NYCU Introduction to Machine Learning, Homework 4

[111550149], [林悦揚]

Part. 1, Kaggle (70% [50% comes from the competition]):

(10%) Implementation Details

Model Architecture & Hyperparameters (5%):

ResNet18 with pretrain weight:

```python
class ResNet18(nn.Module):
    def __init__(self, num_classes=7):
        super(ResNet18, self).__init__()
        self.resnet = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
        self.resnet.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.resnet.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(self.resnet.fc.in_features, num_classes)
        )

    def forward(self, x):
        return self.resnet(x)
```

Bagging ensemble inference uses multiple pre-trained models, specifically ResNet18 models trained on Kaggle's dataset, to predict test data and generate final classification results based on probability averaging:

```python
def bagging_inference(weight_files, test_dir='./data/Images/test'):
    print(f"[INFO] Loading model weights from {len(weight_files)} models...")

    transform = transforms.Compose([
        transforms.ToPILImage(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    models = []
    for weight_file in weight_files:
        print(f"  Loading weights from '{weight_file}'...")
        model = ResNet18(num_classes=7)
        model.load_state_dict(torch.load(weight_file, map_location=device))
        model.to(device)
        model.eval()
        models.append(model)
    print("[INFO] All models loaded successfully.")
```

```python
print(f"[INFO] Loading test data from '{test_dir}'...")
filenames, X_test = load_test_data(test_dir, transform)
X_test = X_test.to(device)
print(f"[INFO] Loaded {len(filenames)} test images.")

print("[INFO] Starting Bagging inference...")
predictions = []
with torch.no_grad():
    for i, img in enumerate(X_test):
        img = img.unsqueeze(0)
        ensemble_outputs = []
        for model in models:
            output = model(img)
            ensemble_outputs.append(output.softmax(dim=1))

        avg_output = torch.mean(torch.stack(ensemble_outputs), dim=0)
        pred = avg_output.argmax(dim=1).item()
        predictions.append(pred)

        if (i + 1) % 10 == 0:
            print(f"  Processed {i+1}/{len(X_test)} images.")

submission = pd.DataFrame({
    'filename': filenames,
    'label': predictions
})
submission_file = 'submission_bagging.csv'
submission.to_csv(submission_file, index=False)
print(f"[INFO] Submission file '{submission_file}' generated successfully.")
```

hyperparameters:

```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

model = ResNet18(num_classes=num_classes)
model.to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

train_losses = []
val_accuracies = []

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0
    for batch_idx, (inputs, labels) in enumerate(train_loader):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
```

```
    val_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])
```

Model backbone (e.g., VGG16, VGG19, Custom, etc)	ResNet18（using PyTorch pre-trained weights:ResNet18_Weights.IMAGENET1K_V1）
Number of model parameters	11.7M
Other hyperparameters …	1.Learning Rate: 0.001 2.Optimizer: Adam 3.Loss Function: CrossEntropyLoss 4.Batch Size: 64 5.Epochs: Bagging Models 1 – 5: 30 Epochs Bagging Model 6: 20 Epochs 6.Dropout Rate: 0.5 7.Scheduler: none 8.Weight Initialization: Using PyTorch pre-trained weights for initialization 9.Normalization: Mean: [0.5, 0.5, 0.5] Std: [0.5, 0.5, 0.5]

Training Strategy (5%):

1. Bagging Models:

```python
def train_bagging_models(image_paths, labels, val_paths, val_labels, device):
    num_bagging_models = 5
    models = []
    for i in range(num_bagging_models):
        print(f"[INFO] Training Bagging Model {i+1}/{num_bagging_models}...")
        train_data, train_labels = bootstrap_sampling(image_paths, labels, len(image_paths))
        model = train_single_model(train_data, train_labels, val_paths, val_labels, model_idx=i+1, device=device,num_epochs=30)
        models.append(model)

    print(f"[INFO] Training Bagging Model 6 (without bootstrap)...")
    model = train_single_model(image_paths, labels, val_paths, val_labels, model_idx=num_bagging_models+1, device=device,num_epochs=20)
    models.append(model)

    return models
```

```python
weight_files = [
    './resnet18_bagging_1.pth',
    './resnet18_bagging_2.pth',
    './resnet18_bagging_3.pth',
    './resnet18_bagging_4.pth',
    './resnet18_bagging_5.pth',
    './resnet18_bagging_6.pth',
    './resnet18_bagging_6.pth',
    './resnet18_bagging_6.pth'
]
```

   - Bagging Models 1 – 5 trained on bootstrap sampled subsets of the training set.
   - Bagging Model 6 trained on the full training set without bootstrap sampling.
   - Adding Bag6 improves the ensemble's performance by leveraging the full dataset, enhancing feature coverage, reducing data bias, increasing diversity, and boosting generalization ability.
   - Bagging Model 6 with weight 3 has improved the accuracy.

2. Data Augmentation:

```python
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(15),
    transforms.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
val_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
```

   - RandomHorizontalFlip (p=0.5)
   - RandomRotation (±15°)
   - RandomResizedCrop (224×224, scale=(0.8, 1.0))
   - ColorJitter (Brightness, Contrast, Saturation, Hue)
   - Data augmentation is applied to increase the diversity of the training dataset and i mprove the model's robustness and generalization ability.

3. Validation:

```python
from sklearn.model_selection import train_test_split
train_paths, val_paths, train_labels, val_labels = train_test_split(
    image_paths, labels, test_size=0.2, random_state=42)
```

```python
model.eval()
val_preds, val_true = [], []
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        preds = outputs.argmax(dim=1)
        val_preds.extend(preds.cpu().numpy())
        val_true.extend(labels.cpu().numpy())

val_accuracy = accuracy_score(val_true, val_preds)
val_accuracies.append(val_accuracy)

print(f"[Model {model_idx}] Epoch {epoch+1}/{num_epochs} - Loss: {train_loss:.4f}, Val Accuracy: {val_accuracy:.4f}")
```

- Validation split: 20% of the training data (stratified sampling).
- Each epoch computes validation accuracy to monitor performance.

4. Load and clean data :

```python
def is_blurry(image_path, threshold=100):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    laplacian_var = cv2.Laplacian(image, cv2.CV_64F).var()
    return laplacian_var < threshold
```

```python
def load_and_clean_data(data_dir, threshold=100):
    print(f"[INFO] Loading and cleaning data from {data_dir}...")
    image_paths = []
    labels = []
    emotions = os.listdir(data_dir)
    for label, emotion in enumerate(emotions):
        folder_path = os.path.join(data_dir, emotion)
        for img_path in glob.glob(f"{folder_path}/*.jpg"):
            if not is_blurry(img_path, threshold):
                image_paths.append(img_path)
                labels.append(label)
    print(f"[INFO] Loaded {len(image_paths)} images after filtering.")
    return image_paths, labels
```

- The data loading and cleaning process filters out blurry images based on their Lapl acian variance to ensure only high-quality data is used for training, improving mod el performance and reliability.

5. Weight Saving:

```python
    torch.save(model.state_dict(), f'resnet18_bagging_{model_idx}.pth')
    print(f"[Model {model_idx}] Training complete. Model weights saved.")
    return model
```

- Saved each model's weights after training (resnet18_bagging_1.pth to resnet18_ba gging_6.pth).

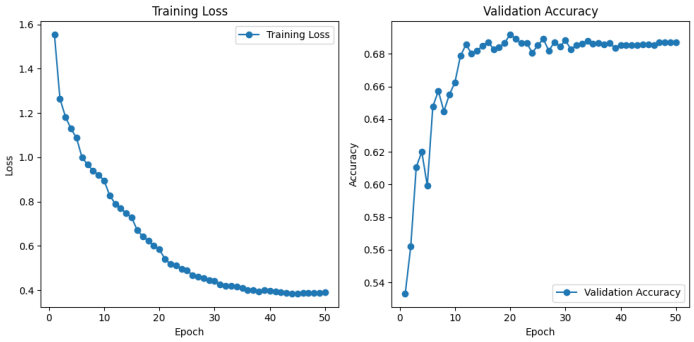
(10%) Experimental Results

Evaluation metrics and learning curve (5%):

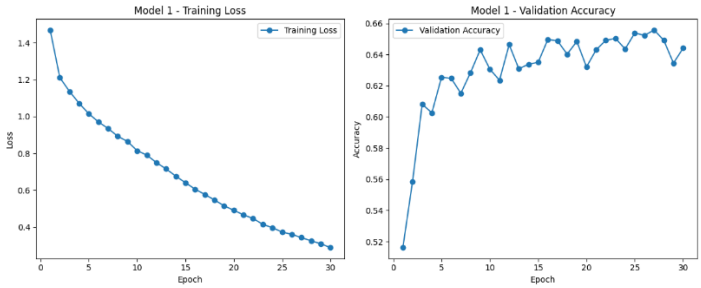
Evaluation metrics : Validation Accuracy

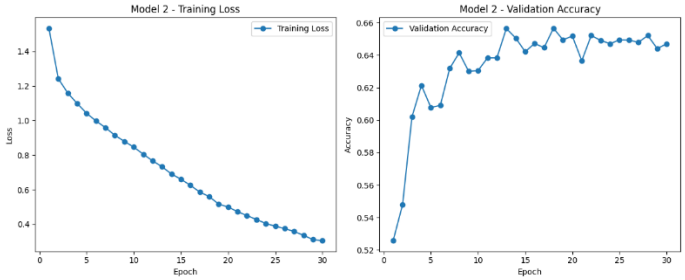
learning curve & Validation Accuracy:

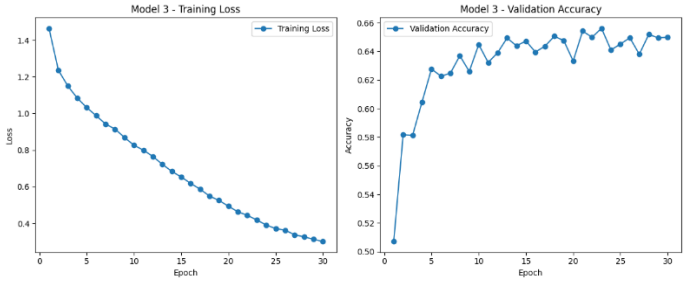
Resnet18 epoch =50:

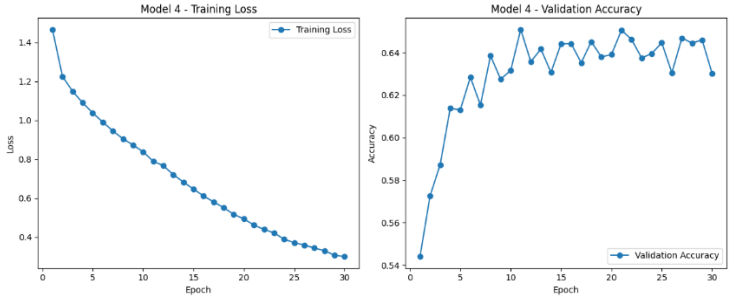


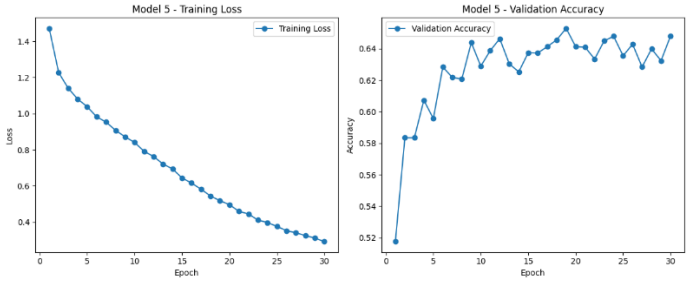
Resnet18 bagging model from 1~5 epoch=30 with bootstrap sampling.:



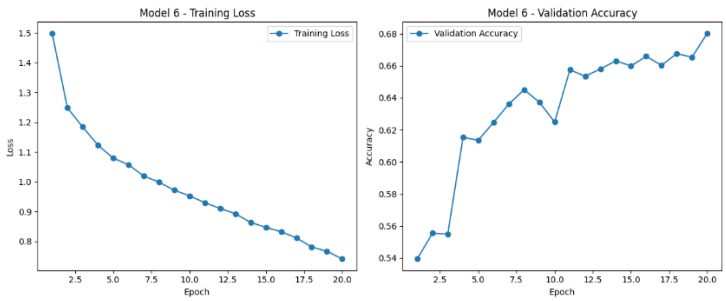




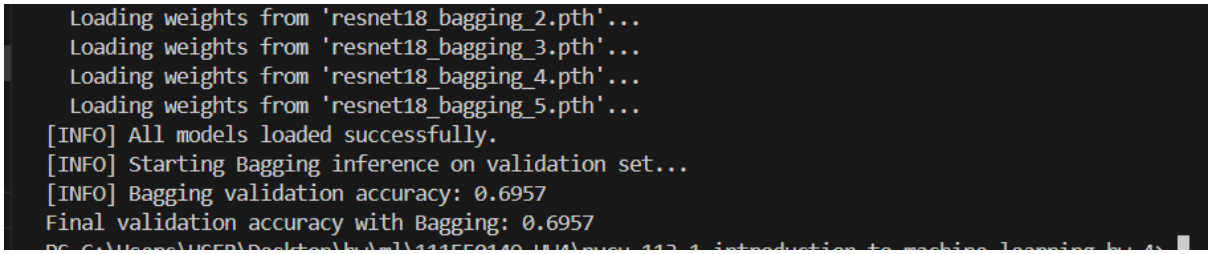




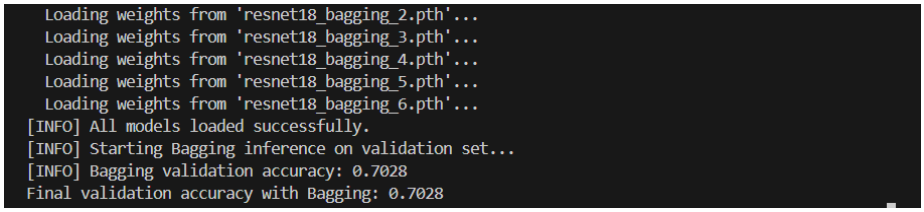
Resnet18 bagging model 6 epoch=20 without bootstrap sampling:



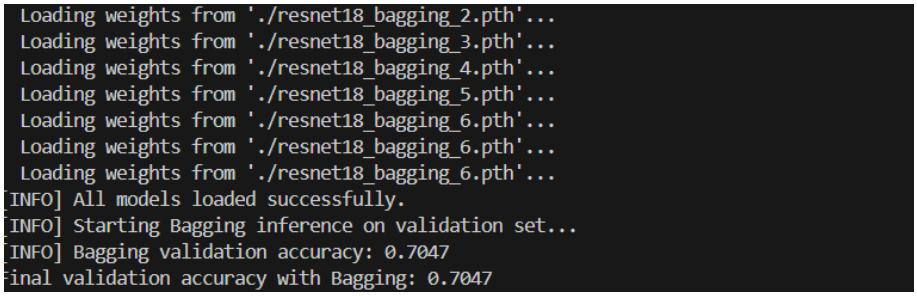
validation accuracy of the Bagging ensemble model from 1~5:



validation accuracy of the Bagging ensemble model from 1~6:



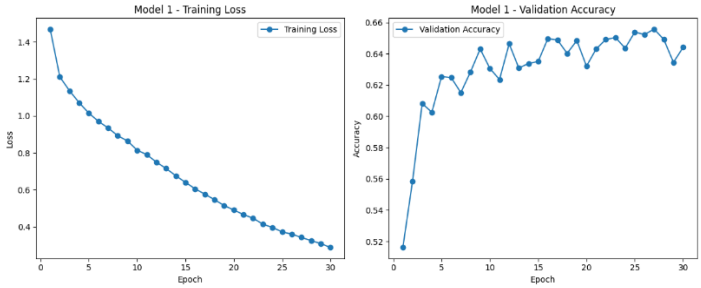
validation accuracy of the Bagging ensemble model from 1~6(6 with weight 3)

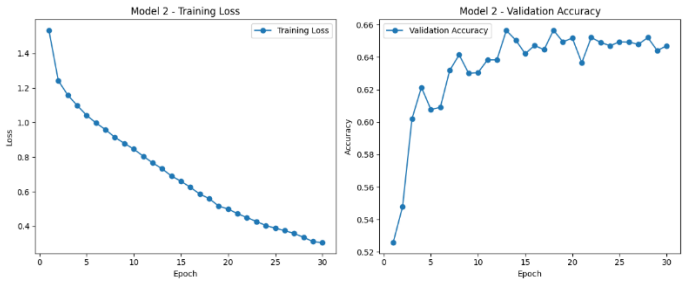


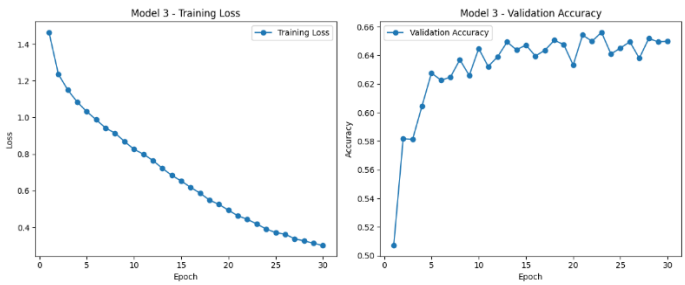
ResNet18 with 50 epochs showed the highest accuracy at epoch 20. Based on this observation, I aimed to further improve the accuracy. Therefore, I trained ResNet18 models (resnet18_bagging_1 to resnet18_bagging_5) using bootstrap sampling. Finally, these five models were combined with ResNet18_bagging_6 (trained for 20 epochs without bootstrap sampling) through bagging ensemble to enhance the overall accuracy.I further enchance the weight of the bag 6 to weight=3，can further improve the accuracy.

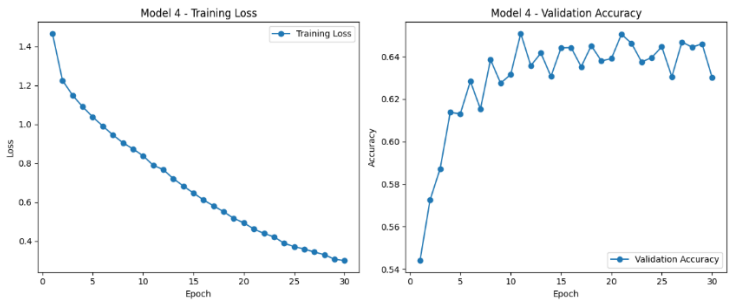
Ablation Study (5%):

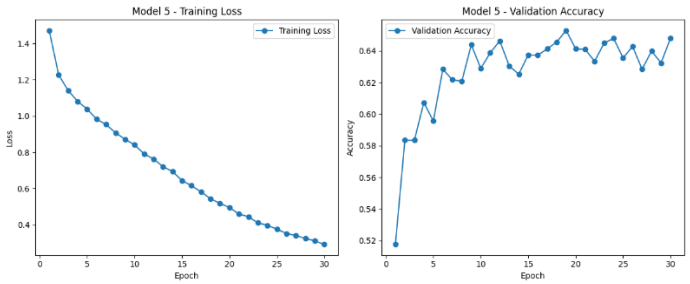
Resnet18 bagging model from 1~5 epoch=30 with bootstrap sampling.:



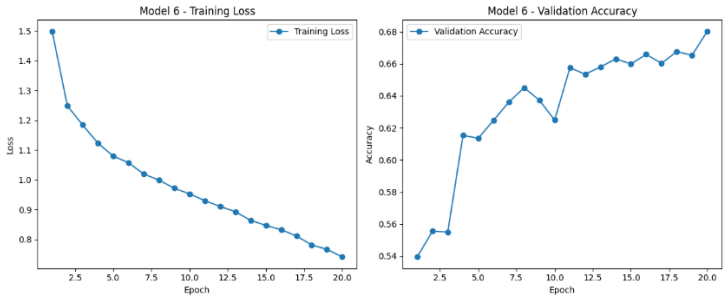








Resnet18 bagging model 6 epoch=20 without bootstrap sampling:



validation accuracy of the Bagging ensemble model from 1~5:

```
  model.load_state_dict(torch.load(weight_file, map_location=device))
  Loading weights from 'resnet18_bagging_2.pth'...
  Loading weights from 'resnet18_bagging_3.pth'...
  Loading weights from 'resnet18_bagging_4.pth'...
  Loading weights from 'resnet18_bagging_5.pth'...
[INFO] All models loaded successfully.
[INFO] Starting Bagging inference on validation set...
[INFO] Bagging validation accuracy: 0.6957
Final validation accuracy with Bagging: 0.6957
```

validation accuracy of the Bagging ensemble model from 1~6:

```
  model.load_state_dict(torch.load(weight_file, map_location=device))
  Loading weights from 'resnet18_bagging_2.pth'...
  Loading weights from 'resnet18_bagging_3.pth'...
  Loading weights from 'resnet18_bagging_4.pth'...
  Loading weights from 'resnet18_bagging_5.pth'...
  Loading weights from 'resnet18_bagging_6.pth'...
[INFO] All models loaded successfully.
[INFO] Starting Bagging inference on validation set...
[INFO] Bagging validation accuracy: 0.7028
Final validation accuracy with Bagging: 0.7028
```

validation accuracy of the Bagging ensemble model from 1~6(6 with weight 3)

```
Loading weights from './resnet18_bagging_2.pth'...
Loading weights from './resnet18_bagging_3.pth'...
Loading weights from './resnet18_bagging_4.pth'...
Loading weights from './resnet18_bagging_5.pth'...
Loading weights from './resnet18_bagging_6.pth'...
Loading weights from './resnet18_bagging_6.pth'...
Loading weights from './resnet18_bagging_6.pth'...
INFO] All models loaded successfully.
INFO] Starting Bagging inference on validation set...
INFO] Bagging validation accuracy: 0.7047
inal validation accuracy with Bagging: 0.7047
```

Based on the accuracy comparison above, using bootstrap sampling results in a decrease in accura cy. The accuracy of individual models (Bag1 to Bag5) trained with bootstrap sampling mostly falls around 0.64. When combining these models into a Bagging ensemble model, the accuracy improve s to 0.6957. On the other hand, the model trained without bootstrap sampling (Bag6), which utilize s the full dataset for training, achieves a higher accuracy of around 0.68 because it learns more co mprehensive features. By combining Bag6 with Bag1 to Bag5 in a Bagging ensemble model, the a ccuracy further increases to 0.7028. Further,Bag6 with weight 3,can further enhance the accuracy to 0.7047.Therefore, this combination is selected as the final implementation.

Part. 2, Questions (30%):

1. (10%) Explain the support vector in SVM and the slack variable in Soft-margin SVM. Plea se provide a precise and concise answer. (each in two sentences)

Support vectors : the data points closest to the decision boundary in a Support Vector Machine . T hey are critical because they determine the position and orientation of the hyperplane, maximizing the margin between classes.

Slack variables : Allow some data points to violate the margin or be misclassified in Soft-margin S VM. They provide a trade-off between maximizing the margin and minimizing classification error s, controlled by the regularization parameter.

2. (10%) In training an SVM, how do the parameter C and the hyperparameters of the kernel function (e.g., γ for the RBF kernel) affect the model's performance? Please explain their roles and describe how to choose these parameters to achieve good performance.

1. Parameter C:
   - Controls the trade-off between maximizing the margin and minimizing misclassification er rors.
   - Small C: Simpler model, may underfit.
   - Large C: Complex model, may overfit.
2. Kernel Hyperparameter (γ for RBF):
   - Determines the influence of each support vector.
   - Small γ : Smooth decision boundary, may underfit.
   - Large γ : Complex boundary, may overfit.

3. (10%) SVM is often more accurate than Logistic Regression. Please compare SVM and Lo gistic Regression in handling outliers.

Logistic Regression (LR):

Sensitive to outliers because all points contribute to the loss function.
Outliers can distort the decision boundary significantly.
Mitigation requires preprocessing (e.g., outlier removal) or regularization (L1/L2).
Support Vector Machines (SVM):

More robust to outliers, as only support vectors near the margin affect the decision boundary.
Soft-margin SVMs allow tolerance for misclassified points, reducing the influence of outliers.
Outliers close to the margin can still affect SVM performance.

Aspect	Logistic Regression	Support Vector Machines
Impact of Outliers	Strongly influenced; outliers distort t he decision boundary.	Less influenced; focuses only on sup port vectors.

Aspect	Logistic Regression	Support Vector Machines
Handling Mechanism	Uses regularization to limit the impact of extreme values.	Uses soft margins to tolerate margin violations.
When Outliers are Close to Decision Boundary	Sensitive, as all points contribute to loss.	Can become sensitive if outliers are support vectors.
Preprocessing Need	Requires careful preprocessing to handle outliers.	Less preprocessing required, but tuning CCC is important.
Scalability with Outliers	Computationally less impacted.	More computationally intensive if outliers are numerous.