Week 6 Peer Review

support_vectors=[(6, 3), (5, 2), (-5, 0), (-5, 1)]
In-Bound Support Vectors: [(5, 2)]
Out-Bound Support Vectors: [(6, 3), (-5, 0), (-5, 1)]

The support vectors lie on or within the margin or are misclassified, as determined by their proximity to the decision boundary and margin lines in the SVM. The in-bound support vectors lie exactly on the margin, such as (-5,0) and (5,2). The out-bound support vectors lie within the margin, such as (-5,1) and (6,3).

In an SVM, points with nonzero slack are typically out-bound support vectors or points that violate the margin due to the soft margin. The outbound support vectors (-5,1) and (6,3) are the only ones with nonzero slack because they are on the wrong side of their support vector boundaries. Nonzero slack is determined by computing the decision function for each candidate point. I had to check the functional margin to see if it's less than 1.

As C increases from 0.1 to 10, the margin narrows, and the decision boundary shifts to classify more training points correctly, reducing slack. The red point remains on the wrong side across all plots, but the boundary adjusts to minimize its impact less as C grows, showing the trade-off between margin size and error penalty. The blue points are progressively better separated, with fewer violations as C increases, indicating tighter fitting to the training data. The choice of C directly affects the trade-off between margin width and classification errors in the SVM. A smaller C favors a larger margin and generalization, while a larger C favors tighter fitting to the training data, at the cost of overfitting.

Hinge loss is less sensitive to outliers because it applies a linear penalty, making it more robust for noisy datasets but potentially less precise for fitting training data perfectly. Squared hinge loss is more sensitive to outliers due to the quadratic penalty, which can lead to overfitting if C is large, but it can improve performance on clean datasets by penalizing large errors more heavily. Hinge loss is computationally simpler. Squared hinge loss can lead to a more complex optimization problem due to the quadratic term, but it can converge faster in some cases or produce a different solution that better fits the data for certain C values. With C=3 in both plots, the effect of the loss function is subtle because (C) moderates the penalty. A smaller C would amplify the difference, with hinge loss allowing more violations and squared hinge loss penalizing them more severely. A larger C would make squared hinge loss more prone to overfitting compared to hinge loss.

A small value of C increases bias, leading to underfitting, while decreasing variance, which promotes better generalization. This approach prioritizes maintaining a larger margin over perfectly fitting the training data. In contrast, a large C decreases bias, reducing underfitting, but increases variance, which can lead to potential overfitting. This setting focuses on fitting the training data closely, even if it results in a smaller margin. A moderate C strikes a balance between bias and variance, aiming for optimal generalization by fitting the training data reasonably well while preserving a margin that helps prevent overfitting.

Begin by selecting a broad range of C values on a logarithmic scale, such as 0.001, 0.01, 0.1, 1, 10, 100, and 1000, to explore how C impacts accuracy. This approach is common because C's effect is

exponential, and small changes on a log scale can significantly affect results. The provided results already include values like 0.01, 0.1, 1, 10, and 100, offering a solid starting point. Use k-fold cross-validation, typically with k = 5 or 10, to estimate validation accuracy for each C value, ensuring the results reflect generalization performance rather than just training accuracy. In each fold, split the data into training and validation sets, calculate the mean accuracy across folds, and use this to evaluate C. Based on initial results, refine the range around the best-performing C values. For instance, if C values of 1, 10, and 100 all achieve 0.900 accuracy, test intermediate values like 0.5, 2, 5, 20, and 50 to fine-tune the model. If accuracy drops or plateaus at lower C values (e.g., 0.01 or 0.1 at 0.700), shift focus to higher C values where performance improves. Consider data characteristics, such as noise, outliers, and class separation. A smaller C can help avoid overfitting in noisy datasets by increasing bias and reducing variance, while a larger C can improve accuracy in cleaner datasets by lowering bias and increasing variance. From earlier analysis, a moderate C value like 1 or 10 balances margin size and errors, achieving 0.900 accuracy. To evaluate overfitting or underfitting, compare training and validation accuracy for each C. A significant drop in validation accuracy at high C values (e.g., C = 100) indicates overfitting, while low accuracy at smaller C values (e.g., 0.01 or 0.1) suggests underfitting. In this case, no overfitting is evident at C = 100, but C = 1 is selected for simplicity and stability. Document the mean and standard deviation of accuracy for each C to assess consistency, and if results vary significantly, repeat the process with different random seeds or folds to ensure robustness. A small C, such as 0.01 or 0.1, imposes a low penalty for misclassifications, prioritizing a larger margin and resulting in a wider decision boundary that allows more points to be misclassified or lie within the margin. This creates a smoother, more general boundary less influenced by outliers or noise, leading to higher bias and underfitting but lower variance and better generalization. In contrast, a moderate C, like 1, balances margin size and classification accuracy, leading to a narrower margin that adjusts to reduce misclassifications while maintaining generalization. This results in lower bias and moderate variance, achieving good training accuracy around 0.900. A large C, such as 10 or 100, heavily penalizes misclassifications, leading to a tighter margin and a decision boundary that closely fits the training data, potentially overfitting by being sensitive to outliers and noise. While it reduces bias, it increases variance and the risk of overfitting. The best choice, C = 1, achieves 0.900 accuracy like C = 10 and C = 100 but avoids the potential overfitting of higher C values while improving over the underfitting seen with smaller C values. It offers an optimal balance between bias and variance, generalizes well across cross-validation, and maintains a reasonable margin while reducing slack, as observed in the plots.

A small Gamma, like 0.01, results in a larger radius of influence, creating a smoother decision boundary with better generalization, leading to low variance but potentially higher bias. In contrast, a large Gamma, such as 10 or 100, focuses more on nearby points, increasing variance and the risk of overfitting while reducing bias. The combination of C equal to 1 and Gamma set to 0.01 achieving perfect accuracy indicates a well-tuned model with a linear or near-linear decision boundary, effectively fitting the data in cross-validation.

The choice of kernel function in a Support Vector Machine (SVM) significantly affects the model’s bias and variance by determining how input data is mapped into a higher-dimensional feature space for an optimal decision boundary. A linear kernel has high bias and low variance, which may lead to underfitting for non-linear data but generalizes well for linear data. The polynomial kernel's bias and variance depend on its degree, with a low degree resulting in high bias and low variance, and a high degree leading to low bias and high variance. The RBF kernel's bias and variance are influenced by gamma, where a small

gamma leads to high bias and low variance, and a large gamma results in low bias and high variance. The sigmoid kernel typically has moderate to high bias and low to moderate variance, but it can be unstable.

```
# ============================

# Define parameter grid over log range (2^-5 to 2^5)
param_grid = {
    'C': [2 ** i for i in range(-5, 6)],         # C values: 2^-5 to 2^5
    'gamma': [2 ** i for i in range(-5, 6)]      # gamma values: 2^-5 to 2^5
}

# Initialize the SVM model
svm = SVC(kernel='rbf')

# Set up GridSearchCV with 3-Fold Cross-Validation
grid = GridSearchCV(svm, param_grid, cv=3, scoring='accuracy', verbose=1)

# Run the grid search
grid.fit(X, y)

# Get the best parameters and best score
best_params = grid.best_params_
best_score = grid.best_score_

# Train the model with the best parameters
best_model = grid.best_estimator_

# Display results
print("\nBest Parameters from Grid Search:")
print("C =", best_params['C'])
print("Gamma =", best_params['gamma'])
print("Best Cross-Validation Accuracy: {:.3f}".format(best_score))

# Plot decision boundary for best model
nonlinear_plot(X, y, best_model)
```

```
Fitting 3 folds for each of 121 candidates, totalling 363 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 363 out of 363 | elapsed:    0.7s finished

Best Parameters from Grid Search:
C = 0.5
Gamma = 0.03125
Best Cross-Validation Accuracy: 1.000
```


To identify high-accuracy regions on the heat map, I focused on the brightest areas, indicating validation accuracies close to 1.0, which suggest strong generalization across cross-validation folds. I traced these regions to the corresponding C and gamma values on the axes and found that the highest accuracies were achieved at {C=8, gamma=8}, {C=16, gamma=8}, and {C=32, gamma=2}. Analyzing trade-offs, I noticed that as C increases, accuracy drops, especially with higher gamma, indicating overfitting due to high variance. Similarly, increasing gamma results in a more complex decision boundary, leading to reduced generalization and lower accuracy, particularly for higher C values.

These high-accuracy regions suggest that moderate to high C values (8–32) paired with appropriately chosen gamma values (2–8) optimize the trade-off between bias and variance. This aligns with SVM principles, where moderate C avoids excessive regularization while ensuring proper margin separation, and appropriately chosen gamma values in an RBF kernel produce a decision boundary that generalizes well. Thus, the optimal combination was determined by using the heat map to find the highest validation

accuracy, balancing bias and variance effectively

```
# Extract best parameters and best accuracy
best_params = grid.best_params_
best_score = grid.best_score_

# Print results
print("Best Parameters Found by Grid Search:")
print("C =", best_params['C'])
print("Gamma =", best_params['gamma'])
print("Best Cross-Validation Accuracy: {:.3f}".format(best_score))
```

```
Best Parameters Found by Grid Search:
C = 0.5
Gamma = 0.03125
Best Cross-Validation Accuracy: 1.000
```

```
: # print best paramters and best accuracy
  # your code here
  print(grid.best_params_)
  print(grid.best_score_)
  nonlinear_plot(X, y, grid.best_estimator_)
```

```
{'C': 8.0, 'gamma': 8.0, 'kernel': 'rbf'}
0.9433333333333334
```

The best decision boundary in this plot is a smooth, non-linear curve created by the RBF kernel with gamma set to 0.03125. It perfectly separates the red points at (5, 2) and (6, 3) from the blue points at (-3, 1) and (-2, 1). With C set to 0.5, the margin is moderately wide, achieving low bias by fitting the data well and low variance by generalizing perfectly, as indicated by the 1.000 accuracy. This boundary is robust to noise, effectively handles non-linear separability, and reflects the optimal balance identified by GridSearchCV, consistent with earlier perfect classification results.