

Project 2

1. Download a dataset from Kaggle

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
```

In [2]:

```
df = pd.read_csv("breast-cancer-wisconsin-data_data.csv")
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...

5 rows × 33 columns

2) Handle missing values (and basic cleaning)

In [3]:

```
import numpy as np

# Drop obvious non-feature columns if present
for col in ["Unnamed: 32", "id"]:
    if col in df.columns:
        df = df.drop(columns=[col])

# Target
if "diagnosis" not in df.columns:
    raise ValueError("Expected a 'diagnosis' column in the dataset.")

y = df["diagnosis"].map({"M": 1, "B": 0})
X = df.drop(columns=["diagnosis"])

# Check missing values
missing = X.isna().sum().sort_values(ascending=False)
print("Top missing columns:\n", missing.head(10))
print("\nAny missing in target?", y.isna().any())
```

Top missing columns:

radius_mean	0
texture_mean	0
symmetry_worst	0
concave points_worst	0
concavity_worst	0
compactness_worst	0
smoothness_worst	0
area_worst	0
perimeter_worst	0
texture_worst	0

dtype: int64

Any missing in target? False

In [4]:

```
import pandas as pd

df = pd.read_csv("breast-cancer-wisconsin-data_data.csv")

missing = df.isna().sum().sort_values(ascending=False)
missing = missing[missing > 0]
print(missing)
```

Unnamed: 32

569

dtype: int64

3) Apply ML models + Grid Search + evaluate performance

split train/test

```

In [10]: from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

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

common_preprocess = [
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
]

pipelines_and_params = {
    "LogReg": (
        Pipeline(common_preprocess + [("model", LogisticRegression(max_iter=5000))]),
        {
            "model__C": [0.01, 0.1, 1, 10],
            "model__penalty": ["l2"],
            "model__solver": ["lbfgs"]
        }
    ),
    "SVM": (
        Pipeline(common_preprocess + [("model", SVC(probability=True))]),
        {
            "model__C": [0.1, 1, 10],
            "model__kernel": ["rbf", "linear"],
            "model__gamma": ["scale", "auto"]
        }
    ),
    "KNN": (
        Pipeline(common_preprocess + [("model", KNeighborsClassifier())]),
        {
            "model__n_neighbors": [3, 5, 7, 9, 11],
            "model__weights": ["uniform", "distance"]
        }
    ),
    "RandomForest": (
        Pipeline([("imputer", SimpleImputer(strategy="median")),
                  ("model", RandomForestClassifier(random_state=42))]),
        {
            "model__n_estimators": [200, 500],
            "model__max_depth": [None, 5, 10],
            "model__min_samples_split": [2, 5],
            "model__min_samples_leaf": [1, 2]
        }
    ),
}

results = []

best_models = {}

for name, (pipe, params) in pipelines_and_params.items():
    gs = GridSearchCV(
        pipe,
        params,
        cv=5,
        scoring="roc_auc",      # strong metric for binary classification
        n_jobs=-1
    )
    gs.fit(X_train, y_train)
    best_models[name] = gs.best_estimator_

    # Predict
    y_pred = gs.predict(X_test)

    # Probabilities for ROC AUC (handle models that provide predict_proba)
    if hasattr(gs.best_estimator_, "predict_proba"):
        y_proba = gs.best_estimator_.predict_proba(X_test)[:, 1]
    else:
        # fallback (rare here because we set SVC(probability=True))
        y_proba = None

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba) if y_proba is not None else np.nan

    results.append({
        "model": name,
        "best_params": gs.best_params_,
        "test_accuracy": acc,
        "test_f1": f1,
        "test_roc_auc": auc
    })

results_df = pd.DataFrame(results).sort_values(by="test_roc_auc", ascending=False)
results_df

```

Out[10]:

	model	best_params	test_accuracy	test_f1	test_roc_auc
3	RandomForest	{'model__max_depth': None, 'model__min_samples...	0.973684	0.962963	0.995040
1	SVM	{'model__C': 10, 'model__gamma': 'scale', 'mod...	0.973684	0.962963	0.992725
0	LogReg	{'model__C': 10, 'model__penalty': 'l2', 'mode...	0.973684	0.963855	0.986442
2	KNN	{'model__n_neighbors': 11, 'model__weights': '...	0.947368	0.925000	0.982308

Detailed report for the top model:

In [8]:

```
top_name = results_df.iloc[0]["model"]
top_model = best_models[top_name]

y_pred = top_model.predict(X_test)
y_proba = top_model.predict_proba(X_test)[:, 1]

print("Top model:", top_name)
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification report:\n", classification_report(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

Top model: RandomForest
Confusion matrix:
[[72 0]
 [3 39]]

Classification report:
precision recall f1-score support

 0 0.96 1.00 0.98 72
 1 1.00 0.93 0.96 42

 accuracy 0.97 0.97 0.97 114
macro avg 0.98 0.96 0.97 114
weighted avg 0.97 0.97 0.97 114

ROC AUC: 0.9950396825396826

4) Two feature selection methods (most important features)

In [32]:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, classification_report, confusion_matrix

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

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

common_preprocess = [
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
]

pipelines_and_params = {
    "LogReg": (
        Pipeline(common_preprocess + [("model", LogisticRegression(max_iter=5000))]),
        {
            "model__C": [0.01, 0.1, 1, 10],
            "model__penalty": ["l2"],
            "model__solver": ["lbfgs"]
        }
    ),
    "SVM": (
        Pipeline(common_preprocess + [("model", SVC(probability=True))]),
        {
            "model__C": [0.1, 1, 10],
            "model__kernel": ["rbf", "linear"],
            "model__gamma": ["scale", "auto"]
        }
    ),
    "KNN": (
        Pipeline(common_preprocess + [("model", KNeighborsClassifier())]),
        {
            "model__n_neighbors": [3, 5, 7, 9, 11],
            "model__weights": ["uniform", "distance"]
        }
    ),
    "RandomForest": (
        Pipeline([("imputer", SimpleImputer(strategy="median")),
                  ("model", RandomForestClassifier(random_state=42))]),
        {
            "model__n_estimators": [200, 500],
            "model__max_depth": [None, 5, 10],
            "model__min_samples_split": [2, 5],
            "model__min_samples_leaf": [1, 2]
```

```
    }
    ),
}

results = []

best_models = {}

for name, (pipe, params) in pipelines_and_params.items():
    gs = GridSearchCV(
        pipe,
        params,
        cv=5,
        scoring="roc_auc",      # strong metric for binary classification
        n_jobs=-1
    )
    gs.fit(X_train, y_train)
    best_models[name] = gs.best_estimator_

    # Predict
    y_pred = gs.predict(X_test)

    # Probabilities for ROC AUC (handle models that provide predict_proba)
    if hasattr(gs.best_estimator_, "predict_proba"):
        y_proba = gs.best_estimator_.predict_proba(X_test)[: , 1]
    else:
        # fallback (rare here because we set SVC(probability=True))
        y_proba = None

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba) if y_proba is not None else np.nan

    results.append({
        "model": name,
        "best_params": gs.best_params_,
        "test_accuracy": acc,
        "test_f1": f1,
        "test_roc_auc": auc
    })

results_df = pd.DataFrame(results).sort_values(by="test_roc_auc", ascending=False)
results_df
```

Out[32]:

	model	best_params	test_accuracy	test_f1	test_roc_auc
3	RandomForest	{'model__max_depth': None, 'model__min_samples...	0.973684	0.962963	0.995040
1	SVM	{'model__C': 10, 'model__gamma': 'scale', 'mod...	0.973684	0.962963	0.992725
0	LogReg	{'model__C': 10, 'model__penalty': 'l2', 'mode...	0.973684	0.963855	0.986442
2	KNN	{'model__n_neighbors': 11, 'model__weights': '...	0.947368	0.925000	0.982308

Detailed report for the top model:

In [33]:

```
top_name = results_df.iloc[0]["model"]
top_model = best_models[top_name]

y_pred = top_model.predict(X_test)
y_proba = top_model.predict_proba(X_test)[: , 1]

print("Top model:", top_name)
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification report:\n", classification_report(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

Top model: RandomForest
Confusion matrix:
[[72 0]
[3 39]]

Classification report:

	precision	recall	f1-score	support
0	0.96	1.00	0.98	72
1	1.00	0.93	0.96	42
accuracy			0.97	114
macro avg	0.98	0.96	0.97	114
weighted avg	0.97	0.97	0.97	114

ROC AUC: 0.9950396825396826

4) Two feature selection methods (most important features)

Method A — Filter method: SelectKBest (Mutual Information)

In [34]:

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# Impute first because feature selection can't handle NaNs
X_imputed = pd.DataFrame(
    SimpleImputer(strategy="median").fit_transform(X),
    columns=X.columns
```

```
)

selector = SelectKBest(score_func=mutual_info_classif, k=10)
selector.fit(X_imputed, y)

selected_mask = selector.get_support()
selected_features_A = X.columns[selected_mask]

scores_A = pd.Series(selector.scores_, index=X.columns).sort_values(ascending=False)

print("Top 10 (SelectKBest - mutual info):")
print(selected_features_A.tolist())
print("\nScores (top 15):")
print(scores_A.head(15))
```

Top 10 (SelectKBest - mutual info):
['radius_mean', 'perimeter_mean', 'area_mean', 'concavity_mean', 'concave points_mean', 'area_se', 'radius_worst', 'perimeter_worst', 'area_worst', 'concave points_worst']

Scores (top 15):

perimeter_worst	0.479811
area_worst	0.463845
radius_worst	0.452730
concave points_mean	0.440047
concave points_worst	0.438302
perimeter_mean	0.402659
concavity_mean	0.374096
radius_mean	0.365323
area_mean	0.360439
area_se	0.341233
concavity_worst	0.315709
perimeter_se	0.274328
radius_se	0.248348
compactness_worst	0.226539
compactness_mean	0.212721

dtype: float64

Method B — Wrapper method: RFE (Recursive Feature Elimination) with Logistic Regression

```
In [35]: from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# Scale + impute because RFE uses a model underneath
from sklearn.preprocessing import StandardScaler

X_imp = SimpleImputer(strategy="median").fit_transform(X)
X_scaled = StandardScaler().fit_transform(X_imp)

base_model = LogisticRegression(max_iter=5000, solver="lbfgs")
rfe = RFE(estimator=base_model, n_features_to_select=10)
rfe.fit(X_scaled, y)

selected_features_B = X.columns[rfe.get_support()]
print("Top 10 (RFE + Logistic Regression):")
print(selected_features_B.tolist())
```

Top 10 (RFE + Logistic Regression):
['concave points_mean', 'radius_se', 'area_se', 'compactness_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'concavity_worst', 'concave points_worst']

Compare overlap:

```
In [38]: set(selected_features_A).intersection(set(selected_features_B))
```

```
Out[38]: {'area_se',
'area_worst',
'concave points_mean',
'concave points_worst',
'perimeter_worst',
'radius_worst'}
```