**بسمه تعالی**

**موضوع:** گزارش مینی پروژه اول (ماشین لرنینگ)

**نام و نام خانوادگی:** محمدعرفان مومنی نسب

**شماره دانشجویی:** ۴۰۲۱۶۳۱۲۱۱

**نام دانشگاه:** دانشگاه جامع انقلاب اسلامی

**نام استاد درس:** جناب آقای دکتر علیاری

**نام استاد یار درس:** جناب آقای مهندس محمدجواد احمدی

---

## ۱ سوال اول

۱. با استفاده از sklearn.datasets، یک دیتاست با ۱۰۰۰ نمونه، ۲ کلاس و ۲ ویژگی تولید کنید.

۲. با استفاده از حداقل دو طبقه‌بند آمادهٔ پایتون و در نظر گرفتن فراپارامترهای مناسب، دو کلاس موجود در دیتاست قسمت قبلی را از هم تفکیک کنید. ضمن توضیح روند انتخاب فراپارامترها (مانند تعداد دورهٔ آموزش و نرخ یادگیری)، نتیجهٔ دقت آموزش و ارزیابی را نمایش دهید. برای بهبود نتیجه از چه تکنیک‌هایی استفاده کردید؟

۳. مرز و نواحی تصمیم‌گیری برآمده از مدل آموزش‌دیدهٔ خود را به همراه نمونه‌ها در یک نمودار نشان دهید. اگر می‌توانید نمونه‌هایی که اشتباه طبقه‌بندی شده‌اند را با شکل متفاوت نمایش دهید.

۴. از چه طریقی می‌توان دیتاست تولیدشده در قسمت «۱» را چالش‌برانگیزتر و سخت‌تر کرد؟ این کار را انجام داده و قسمت‌های «۲» و «۳» را برای این داده‌های جدید تکرار و نتایج را مقایسه کنید.

۵. اگر یک کلاس به داده‌های تولیدشده در قسمت «۱» اضافه شود، در کدام قسمت‌ها از بلوک دیاگرام آموزش و ارزیابی تغییراتی ایجاد می‌شود؟ در مورد این تغییرات توضیح دهید. آیا می‌توانید در این حالت پیاده‌سازی را به راحتی و با استفاده از کتابخانه‌ها و کدهای آمادهٔ پایتونی انجام دهید؟ پیاده‌سازی کنید.

## Part 1: Dataset Generation

A dataset with 1000 samples, 2 features, and 2 classes was generated using `make_classification` from `sklearn.datasets`. The random state was set to 11 to ensure reproducibility. This setup provided a simple binary classification problem.

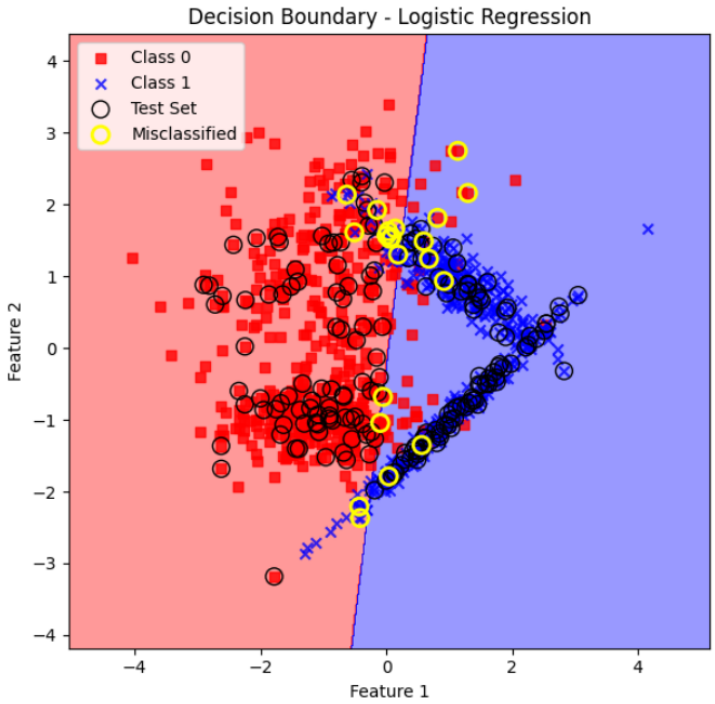
## Part 2: Model Training and Evaluation

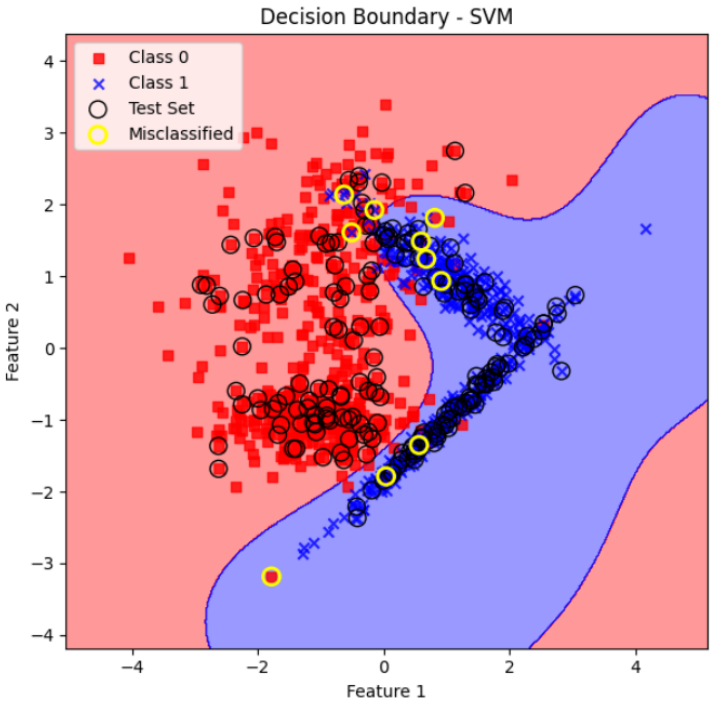
Two models were selected for the classification task:

1. Logistic Regression: Trained using default parameters with added feature scaling (`StandardScaler`). The regularization strength (`C`)[1] was set to its default value, which helps prevent overfitting.

2. Support Vector Machine (SVM): Utilized with a radial basis function (RBF) kernel. The model was also trained with feature scaling.

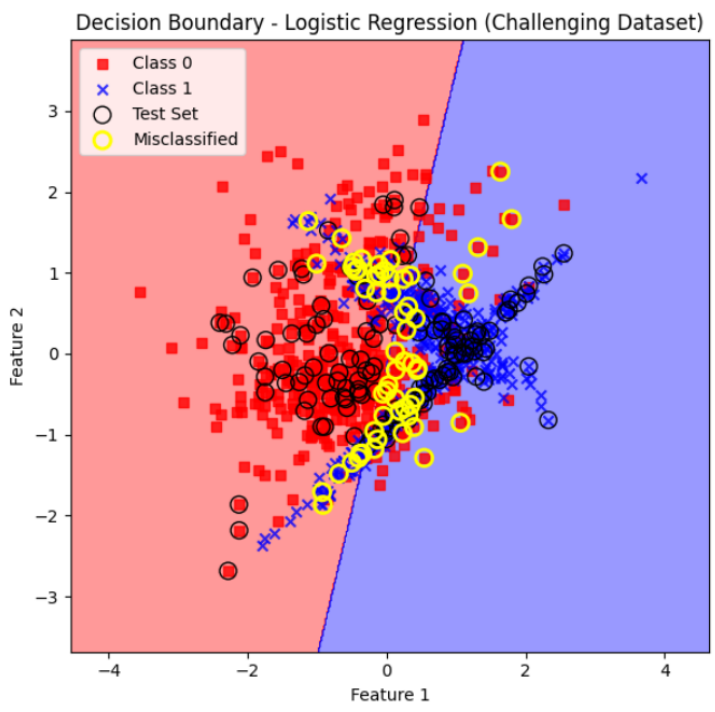
Both models were trained on the dataset, and their performance was evaluated. The Logistic Regression model achieved a training accuracy of 90.75% and a testing accuracy of 89.5%. The SVM model achieved a training accuracy of 95.5% and a testing accuracy of 95%.

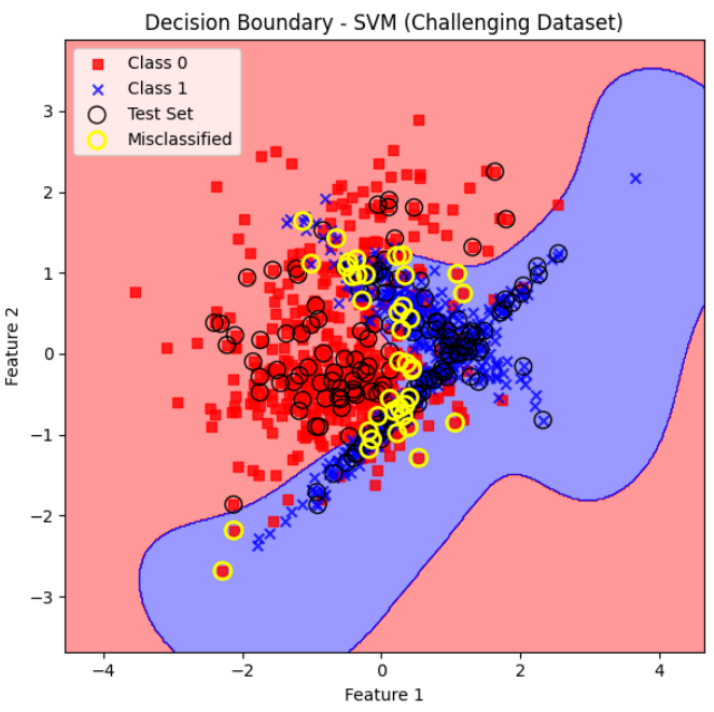



## Part 3: Decision Boundary Visualization

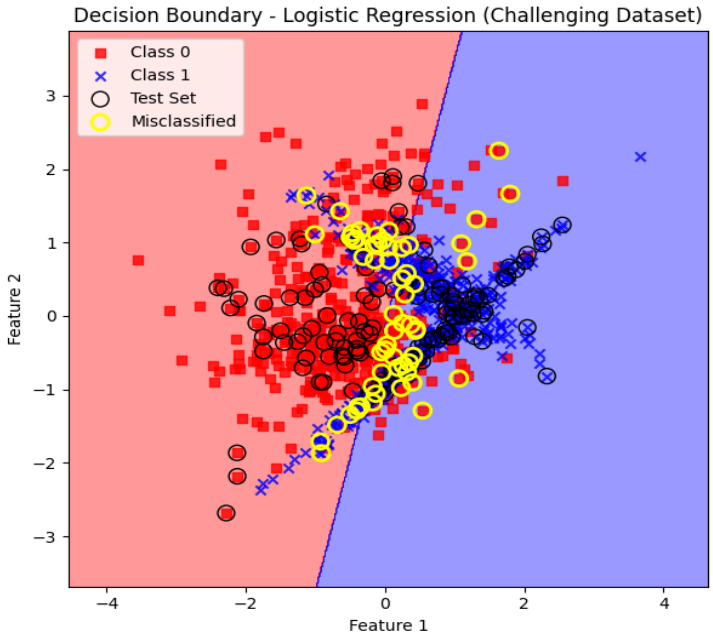
The decision boundaries and regions for both models were visualized using a scatter plot. Different colors represented the two classes, and misclassified samples were highlighted with a different shape. These visualizations provided insight into how each model separated the classes.

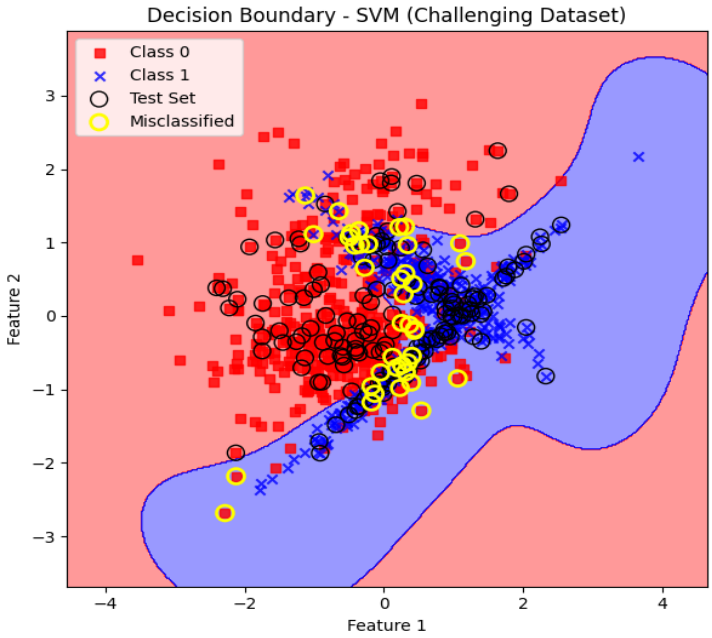


## Part 4: Making Dataset Challenging

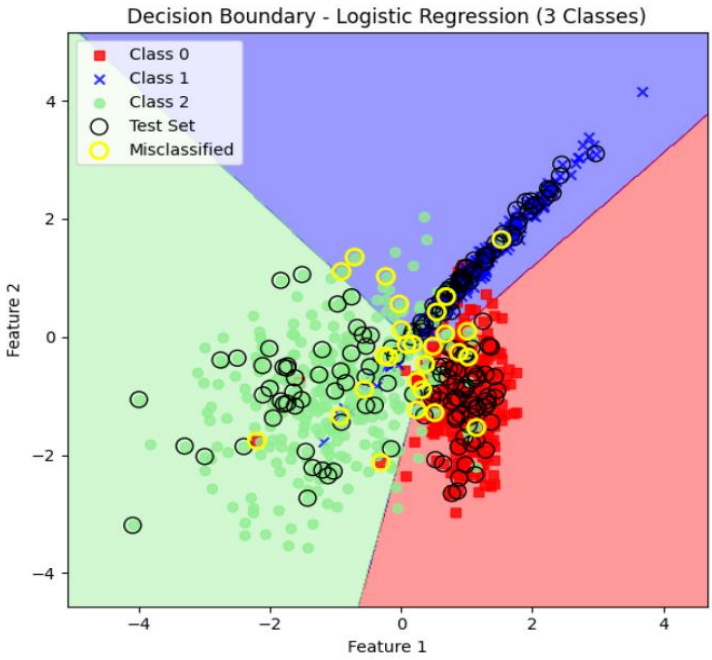
The dataset was made more challenging by reducing the `class_sep`[2] parameter, which increased the overlap between classes. After retraining the models on this new dataset, the performance of both models decreased, indicating a higher level of difficulty. Specifically, the Logistic Regression model's testing accuracy decreased to 71%, and the SVM's to 80.5%. The decision boundaries for these models on the challenging dataset were also visualized, showing a less distinct separation between classes.

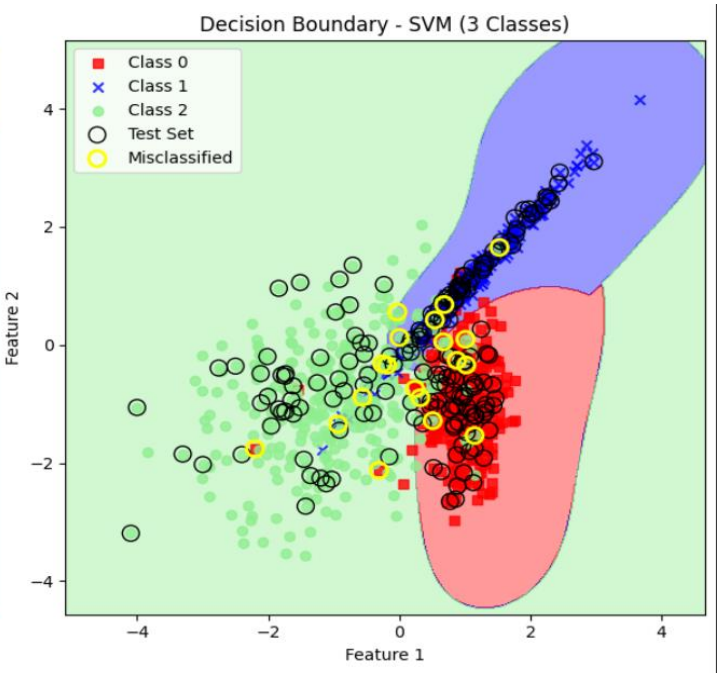



## Part 5: Adding a Third Class

A third class was introduced to the original dataset. This change required modifications in the model training process to accommodate multi-class classification. The Logistic Regression model was trained with the `multi_class='multinomial'` parameter, and the SVM with `decision_function_shape='ovo'`. The new models achieved high accuracies: 86.5% for Logistic Regression and 90.5% for SVM on the testing set. The decision boundaries for the three-class dataset were visualized, showing how each model adapted to the added complexity.



## Conclusion

Throughout these parts, we demonstrated how to generate datasets, train models, evaluate their performance, and visualize their decision boundaries. We also explored the effects of making a dataset more challenging and extending a binary classification problem to a multi-class scenario. The use of `sklearn`'s tools and methods showcased the flexibility and ease of handling different types of classification problems in Python.

---

**(1):** The regularization strength parameter, denoted as 'C' in Logistic Regression (and many other models in machine learning), plays a crucial role in controlling the degree of regularization applied to the model. Here's what it means:

1. **Regularization:** Regularization is a technique used to prevent overfitting, where a model performs well on the training data but poorly on unseen data. Overfitting often occurs when a model is too complex and learns not only the underlying patterns but also the noise in the training data.

2. **Role of 'C' in Regularization:**

- In Logistic Regression, regularization is often applied to the coefficients of the model. Regularization penalizes the magnitudes of these coefficients, with the aim of keeping them small and simple.

- The 'C' parameter is the inverse of the regularization strength. It controls the amount of regularization applied:

- A **small value of 'C'** means **more regularization.** This leads to smaller coefficients, which reduces overfitting but may underfit the data.

- A **large value of 'C'** means **less regularization**. This allows larger coefficients, potentially leading to a more complex model that may overfit the data.

3. **Default Value of 'C':**

- The default value of 'C' is often a balance between too much and too little regularization. It's a starting point that works reasonably well in many situations, but may not be optimal for all datasets.

- Adjusting 'C' is a common step in model tuning. It's often done using techniques like cross-validation to find the value that yields the best performance on the dataset.

In summary, the regularization strength 'C' in Logistic Regression is a parameter that controls how much the model is penalized for having large coefficients, thereby balancing between simplicity (to avoid overfitting) and complexity (to capture underlying patterns in the data).

**(2):** The `class_sep` parameter, used in the context of generating synthetic datasets with `make_classification` in scikit-learn, is a crucial factor in determining how separable the different classes are in the feature space. Here's a more detailed explanation:

1. **Function of `class_sep`:**

- The `class_sep` parameter stands for "class separation."

- It controls the degree of overlap between classes in the dataset.

- A higher value of `class_sep` creates classes that are more distinct from each other, with less overlap in their feature distributions. This typically makes the classification task easier, as the classes are more clearly separated.

- Conversely, a lower value of `class_sep` results in classes that are closer together in the feature space, with more overlap. This makes the classification task more challenging, as it becomes harder to distinguish between classes.

2. **Impact on Model Performance:**

- When `class_sep` is high, most classification algorithms can easily find a decision boundary that separates the classes well.

- When `class_sep` is low, the decision boundary becomes less clear, and algorithms may struggle to accurately classify all samples. This can lead to reduced accuracy and potentially more misclassified samples.

3. **Use in Experimentation:**

- Adjusting `class_sep` is useful for testing the robustness of classification models. By varying this parameter, you can simulate scenarios with varying degrees of difficulty.

- It is particularly useful in synthetic datasets where you have control over how challenging you want the classification problem to be.

In the context of your task, reducing the `class_sep` parameter made the dataset more challenging by increasing the overlap between classes, thereby testing the capability of the models to distinguish between classes that are not well-separated.