**Exercise 5: Feature Normalization**

# Objective

Demonstrate the impact of various normalization schemes on multiple models using a high-dimensional, imbalanced, noisy dataset.

# Introduction

Normalization (feature scaling) strongly affects distance-based and margin-based learners. In high dimensions with heterogeneous feature scales, models like KNN and SVM can degrade without proper scaling, while tree ensembles are comparatively robust. This lab uses a tough synthetic dataset with: 200 features (mixed scales), class imbalance, redundant and repeated features, injected outliers and label noise.

# Dataset (Complex Synthetic)

We construct a dataset with 5,000 samples and 200 features using sklearn's make\_classification. We then amplify subsets of features by different factors (×0.1, ×10, ×100) and inject heavy-tailed noise to simulate outliers. Classes: 3 (60%, 30%, 10%), 30 informative, 30 redundant, 10 repeated, label noise 2%.

# Procedure

1) Generate dataset; perturb feature scales and inject outliers.  
2) Split into train/test.  
3) Build pipelines combining a scaler with a model.  
4) Evaluate with 5-fold Stratified CV using Accuracy and Macro-F1.  
5) Compare scalers: None, StandardScaler, MinMaxScaler, RobustScaler.  
6) Models: KNN, RBF-SVM, Multinomial Logistic Regression, Random Forest (as scale-robust baseline).

# Python Program

import numpy as np  
from sk learn.datasets import make\_classification  
from sklearn.model\_selection import StratifiedKFold, cross\_validate, train\_test\_split  
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler  
from sklearn.pipeline import Pipeline  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import make\_scorer, accuracy\_score, f1\_score  
  
# 1) Generate complex dataset  
X, y = make\_classification(  
 n\_samples=5000, n\_features=200, n\_informative=30, n\_redundant=30, n\_repeated=10,  
 n\_classes=3, weights=[0.6,0.3,0.1], flip\_y=0.02, random\_state=42  
)  
  
# Inject heterogeneous scales and outliers  
X[:, 0:50] \*= 0.1  
X[:, 50:100] \*= 10  
X[:, 100:150] \*= 100  
rng = np.random.RandomState(42)  
outlier\_cols = slice(150, 170)  
X[:, outlier\_cols] += rng.standard\_t(df=2, size=X[:, outlier\_cols].shape) \* 20  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, test\_size=0.2, random\_state=0)  
  
# 2) Define scalers and models  
scalers = {  
 "None": None,  
 "StandardScaler": StandardScaler(),  
 "MinMaxScaler": MinMaxScaler(),  
 "RobustScaler": RobustScaler()  
}  
  
models = {  
 "KNN(k=11)": KNeighborsClassifier(n\_neighbors=11),  
 "SVM(RBF)": SVC(kernel='rbf', C=10, gamma='scale'),  
 "LogisticRegression": LogisticRegression(max\_iter=3000, multi\_class='multinomial'),  
 "DecisionTree": DecisionTreeClassifier(max\_depth=None, random\_state=42)  
}  
  
cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=0)  
scoring = {'acc': make\_scorer(accuracy\_score), 'f1': make\_scorer(f1\_score, average='macro')}  
  
def evaluate\_combo(scaler\_name, scaler, model\_name, model):  
 steps = []  
 if scaler is not None:  
 steps.append(('scaler', scaler))  
 steps.append(('model', model))  
 pipe = Pipeline(steps)  
 cvres = cross\_validate(pipe, X\_train, y\_train, cv=cv, scoring=scoring, n\_jobs=-1)  
 return {  
 'Scaler': scaler\_name,  
 'Model': model\_name,  
 'CV\_Acc\_Mean': np.mean(cvres['test\_acc']),  
 'CV\_F1\_Mean': np.mean(cvres['test\_f1'])  
 }  
  
results = []  
for s\_name, scaler in scalers.items():  
 for m\_name, model in models.items():  
 results.append(evaluate\_combo(s\_name, scaler, m\_name, model))  
  
# Print sorted by Macro-F1  
results\_sorted = sorted(results, key=lambda d: d['CV\_F1\_Mean'], reverse=True)  
for r in results\_sorted:  
 print(r)

# Tasks

Task 1 [CO3] [BTL 3] [3 marks]  
Run the full script and paste the sorted CV table. Discuss why scaling helps KNN/SVM but barely changes Decision tree. Repeat with different scale perturbations (e.g., multiply columns 0:50 by 1000, 50:100 by 0.01). Report the most sensitive model-scaler pairs.

**Solution:**

|  |  |  |  |
| --- | --- | --- | --- |
| Scaler | Model | CV\_Acc\_Mean | CV\_F1\_Mean |
| MinMaxScaler | SVM(RBF) | **0.8440** | **0.7562** |
| StandardScaler | SVM(RBF) | 0.8396 | 0.7424 |
| RobustScaler | SVM(RBF) | 0.8090 | 0.7086 |
| None | SVM(RBF) | 0.7835 | 0.6951 |
| None | LogisticRegression | 0.7386 | 0.6298 |
| MinMaxScaler | LogisticRegression | 0.7390 | 0.6112 |
| StandardScaler | LogisticRegression | 0.7313 | 0.6167 |
| RobustScaler | LogisticRegression | 0.7305 | 0.6135 |
| StandardScaler | KNN(k=11) | 0.7393 | 0.5423 |
| MinMaxScaler | KNN(k=11) | 0.7412 | 0.5343 |
| None | DecisionTree | 0.6420 | 0.5286 |
| StandardScaler | DecisionTree | 0.6420 | 0.5286 |
| RobustScaler | DecisionTree | 0.6420 | 0.5286 |
| MinMaxScaler | DecisionTree | 0.6420 | 0.5286 |

The experimental results highlight the critical role of feature scaling in machine learning workflows, particularly for distance-based algorithms. Among all model–scaler combinations**, SVM with MinMaxScaler** achieved the best performance, delivering the highest cross-validation accuracy (≈0.84) and macro-F1 score (≈0.76). This confirms that when features exist on highly heterogeneous scales, MinMax normalization ensures that all attributes contribute proportionally to the model’s decision boundary.

A clear distinction was observed across algorithms regarding their sensitivity to scaling. **SVM and KNN**, both relying on distance metrics and geometric similarity in feature space, showed strong improvements once scaling was applied. Without normalization, their performance dropped substantially due to dominance of features with larger numeric ranges. In contrast, **Logistic Regression** showed only minor variations across scalers, as it is less reliant on raw distance but still benefits from normalization for numerical stability. **Decision Trees** remained completely unaffected by scaling, since tree-based methods split data based on thresholds rather than distances; hence, performance stayed constant across all scalers.

When feature perturbations were introduced (multiplying some feature groups by ×1000 and others by ×0.01), the sensitivity patterns became even clearer. **Unscaled SVM and KNN deteriorated severely**, confirming that these models are highly scale-dependent. However, once appropriate scalers were applied, their performance stabilized close to the original baseline. On the other hand, **Decision Trees retained identical results** under perturbations, further reinforcing their robustness to scale differences.

In summary, the study demonstrates that **scaling is indispensable for SVM and KNN, optional but beneficial for Logistic Regression, and irrelevant for Decision Trees.** The **most sensitive configuration** was found to be **SVM without any scaling,** while the **most robust and best-performing approach** overall was **SVM combined with MinMaxScaler.**  
  
Task 2 [CO3] [BTL 5] [2 marks]  
Augment the dataset with 5% extreme outliers in 30 random columns. Compare StandardScaler vs RobustScaler for SVM and Logistic Regression; justify using robust statistics.