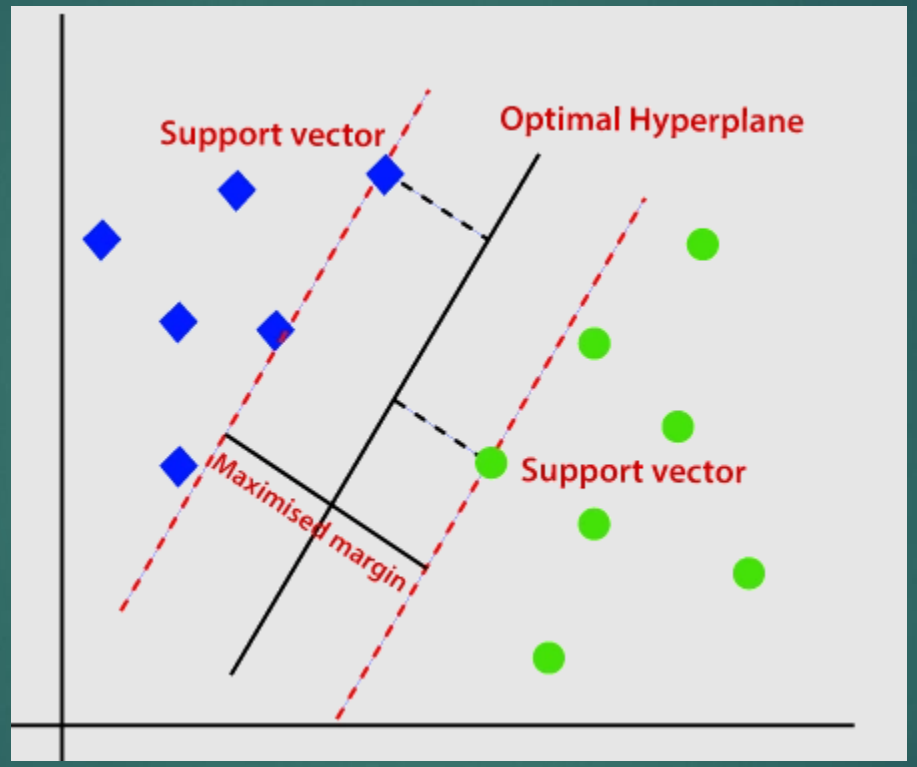
SVM



SVM



SVM


Support vector
Optimal Hyperplane
Maximised margin
Support vector

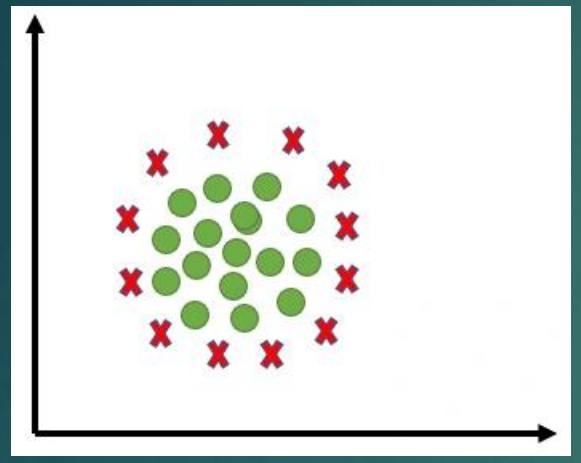
SVM

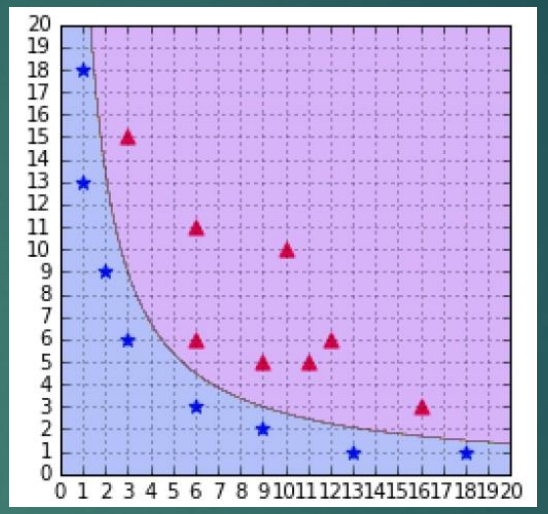
- Find best boundary between the classes
- Find Hyperplane that separates class with maximum margin
- Terms:
  - Hyperplane: Decision boundary separating the class
  - Margin: Distance between hyperplane & nearest point (Support Vector)
  - Support Vector: Data point closes to Hyperplane
- Advantage:
  - Works well with high-dimensional space
  - Works well with clear margin of separation
- Disadvantage:
  - Sensitive to noisy data / overlapping class
  - Computationally expensive for large dataset

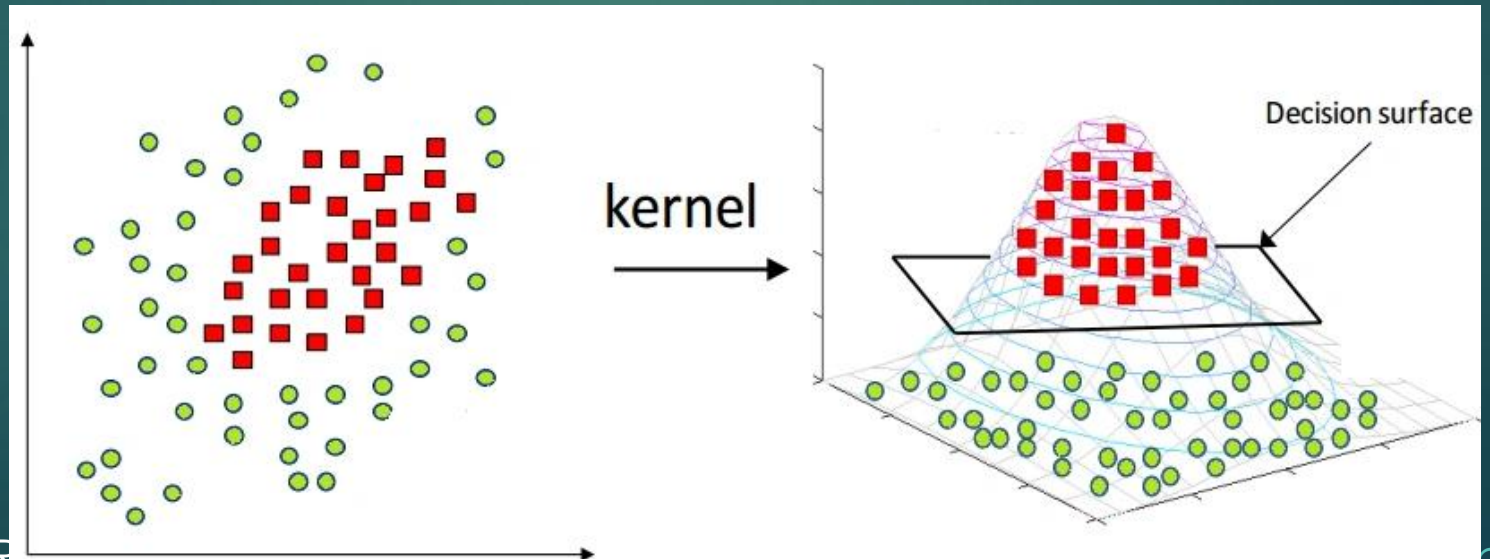
Algorithm

- Identify the Support Vectors. Data points closest to data point of another class
- Identify the equation of hyperplane such that this plane will have maximum distance with Support Vectors identifies
- If there are multiple support vectors finding equation of hyperplane is like regression of their midpoint
- On training evaluate a point with hyperplane equation and assign label according to ax1 + by1 + c > 0 or less

Non – Linear Boundary







Kernel Trick

- Sometimes classes are not linearly separable in original space.

Kernel Trick:

- Map input data into higher-dimensional space
- Allows linear separation in that space without explicitly computing coordinates

Common Kernels:

- Linear: No mapping, original feature space
- Polynomial: Non-linear boundaries (degree d)
- RBF (Gaussian): Infinite-dimensional space, flexible boundary
- Sigmoid: Similar to neural network activation

Benefits:

- Allows SVM to handle non-linear classification problems
- Keeps computation efficient via kernel functions

Hyper Parameter

- Parameter: Learned from data (e.g., support vectors)
- Hyperparameter: Set before training, controls model behavior

C (Regularization):

- Small C → Wide margin, more misclassifications allowed (underfitting)
- Large C → Narrow margin, fewer misclassifications (overfitting)

Kernel type: linear, poly, rbf, sigmoid

Gamma (for RBF / poly):

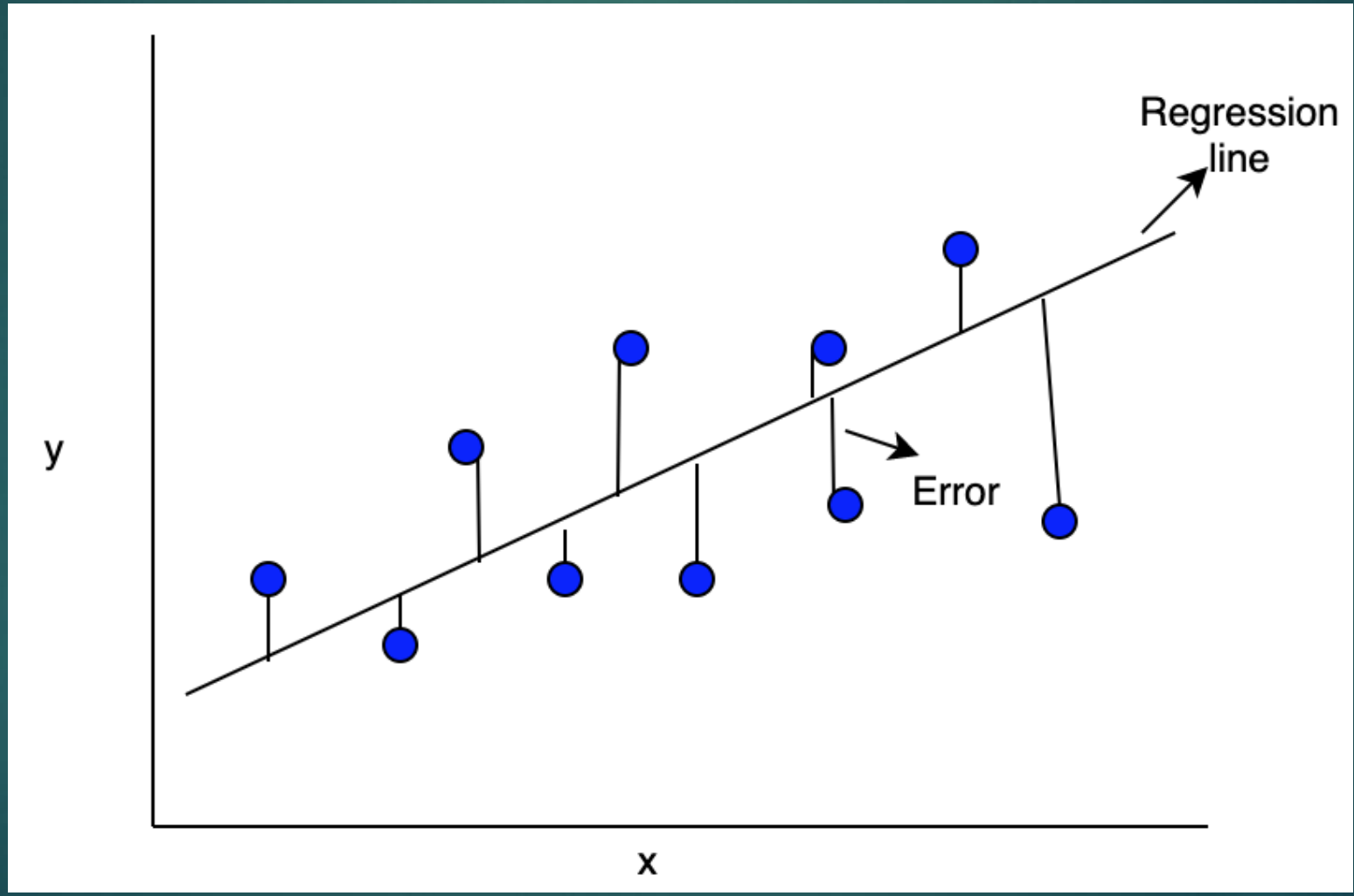
- Small gamma → smooth decision boundary
- Large gamma → more complex boundary

Degree (for polynomial kernel)

Classification Code

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

svm_model = SVC(kernel='rbf', C=1, gamma='scale')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", xticklabels=iris.target_names, yticklabels=iris.target_names,
cmap="Blues")
plt.xlabel("Predicted"); plt.ylabel("Actual"); plt.title("Confusion Matrix (Iris Dataset)"); plt.show()

print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

Linear Regression



Goal – Minimize Error

1. Find the residual or error. (Error e = Ypred – Yactual)
2. Sum of error = sum(ei)
3. If the equation of line is Yp = b1X + bo

1. The Goal: Minimize Error

Our objective is to minimize the **Sum of Squared Residuals (SSR)**.

$$SSR = \sum(Y_i - \hat{Y}_i)^2 = \sum(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2$$

2. Derivative for the Intercept ($\hat{\beta}_0$)

We take the partial derivative of the SSR with respect to $\hat{\beta}_0$ and set it to zero.

$$\frac{\partial SSR}{\partial \hat{\beta}_0} = -2\sum(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i) = 0$$

This simplifies to our first **normal equation**:

$$\sum Y_i = n\hat{\beta}_0 + \hat{\beta}_1 \sum X_i$$

Goal – Minimize Error

3. Derivative for the Slope ($\hat{\beta}_1$)

Next, we take the partial derivative of the SSR with respect to $\hat{\beta}_1$ and set it to zero.

$$\frac{\partial SSR}{\partial \hat{\beta}_1} = -2 \sum X_i (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i) = 0$$

This simplifies to our second **normal equation**:

$$\sum X_i Y_i = \hat{\beta}_0 \sum X_i + \hat{\beta}_1 \sum X_i^2$$

Solving the two normal equations simultaneously gives us the final formulas for our estimators.

Slope ($\hat{\beta}_1$):

$$\hat{\beta}_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - X)^2}$$

Intercept ($\hat{\beta}_0$):

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

Goal – Minimize Error

1. **Solve (1) for β_0:**

$$n\beta_0 = \sum_i y_i - \beta_1 \sum_i x_i \quad \Rightarrow \quad \beta_0 = \frac{\sum_i y_i - \beta_1 \sum_i x_i}{n} = \bar{y} - \beta_1 \bar{x}.$$

2. **Substitute β_0 into (2):**

$$(\bar{y} - \beta_1 \bar{x}) \sum_i x_i + \beta_1 \sum_i x_i^2 = \sum_i x_i y_i.$$

3. Replace $\sum_i x_i = n\bar{x}$:

$$\bar{y}\, n\bar{x} - \beta_1 \bar{x}\, n\bar{x} + \beta_1 \sum_i x_i^2 = \sum_i x_i y_i.$$

4. Collect terms containing β_1 on left:

$$\beta_1 \left(\sum_i x_i^2 - n\bar{x}^2 \right) = \sum_i x_i y_i - n\bar{x}\bar{y}.$$

5. Solve for β_1:

$$\boxed{\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}}$$

$$\sum Y_i = n\hat{\beta}_0 + \hat{\beta}_1 \sum X_i$$

$$\sum X_i Y_i = \hat{\beta}_0 \sum X_i + \hat{\beta}_1 \sum X_i^2$$

Python Code

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd

housing = fetch_california_housing()
X = pd.DataFrame(housing.data, columns=housing.feature_names)
y = housing.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("R² Score:", r2)

coefficients = pd.DataFrame({"Feature": housing.feature_names, "Coefficient": model.coef_})
print(coefficients)
```

Cross - Validation

- Model performance is dependent in a way data is split
- Solution: cross-validation.
- Split data into n group. Hold one and train on other. Repeat process
- Using more folds can be computationally expensive
- Stores result of R2 value in an array.
- **scoring**: precision, recall, f1, accuracy, roc_auc, neg_mean_squared_error, ..

Example:

```
from sklearn.model_selection import cross_val_score
cv_result = cross_val_score(model, X, y, cv=5) # 5-fold cv
```

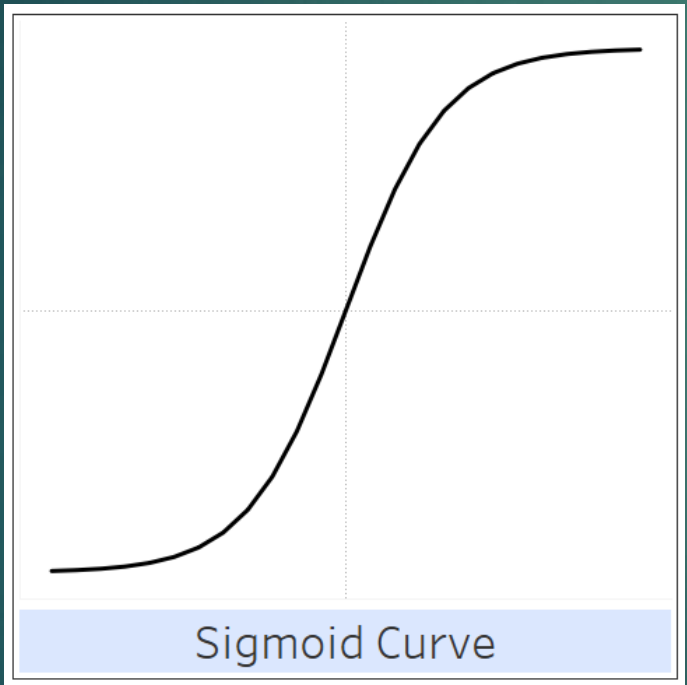
Logistic Regression

- Used for classification problem
- It outputs a probability using activation functions
- If p > .5 then predicted as 1 else predicted as 0
- If we set threshold, p=0 then all will be predicted True
- ROC (Receiver Operating Characteristics) curve shows how well classifier works for different threshold
- For each threshold, calculate FPR , TPR and plot FPR vs TPR in graph

X-axis ➔ FPR how often the model says "yes" when it should have said "no."
Y-axis ➔ how often the model correctly says "yes" when it really is "yes."

Activation Function

- Logistic Regression always uses sigmoid activation function
- For multi-class, it uses SoftMax activation function



Sigmoid Curve

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

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Logistic Regression

```
from sklearn.metrics import roc_curve
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
import matplotlib.pyplot as plt

breast_cancer = datasets.load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=42)
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X_train, y_train)

# Compute predicted probabilities
y_pred_prob = logreg.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# To plot dotted diagonal line
plt.plot([0, 1], [0, 1], "k--“, label=“Random Guess“)
plt.plot(fpr, tpr, label="Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Logistic Regression ROC curve")
plt.show()
```

Area Under Curve

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

breast_cancer = datasets.load_breast_cancer()
X = breast_cancer.data
y = breast_cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)

# Did Not converge on default iteration rate
logreg = LogisticRegression(max_iter=10000)
logreg.fit(X, y)

# Compute predicted probabilities
y_pred_prob = logreg.predict_proba(X_test)[:, 1]

print("AUC: {}".format(roc_auc_score(y_test, y_pred_prob)))
cv_auc = cross_val_score(logreg, X, y, cv=5, scoring="roc_auc")
print("Cross validation score {}".format(cv_auc))
```

Tuning Hyperparameter

- Parameters like:
  - a in lasso and ridge
  - k in knn
  - max_iter in logistic regression

- Hyperparameter can't be learned by fitting a model

- Choosing a hyperparameter
  - Try a bunch of different hyperparameter value
  - fit them separately
  - See how well each parameter performs
  - Choose best performing one

Grid Search Cross Validation

- For hyperparameter, validation is done for all combination
- Computationally expensive but guaranteed to give best model
- Based on performance choose hyperparameter that gives best result
- Parameter values to try has to be passed as dictionary

Randomized Search CV

- Doesn't try to fit for each combination of hyperparameter
- It can jump values on grid
- Used to same computation time
- Same code can be used for Python
- Instead of GridSearchCV use RandomizedSearchCV

Ensemble Learning

- Combine multiple model to improve robustness and accuracy

Bagging

- Trains the same algorithm on random set of data and average
- Example: BaggingClassifier, RandomForestClassifier

Stacking:

- Combine multiple models independently into a meta model
- Example: StackingClassifer

Boosting:

- Train model sequentially, new model focus on correcting mistake of previous one
- For weak learner
- Example: AdaBoostClassifier, GradientBoostingClassifier, XGBoost

Bagging Classifier

```
from sklearn.ensemble import BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Base model
knn = KNeighborsClassifier(n_neighbors=11)

# Bagging ensemble
bagging_knn = BaggingClassifier(estimator=knn, n_estimators=50, random_state=42)
bagging_knn.fit(X_train, y_train)

# Evaluate
y_pred = bagging_knn.predict(X_test)
print("Bagging KNN Accuracy:", accuracy_score(y_test, y_pred))
```

Stacking Classifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define base learners
base_learners = [
    ("knn", KNeighborsClassifier(n_neighbors=11)),
    ("svm", SVC(probability=True)),
    ("tree", DecisionTreeClassifier()),
]

# Meta-model
meta_model = LogisticRegression()

# Stacking
stack = StackingClassifier(estimators=base_learners, final_estimator=meta_model, cv=5)
stack.fit(X_train, y_train)

print("Stacking Accuracy:", stack.score(X_test, y_test))
```

Boosting

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Weak learner
tree = DecisionTreeClassifier(max_depth=1)  # decision stump

# Boosting
boost = AdaBoostClassifier(
    estimator=tree,
    n_estimators=50,
    learning_rate=1.0,
    random_state=42
)

boost.fit(X_train, y_train)
y_pred = boost.predict(X_test)
```

Encoding

- Scikit learn doesn't accept categorical feature
- Categorical features has to be encoded numerically
- Convert data to dummy variables.
- Some common encoding techniques are:
  - Label Encoding
  - One-Hot Encoding
  - Binary Encoding
  - Target Mean Encoding
  - Frequency / Count Encoding
  - Embedding

Label Encoding

- Each category is assigned an integer value.
- Example: [“red”, “blue”, “green”] → [2, 0, 1]
- Useful for ordinal data where order matters
- Can mislead for nominal data as model may assume numeric relationships

One - Hot Encoding

- Creates a binary column for each category
- Example: [“red”, “blue”, “green”] →

red	blue	green
1	0	0
0	1	0
0	0	1

- Works well for nominal data
- Downside: Creates many columns if category are numerous

Binary Encoding

Binary Encoding

- First convert categories into integers, then into binary
- After its separated into multiple columns (of binary digit)
- Example: “red” → 1 → 01; “blue” → 2 → 10; “Green” → 3 → 11
- More compact that One Hot Encoding
- Works well for high cardinality feature

Other Encoding

Target Mean Encoding

- Replace categories with the mean of target variable for each
- Example: if target is purchasing probability
- Gender: female → 0.6; male → 0.4;

Frequency Encoding

- Replace categories with their frequency or count
- Works well with high cardinality feature
- City: "London" (1000 times), "Paris" (500 times) → 1000, 500

Embedding (For Deep Learning)

- Represent category in dense vector space
- Used in NLP and recommendation system

Python Code

Pipeline

- We have multiple steps on training of ML algorithm
- Instead of doing fit and transform in each we combine these steps as a pipeline and use it for train or predict

Save Model

- After model has been completed we need to store model
- Saving and importing model ensure that we don't run into same long running and time-consuming training process again
- We make use of joblib library for it

Example

```
import joblib
joblib.dump(model, "model_name.pkl")
loaded_model = joblib.load(" model_name.pkl")
```