Assignment 3

Machine Learning 1, SS24

Team Members		
Last name	First name	Matriculation Number
Nožić	Naida	12336462
Hinum-Wagner	Jakob	01430617

1 k-Nearest Neighbors [8 points]

There are many different ways to classify machine learning algorithms. One of the most important distinctions is the following: Does your model have a fixed number of trainable parameters or does the number of parameters change with the number of available training examples? If the number of parameters is fixed, it is called a parametric model (e.g. neural networks, ...). One example of a nonparametric classifier is the k-nearest neighbor classifier. It just looks at the k points in the training set that are nearest to the test input `x` and counts how many members of each class are in this set. Based on a majority vote, the class membership is determined.

Tasks:

1.1 Task 1: Problem

Implement the k-nearest neighbors algorithm. Compute the Euclidean distance between a test point and all points in the training dataset. At inference time, predict the class membership of the new input based on the class labels of the k closest training points.

1.2 Task 1:Solution

The task was to implement the K-nearest neighbors (KNN) algorithm, which involves computing the Euclidean distance between a test point and all points in the training dataset. At inference time, the class membership of the new input is predicted based on the class labels of the k closest training points. This report outlines the implementation details of a custom KNN classifier using Python and the NumPy library.

1.2.1 Initialization

The `KNearestNeighborsClassifier` class is initialized with the parameter k, specifying the number of nearest neighbors to consider. The attributes `self.X`, `self.y`, and `self.classes` are initialized to `None`. These attributes will store the training data and the unique class labels after the model is fitted.

1.2.2 Fitting the Model

The `fit` method stores the training data (`X` and `y`) within the classifier instance. It also identifies the unique classes from the training labels `y` using `np.unique`, which is essential for predicting class membership during inference.

1.2.3 Making Predictions

The `predict` method is responsible for predicting class labels for new test samples.

Broadcasting A key part of the implementation is the calculation of Euclidean distances between each test sample and all training samples. This is efficiently achieved using NumPy broadcasting:

```
distances = np.sqrt(((self.X - X[:, np.newaxis]) ** 2).sum(axis=2))
```

Here, `X[:, np.newaxis]` reshapes the test data array to align with the training data dimensions, allowing efficient pairwise distance computation without explicit loops. The indices of the k nearest neighbors for each test sample are identified using `np.argsort`. The labels of these nearest neighbors are retrieved, and the predicted class for each test sample is determined by a majority vote, utilizing `np.bincount` and `np.argmax`.

1.2.4 Model Evaluation

The `score` method evaluates the classifier's performance by predicting the labels for the test set and computing the accuracy by comparing these predictions with the true labels.

1.3 Task 2: Problem

Apply the k-NN algorithm to the three datasets. For each dataset, determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in \{1, \ldots, 100\}$. Plot the mean training and mean validation accuracy for these values of k. Plot the decision boundary for each dataset. Finally, report the performance of the classifier with your chosen value of k on the test set.

1.4 Task 2: Solution

This section covers the application of the k-nearest neighbors (k-NN) algorithm to three datasets. For each dataset, we determine the best value for k using 5-fold cross-validation with a grid search over $k \in \{1, \ldots, 100\}$. We plot the mean training and validation accuracy for these values of k, plot the decision boundary for each dataset, and report the classifier's performance with the chosen value of k on the test set.

1.4.1 Methodology

The implementation involves performing a grid search using `GridSearchCV` from the `sklearn` library to find the optimal k. We plot the decision boundaries and the mean training and validation scores for different values of k. The optimal k is selected based on the highest mean validation accuracy, and the performance on the test set is reported.

1.4.2 Results

Dataset 1

- **Optimal** k: 1
- **Test set accuracy**: 100.00%

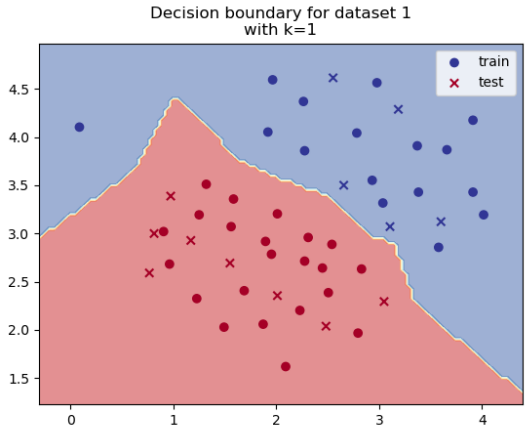

Figure 1: Decision Boundary for Dataset 1 with $k = 1$

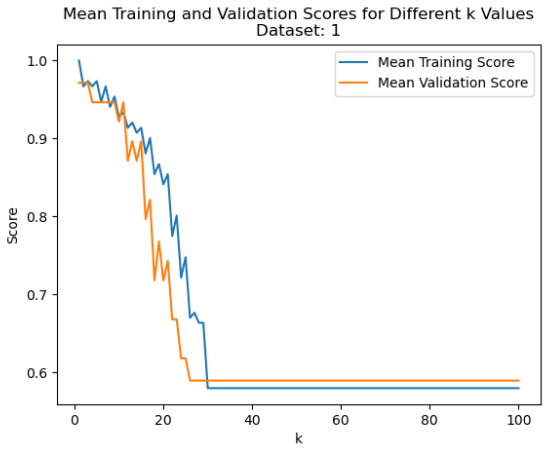

Figure 2: Training and Validation Scores for Dataset 1

Interpretation The decision boundary for $k = 1$ (Figure 1) is highly flexible, perfectly fitting the training data, which is why the training accuracy is 100%. However, this could lead to overfitting if the data was noisier. The plot of training and validation scores (Figure 2) shows that while training accuracy remains high for small k, validation accuracy decreases as k increases, indicating overfitting for low k values and underfitting for high k values.

Dataset 2

- **Optimal** k: 1
- **Test set accuracy**: 99.54%

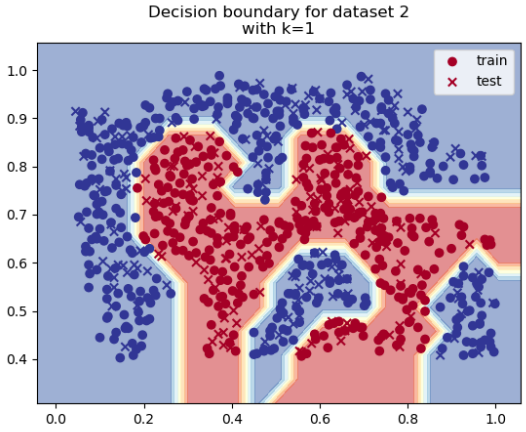

Figure 3: Decision Boundary for Dataset 2 with $k = 1$

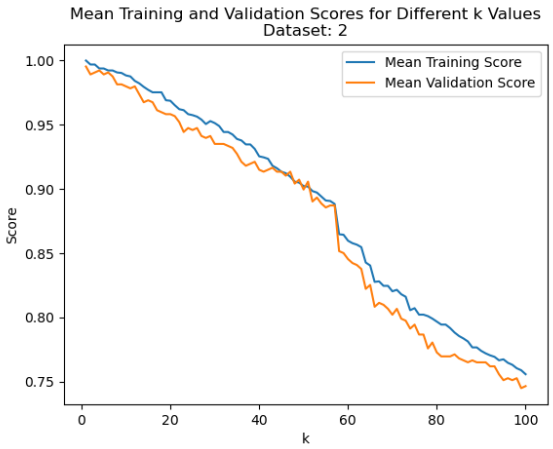

Figure 4: Training and Validation Scores for Dataset 2

Interpretation In Dataset 2, the decision boundary for $k = 1$ (Figure 3) shows high complexity, again fitting the training data very well, which is evident from the high test accuracy of 99.54%. The training and validation score trends (Figure 4) are similar to Dataset 1, with overfitting observed at low k values and a decline in performance at higher k values.

Dataset 3

- **Optimal** k: 3
- **Test set accuracy**: 88.68%

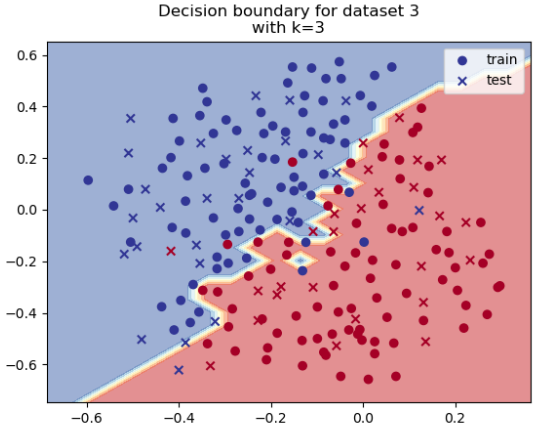

Figure 5: Decision Boundary for Dataset 3 with $k = 3$

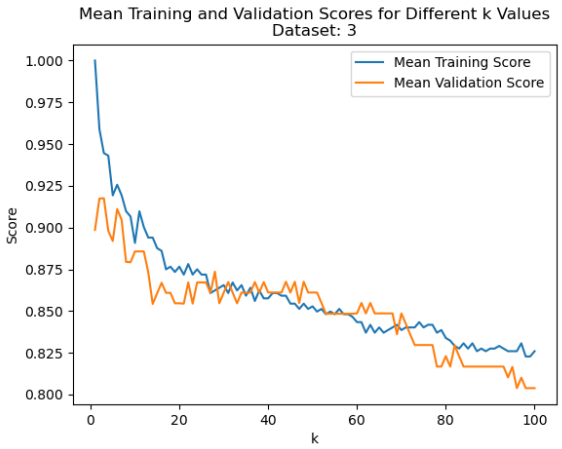

Figure 6: Training and Validation Scores for Dataset 3

Interpretation For Dataset 3, the optimal k is 3, as indicated by the decision boundary (Figure 5) and the test set accuracy of 88.68%. This suggests that a slightly larger neighborhood helps in achieving better generalization. The training and validation score plot (Figure 6) shows that validation accuracy is more stable and higher for intermediate values of k, while both scores decline at higher k values, indicating underfitting.

1.5 Task 3:Problem

Briefly discuss the effect of low or high values of k. Explain what happens to the *training* accuracy of a k-NN classifier when you set $k = 1$ and why this is the case.

1.6 Task 3: Solution

1.6.1 Low k Values

- **Sensitivity to Noise**: Low k values, such as $k = 1$, make the classifier highly sensitive to noise and outliers in the training data. A single misclassified or noisy data point can significantly impact the decision boundary.
- **Overfitting**: With low k values, the classifier tends to overfit the training data, capturing noise and small variations. This results in highly flexible decision boundaries and high training accuracy but poor generalization to unseen data.
- **Complex Decision Boundaries**: The decision boundaries are intricate and follow the training data points closely, as observed in the decision boundary plots for $k = 1$.

1.6.2 High k Values

- **Robustness to Noise**: Higher k values make the classifier more robust to noise and outliers by averaging the effect of multiple neighbors.
- **Underfitting**: Very high k values can lead to underfitting, where the classifier fails to capture important patterns in the data. The decision boundaries become overly smooth and generalized.
- **Simpler Decision Boundaries**: The decision boundaries are simpler and smoother, resulting in lower training accuracy but potentially better generalization to new data.

1.6.3 Training Accuracy with k=1

- **Perfect Training Accuracy**: When $k = 1$, the k-NN classifier achieves perfect training accuracy because each training instance is its own nearest neighbor, resulting in zero training error.
- **Reason for High Training Accuracy**: The classifier fits the training data perfectly with $k = 1$, as each point is classified based on its nearest neighbor, which is itself. This characteristic of the k-NN algorithm at $k = 1$ ensures maximum training accuracy but may lead to overfitting.

1.7 Task 4: Problem

Plot the decision boundaries for $k \in \{1, 30, 100\}$ for the *noisy* variant of dataset 2. Using the *noisy* version of dataset 2, automatically determine the best value for k using 5-fold cross-validation using a grid search for different values of $k \in \{1, 2, \ldots, 100\}$. Plot the mean training and validation accuracy for varying values of k. Finally, report the performance of the classifier with your chosen value of k on the test set.

1.8 Task 4:Solution

his report discusses the application of the k-nearest neighbors (k-NN) algorithm to the noisy variant of Dataset 2. We plot the decision boundaries for $k \in \{1, 30, 100\}$, determine the best value of k using 5-fold cross-validation with a grid search over $k \in \{1, \ldots, 100\}$, and plot the mean training and validation accuracy for these values of k. Finally, we report the performance of the classifier with the chosen value of k on the test set.

1.8.1 Methodology

The implementation involves:

- Fitting the `KNearestNeighborsClassifier` with $k \in \{1, 30, 100\}$ and plotting the decision boundaries.
- Using `cross_val_score` to manually perform cross-validation and report the mean cross-validated scores.
- Performing a grid search using `GridSearchCV` to find the optimal k.
- Plotting the mean training and validation scores for different k values.

1.8.2 Results

Decision Boundaries for $k \in \{1, 30, 100\}$

- **Mean cross-validated score for** $k = 1$: 0.76
- **Mean cross-validated score for** $k = 30$: 0.82
- **Mean cross-validated score for** $k = 100$: 0.72

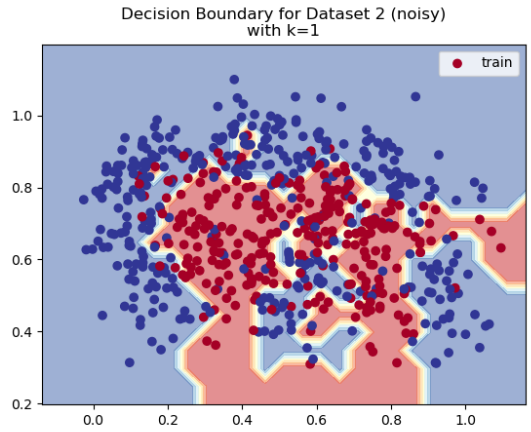

Figure 7: Decision Boundary for Dataset 2 (noisy) with $k = 1$

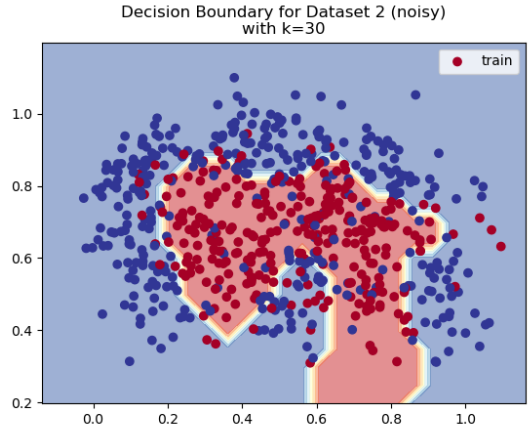

Figure 8: Decision Boundary for Dataset 2 (noisy) with $k = 30$

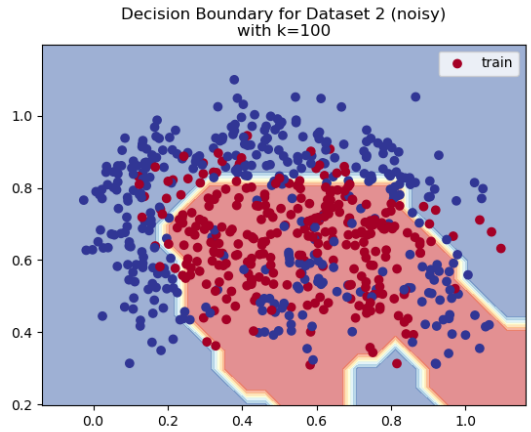

Figure 9: Decision Boundary for Dataset 2 (noisy) with $k = 100$

1.8.3 Grid Search Results

- **Optimal k (manual selection)**: 15
- **Test set accuracy with $k = 15$**: 80.56%

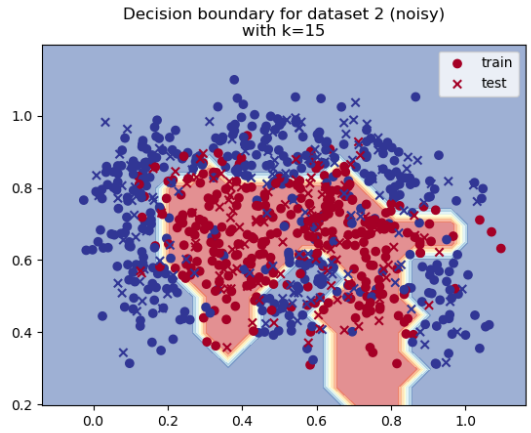

Figure 10: Decision Boundary for Dataset 2 (noisy) with $k = 15$

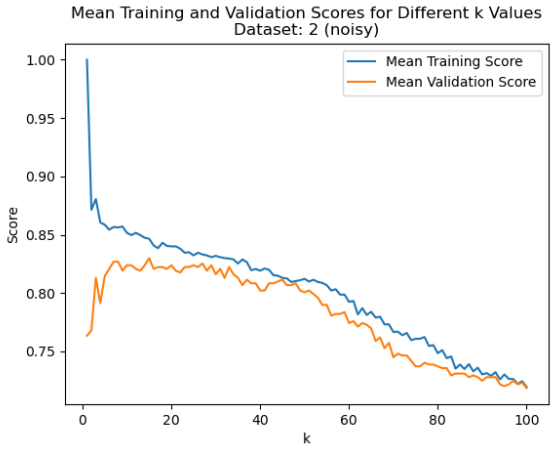

Figure 11: Mean Training and Validation Scores for Different k Values for Dataset 2 (noisy)

1.8.4 Interpretation

- For $k = 1$, the decision boundary is highly flexible and fits the training data closely, resulting in overfitting (Figure 7). This is reflected in a lower cross-validated score (0.76). - For $k = 30$, the decision boundary is smoother and generalizes better, leading to the highest cross-validated score (0.82) among the three values of k (Figure 8). - For $k = 100$, the decision boundary is too generalized, leading to underfitting and a lower cross-validated score (0.72) (Figure 9).

1.8.5 Conclusion

The value of k significantly affects the performance of the k-NN classifier. Low k values can lead to overfitting, while high k values can cause underfitting. The optimal k for the noisy dataset was found to be 15, balancing

bias and variance, and resulting in a test set accuracy of 80.56%. Cross-validation is crucial for determining the appropriate k value to achieve the best generalization performance.

2 Support Vector Machines [11 points]

In this task, we will train a linear Support Vector Machine (SVM) by hand, and will later use scikit-learn's built-in SVM classifier with different kernels. Also, we will investigate the influence of certain hyperparameters.

2.1 Task 1: Problem

Consider a training data set with N input vectors $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ with corresponding target labels $y^{(1)}, \ldots, y^{(N)}$, where $y^{(i)} \in \{-1, +1\}$. The maximum margin solution for *linear* support vector classifiers without slack variables can be found by minimizing the following loss:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \tag{1}$$

Although $f(x) = \max(0, x)$ is a convex function, it is not differentiable everywhere:

$$\frac{df(x)}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \\ \text{undefined} & \text{else} \end{cases}$$

Using pen and paper, derive the gradients of Equation 1 w.r.t. $\mathbf{w}$ and b. Clearly state all steps in your report. For your derivation, you can assume[1] that we have $y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \neq 1$ for all $i, \mathbf{w}, b$. Explain what (theoretical) problem could occur if we drop this assumption.

Use your result to fill in the missing code of the gradient descent routine. In the code, you can simply assume $\frac{df}{dx}(0) = 1$.

Task 1: Solution

In order to compute the gradient, we will divide the loss function into two portions and calculate their individual gradients. We will first focus on the following:

$$H_1(w, \mathbf{x}^{(i)}, y^{(i)}) = \max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \tag{2}$$

We can rewrite it as:

$$H_1(w, \mathbf{x}^{(i)}, y^{(i)}) = \begin{cases} 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right), & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) > 0 \\ 0, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \leq 0 \end{cases}$$

Note that the derivative is not defined when $\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) = 0$. This is also explicitly stated in the task description. However, we have chosen to adopt the convention that the derivative is 0 in this case. In order to calculate the gradient, we have to obtain the derivative with respect to w and b.

$$\frac{dH_1(w, \mathbf{x}^{(i)}, y^{(i)})}{dw} = \begin{cases} -y^{(i)}\mathbf{x}^{(i)}, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) > 0 \\ 0, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \leq 0 \end{cases}$$

$$\frac{dH_1(w, \mathbf{x}^{(i)}, y^{(i)})}{db} = \begin{cases} -y^{(i)}, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) > 0 \\ 0, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \leq 0 \end{cases}$$

We can now focus on the:

[1] While this is not necessarily true in practice, there are a number of reasons why we do not have to worry about these bad singular points.

$$H_2(w, \mathbf{x}^{(i)}, y^{(i)}) = \min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{3}$$

We know that the following holds:

$$\|\mathbf{w}\|_2^2 = \sum_{i=1}^{N} (w_i)^2$$

Therefore, calculating the derivative of the sum with respect to w is rather simple. The derivative will be equal to $\frac{2}{2}\sum_{i=1}^{N}(w_i)$, and the corresponding gradient is:

$$\frac{d(\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2)}{dw} = w$$

.

$$\frac{d(\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|_2^2)}{db} = 0$$

.

By combining the calculated equations, we have:

$$\begin{aligned} \frac{dH}{dw} &= w + C\sum_{i=1}^{N} \frac{dH_1}{dw}, \\ \text{where} \quad \frac{dH_1}{dw} &= \begin{cases} -y^{(i)}x^{(i)}, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) > 0 \\ 0, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \leq 0 \end{cases} \end{aligned} \tag{4}$$

$$\begin{aligned} \frac{dH}{db} &= 0 + C\sum_{i=1}^{N} \frac{dH_1}{db}, \\ \text{where} \quad \frac{dH_1}{db} &= \begin{cases} -y^{(i)}, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) > 0 \\ 0, & \text{if } 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \leq 0 \end{cases} \end{aligned} \tag{5}$$

When calculating the gradients, we have used the assumption that $y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \neq 1$. If we drop it and state that it could indeed be equal to one, then for any i, it means that some data points are lying exactly on the decision boundary. This problem could lead to ambiguity, because there are potential multiple margins that could fit these conditions. Therefore, the classifier could have issues in finding a unique solution.

2.2 Task 2 : Problem

Try different values for C and for the learning rate η using dataset 1 (with the outlier removed). Use a grid where parameters are exponentially spaced apart, e.g., `'C' : 10.**np.arange(-3, 4)` (this serves just as an example – pick sensible parameter ranges yourself). Plot the decision boundary after you trained the classifier until convergence (monitor the loss).

In general: Looking at Equation 1 above, what tradeoff does C control? Explain what happens as $C \to 0$ (from the positive side) and as $C \to \infty$.

Task 2 : Solution

From the tested ranges for the C and learning rate, the best results are obtained with $C = 50.0$ and $\eta = 0.0001$. We have provided the corresponding plots for the decision boundary and loss values over the iterations. For most of the tested hyperparameters, we have obtained loss values that oscillate and do not maintain a stable curve. In the below plot, we can notice that for $C = 50.0$ and $\eta = 0.0001$, the loss steadily decreases with each iteration and converges to the minimum value. This is the exact effect we want to achieve. On the other hand, the decision boundary correctly separates all of the data items with largest minimum margin.

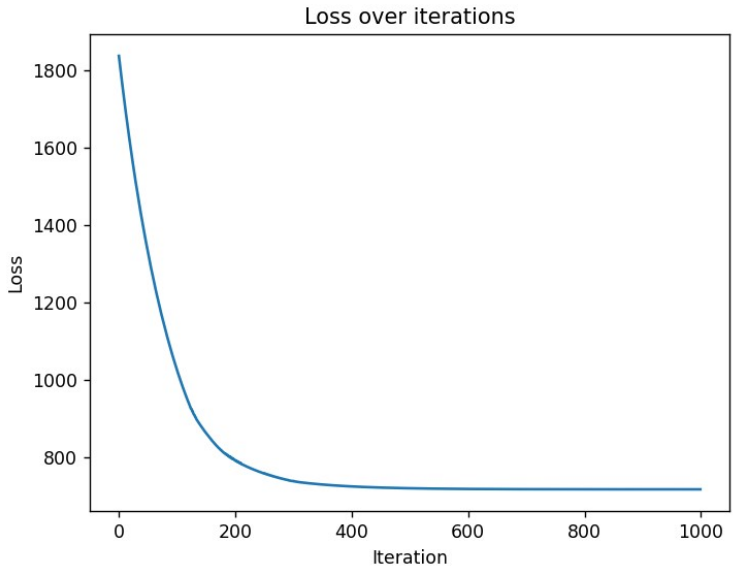

Figure 12: SVM Decision Boundary for Dataset 1 with $C = 50.0$ and $\eta = 0.0001$

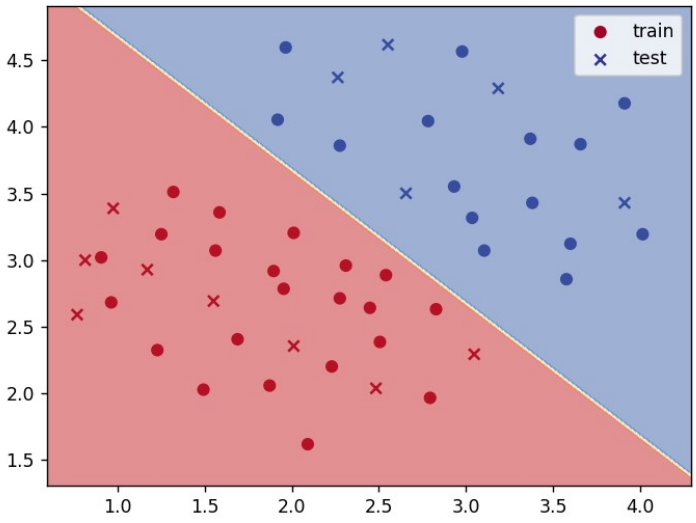

Figure 13: SVM loss over iterations for Dataset 1 with $C = 50.0$ and $\eta = 0.0001$

What tradeoff does C control?

The parameter C controls the tradeoff between having low training error and a low testing error. For example, large values of C would result in a smaller margin, which will try to classify most training data points correctly. As a results, the testing results and generalization will worsen. Therefore, we have a risk of overfitting. On the other hand, if the C is smaller, the optimizer will try and find a larger margin, even if that hyperplane misclassifies more points.

What happens as C approaches 0(from the positive side) and as C approaches positive infinity?

C primarily impacts the extent to which we weight the slack variables in the SVM. In other words, if we have a large C then the penalty is larger. Therefore, as C approaches infinity, the penalty will be infinite and our classifier will be a Hard Margin SVM Classifier, meaning it perfectly separates classes in the training data without allowing any misclassifications. However, if C is very small or zero, then we can tolerate a certain number of slack variables. The goal is to maximize the margin, with the cost of some misclassifications. In these cases, we have a risk of underfitting, while for $C \to \infty$ we have a problem of overfitting.

2.3 Task 3: Problem

For this task we will use scikit-learn's built-in support vector machine classifier. It supports slack variables, various kernels such as linear, polynomial or Gaussian kernels and more. We will try out linear and Gaussian kernels. For all three datasets, perform a grid search over C, `kernel`, and – if `kernel` is `'rbf'` – over γ. Pick sensible parameter ranges, state the best parameter configuration and mean cross-validation accuracy for each dataset. Include a plot of the decision boundaries for each dataset in your report. Finally, for each dataset, report the accuracy on the test set.

Task 3: Solution

For the parameters, we have tried out the following values: 'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf'], 'gamma': [0.1, 1, 10]. The best parameters, test accuracy and Mean Cross Validated Scores are:

1. Dataset 1: **Best parameters** - 'C': 0.1, 'gamma': 0.1, 'kernel': 'linear', **Mean CV accuracy** - 0.97142, **Accuracy on test set** - 1.0
2. Dataset 2: **Best parameters** - 'C': 10, 'gamma': 10, 'kernel': 'rbf', **Mean CV accuracy** - 0.92431, **Accuracy on test set**: 0.93981
3. Dataset 3: **Best parameters** - 'C': 10, 'gamma': 0.1, 'kernel': 'linear', **Mean CV accuracy** - 0.89213, **Accuracy on test set**: 0.90566

The corresponding graphs:

3 Decision Trees & Ensemble Methods [6 points, 5* bonus points]

Decision trees are defined by recursively partitioning the input space, and defining a local model in each resulting region of input space. This can be represented by a tree, with one leaf per region. Whilst being easy to interpret and relatively robust to outliers, they don't predict very accurately compared to other models. One way to reduce the variance of an estimate is to average over many estimates. For example, we can train many different trees on different subsets of the training data and input variables, chosen randomly with replacement. This technique is known as random forests.

Tasks:

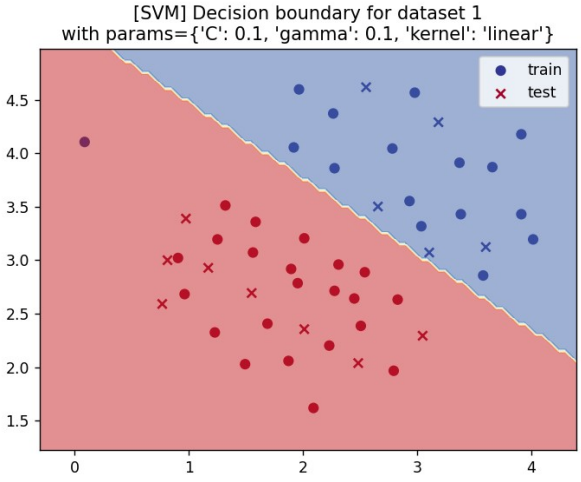

Figure 14: Dataset 1

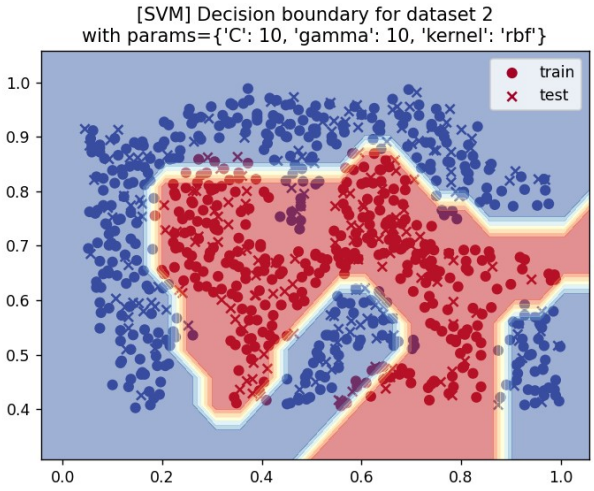

Figure 15: Dataset 2

3.1 Task 1: Problem

For each of the three datasets and for each `n_estimators` $\in \{1, 100\}$, we will search over `max_depth` $\in \{1, 2, \ldots, 25\}$ using `GridSearchCV` and fit a `RandomForestClassifier` with `random_state=0`. For each dataset and `n_estimators` $\in \{1, 100\}$, report the best value for `max_depth`, as well as the mean cross-validation accuracy (6 values in total). Also, include a plot of the decision boundary for each of the 6 configurations in your report.

For each dataset, the code skeleton creates a plot that shows the relationship between `max_depth` $\in \{1, 2, \ldots, 25\}$ (x-axis) and both the mean test CV score and the mean train CV score (y-axis).

For each dataset, instantiate a model using the best of the two configurations you have tried, train the model with the full training dataset, and report the test accuracy.

Discuss the effects of varying the number of trees and the maximum tree depths on the decision boundary and the performance. Will the random forest be more or less affected by outliers or noise in the data as you increase the number of trees in the forest?

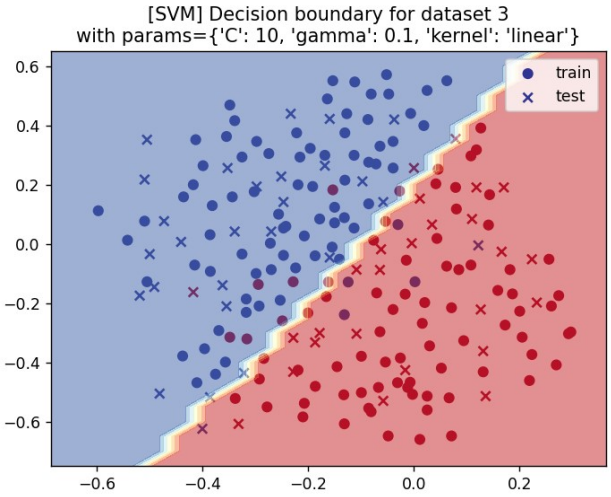

Figure 16: Dataset 3

3.2 Task 1:Solution

For each of the three datasets, we performed the following:

1. Used `GridSearchCV` to find the best `max_depth` for `n_estimators` $\in \{1, 100\}$.
2. Reported the best `max_depth` and mean cross-validation accuracy for training and validation sets (6 values in total).
3. Plotted the decision boundaries for the 6 configurations.
4. Showed the relationship between `max_depth` and the mean test CV score and mean train CV score.
5. Instantiated and evaluated the best model for each dataset.
6. Discussed the effects of varying the number of trees and maximum tree depths on decision boundary and performance.

3.2.1 Results

Here are the results for the best parameters, mean cross-validation accuracies, and test set accuracies for each dataset.

Dataset 1

- **n_estimators=1**:
  - Best max_depth: 3
  - Mean CV training accuracy: 0.9537
  - Mean CV validation accuracy: 0.8571
- **n_estimators=100**:
  - Best max_depth: 2
  - Mean CV training accuracy: 1.0000
  - Mean CV validation accuracy: 0.9980
- **Best Parameters**: n_estimators=100, max_depth=2
- **Test Set Accuracy**: 92.31%

Dataset 2

- **n_estimators=1**:
  - Best max_depth: 12
  - Mean CV training accuracy: 0.9189
  - Mean CV validation accuracy: 0.8777
- **n_estimators=100**:
  - Best max_depth: 11
  - Mean CV training accuracy: 0.9684
  - Mean CV validation accuracy: 0.9381
- **Best Parameters**: n_estimators=100, max_depth=11
- **Test Set Accuracy**: 97.69%

Dataset 3

- **n_estimators=1**:
  - Best max_depth: 4
  - Mean CV training accuracy: 0.9354
  - Mean CV validation accuracy: 0.8587
- **n_estimators=100**:
  - Best max_depth: 7
  - Mean CV training accuracy: 0.9882
  - Mean CV validation accuracy: 0.8992
- **Best Parameters**: n_estimators=100, max_depth=7
- **Test Set Accuracy**: 88.68%

3.2.2 Decision Boundaries

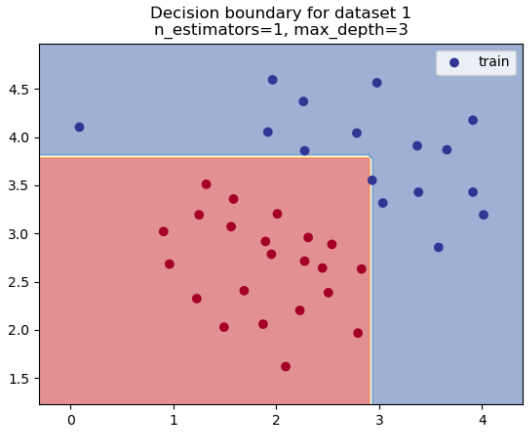

Figure 17: Decision Boundary for Dataset 1, n_estimators=1, max_depth=3. The decision boundary is rigid, indicating potential overfitting to the training data.

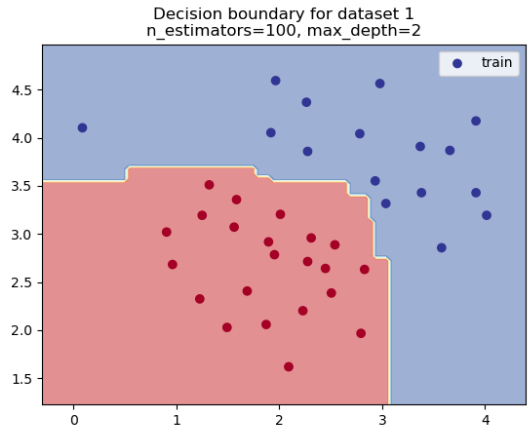

Figure 18: Decision Boundary for Dataset 1, n_estimators=100, max_depth=2. The decision boundary is smoother, indicating better generalization.

Dataset 1

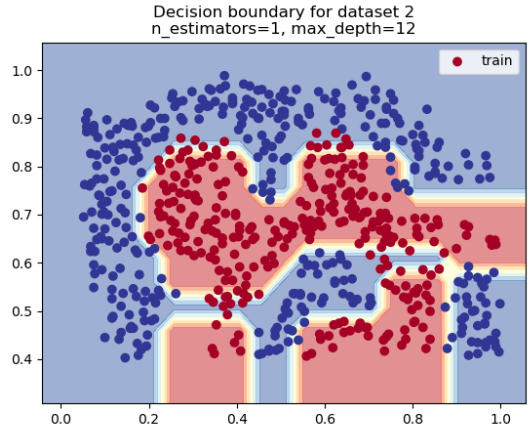

Figure 19: Decision Boundary for Dataset 2, n_estimators=1, max_depth=12. The model captures fine details, showing higher variance and overfitting.

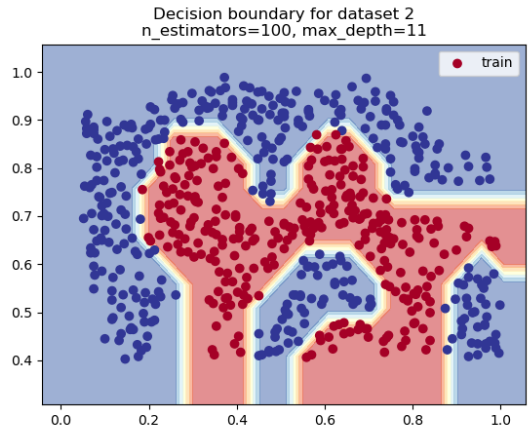

Figure 20: Decision Boundary for Dataset 2, n_estimators=100, max_depth=11. The model generalizes better with smoother boundaries.

Dataset 2

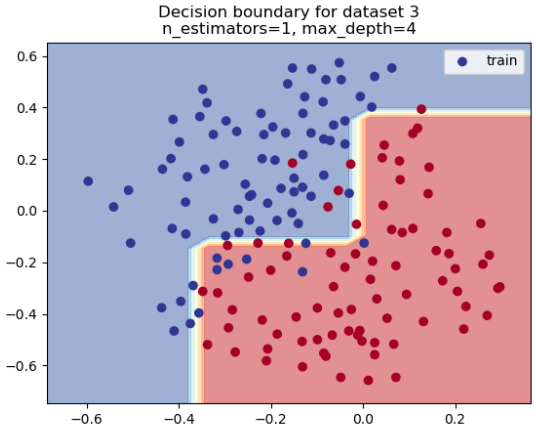

Figure 21: Decision Boundary for Dataset 3, n_estimators=1, max_depth=4. The rigid boundaries indicate higher variance and again potential overfitting.

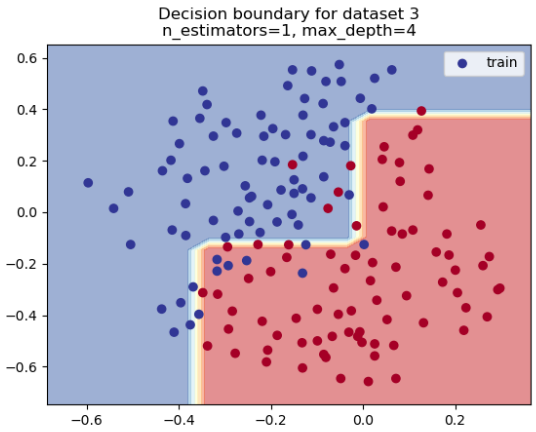

Figure 22: Decision Boundary for Dataset 3, n_estimators=100, max_depth=7. The smoother boundaries indicate better generalization.

Dataset 3

3.2.3 Training and Validation Scores

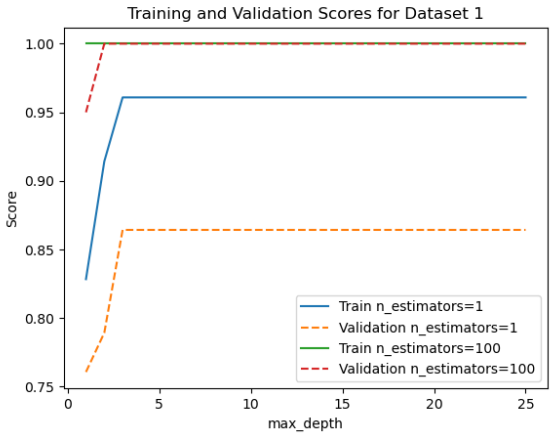

Figure 23: Training and Validation Scores for Dataset 1. The training accuracy increases with depth, while validation accuracy peaks at an optimal depth before decreasing due to overfitting. Models with 100 trees consistently show higher validation accuracy.

Dataset 1

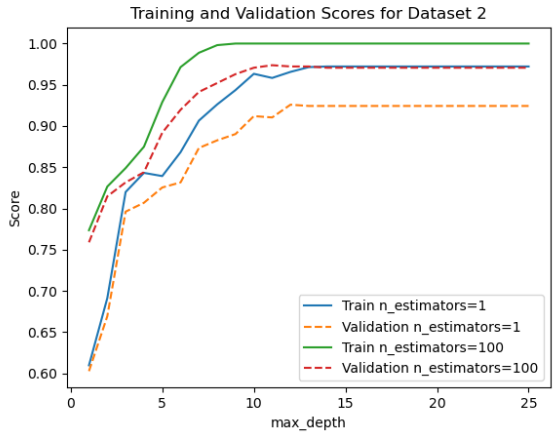

Figure 24: Training and Validation Scores for Dataset 2. Similar to Dataset 1, the training accuracy increases with depth. Models with 100 trees show higher validation accuracy, indicating better generalization.

Dataset 2

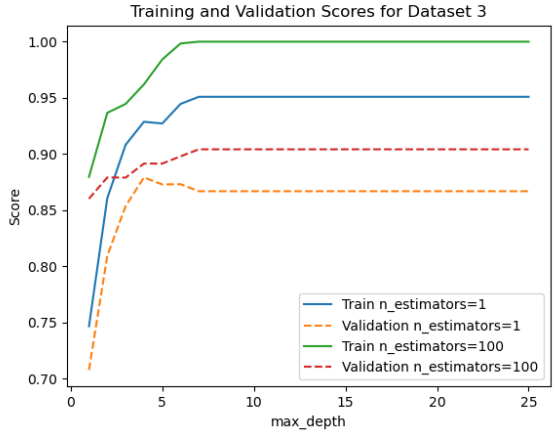

Figure 25: Training and Validation Scores for Dataset 3. The pattern is consistent with the other datasets, where models with 100 trees perform better on validation data compared to models with only 1 tree.

Dataset 3

3.2.4 Discussion

1. **Effects of Varying Number of Trees and Maximum Tree Depths**: Increasing the number of trees (`n_estimators`) generally improves the model's robustness and reduces variance, leading to better performance. This is evident in the smoother decision boundaries and higher validation scores for models with 100 trees compared to those with only 1 tree.

2. **Impact on Decision Boundaries**: With `n_estimators`=1, the decision boundaries are more rigid and less smooth, indicating higher variance and overfitting to the training data. For example, in Dataset 1 with `n_estimators`=1 and `max_depth`=3, the decision boundary is very sharp and closely follows the training points. With `n_estimators`=100, the decision boundaries are smoother and more generalized, indicating better fit for the validation data. This is observed in Dataset 1 with `n_estimators`=100 and `max_depth`=2, where the boundary is smoother and shows better generalization.

3. **Effect of Maximum Tree Depth**: A higher `max_depth` allows the trees to capture more details and potentially overfit the training data. For instance, in Dataset 2 with `n_estimators`=1 and `max_depth`=12, the model captures very fine details of the training data. However, with enough trees (`n_estimators`=100), this overfitting effect is mitigated, as seen in the improved validation scores and smoother boundaries for `n_estimators`=100 and `max_depth`=11 in Dataset 2.

4. **Sensitivity to Outliers and Noise**: Random forests are less affected by outliers and noise as the number of trees increases. This is because each tree is trained on a different subset of data, and averaging their predictions reduces the influence of outliers. The higher the number of trees, the more robust the model becomes, as it aggregates the predictions from multiple trees to minimize the impact of any single noisy or outlier data point.

5. **Training and Validation Scores**: The plots of training and validation scores for different `max_depth` values show that training accuracy increases with depth, while validation accuracy peaks at an optimal depth before decreasing due to overfitting. For all three datasets, models with 100 trees consistently show higher validation accuracy, indicating better generalization compared to models with only 1 tree.

3.2.5 Conclusion

Random forests significantly improve the performance of decision trees by averaging over multiple trees. By tuning the number of trees and maximum tree depths, we can achieve a good balance between bias and variance, leading to robust models with better generalization. The results show that using 100 estimators generally leads to better performance compared to using a single estimator, highlighting the effectiveness of the random forest approach. The sensitivity to outliers and noise decreases with the number of trees, making random forests a reliable choice for various datasets.

3.3 Task 2:Problem

In general: Assume you're given a training dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ where each $\mathbf{x}^{(i)}$ is a unique[2] feature vector and each target $y^{(i)}$ is one of k classes. Is it always possible to construct a single decision tree that achieves 100% train accuracy? Briefly explain why you think this is true/false.

3.4 Task 2: Solution

Yes, it is always possible to construct a single decision tree that achieves 100% training accuracy for the dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ where each $\mathbf{x}^{(i)}$ is unique and each target $y^{(i)}$ belongs to one of k classes. This is because decision trees can perfectly fit the training data by creating leaves that correspond to each unique training example.

Here is why this is the case:

1. **Uniqueness of Feature Vectors**: Since each $\mathbf{x}^{(i)}$ is unique, there are no duplicate feature vectors in the dataset. This uniqueness allows the decision tree to distinguish between each training example without any ambiguity.

2. **Tree Construction Process**: Decision trees are constructed by recursively splitting the dataset based on feature values. At each node, the tree selects the feature and threshold that best separate the data into more homogeneous subsets with respect to the target variable y.

3. **Leaf Node Formation**: If the decision tree is allowed to grow without any restrictions (e.g., no pruning or depth limit), it will continue to split the data until each leaf node contains only one training example. Given the uniqueness of each feature vector, the tree can always generate splits that isolate each training example into its own leaf node.

4. **Perfect Classification**: When each leaf node corresponds to a single training example, the decision tree can accurately classify each $\mathbf{x}^{(i)}$ with its associated class label $y^{(i)}$. This results in 100% training accuracy because there will be no misclassifications.

[2] i.e., $\forall\, \mathbf{x}^{(i)}, \mathbf{x}^{(j)} \in \mathcal{D} : \left(\mathbf{x}^{(i)} = \mathbf{x}^{(j)}\right) \implies (i = j)$.

In summary, the unique nature of the feature vectors in the dataset ensures that a decision tree can be constructed to perfectly fit the training data, achieving 100% training accuracy. The decision tree achieves this perfect fit by making enough splits to isolate each training example.
Bonus task [5* bonus point]:

3.5 Task Bonus: Problem

Tree-based methods have the useful property that they can be used to determine the importance of individual features for classification or regression. Many real-world datasets contain a large number of features, many of which are not useful for a particular task. Pre-selecting the right features is often an important step to achieve good performance. In this task, we will explore a dataset we know nothing about: It has 25 features and a total of 800 training examples and 200 test examples with 8 different classes.

Fit a `RandomForestClassifier` on the dataset and find the most important features. Conveniently, the RandomForestClassifier provides the `feature_importances_` attribute once we fit it to the dataset. Plot the relative feature importance and report the CV validation accuracy of a `SVC` classifier. Briefly explain the intuition on how scikit-learn can compute feature importances in a tree-based model (no formulas needed, just a concise high-level explanation).

The goal of *recursive feature elimination* (RFE) is to select features by recursively considering smaller and smaller sets of features. First, an estimator is trained on the initial set of features and the importance of each feature is obtained, typically through a specific attribute (e.g., `coef_` or `feature_importances_`). Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached. RFECV performs RFE in a cross-validation loop to find the optimal number of features automatically. We will use this approach to filter the dataset and train another SVC on the new dataset. Report the mean cross-validated accuracy and compare it to the CV accuracy that you achieved without feature elimination. If this classifier performs better, use it to report the test accuracy. Otherwise, use the previous SVC classifier. Report which features (indices) are still in the pruned feature set.

3.6 Task Bonus Solution

In this task, we explore a dataset with 25 features, 800 training examples, and 200 test examples, classified into 8 different classes. Our objectives are as follows:

- Fit a `RandomForestClassifier` to the dataset and identify the most important features.
- Plot the relative feature importances.
- Report the cross-validation (CV) accuracy of an `SVC` classifier.
- Use Recursive Feature Elimination (RFE) to select the best subset of features and compare the performance of the `SVC` classifier before and after feature elimination.
- Report the test accuracy of the better-performing classifier and the selected features after RFE.

3.6.1 Methodology

Random Forest Feature Importance First, we fit a `RandomForestClassifier` to the dataset. The classifier's `feature_importances_` attribute provides a measure of the importance of each feature. This importance is calculated based on the mean decrease in impurity (such as Gini impurity or entropy) across all trees in the forest. Features that result in larger decreases in impurity are considered more important.

Plotting Feature Importances We create a horizontal bar plot to visualize the relative importance of each feature, allowing us to see which features contribute most to the model's predictions.

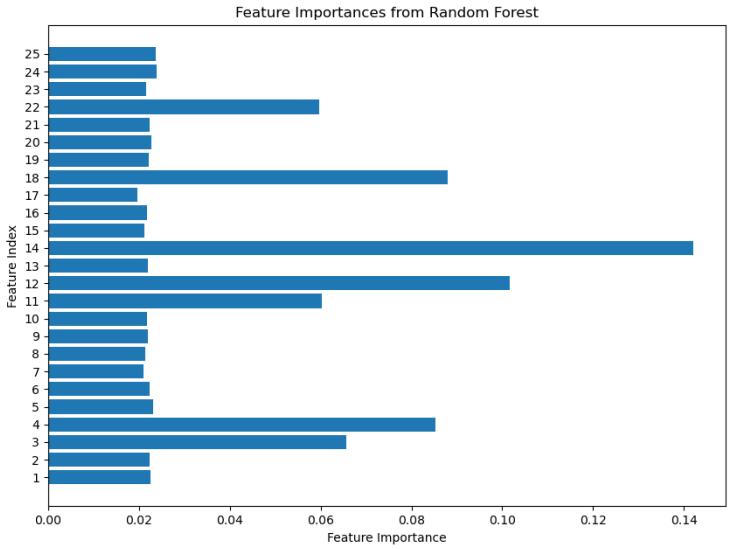

Figure 26: Feature Importances from Random Forest

As shown in Figure 26, features 14, 22, 11, and 4 exhibit the highest importance scores. This indicates that these features contribute the most to the predictions of the RandomForestClassifier, making them critical for classification in this dataset.

Cross-Validation Accuracy of SVC We perform a grid search using `GridSearchCV` to find the best parameters for the `SVC` classifier. The mean cross-validated accuracy is then reported. This step ensures that we have a strong baseline performance for comparison.

Recursive Feature Elimination (RFE) Recursive Feature Elimination (RFE) is used to recursively eliminate less important features. The process starts with an estimator trained on the initial set of features, and the importance of each feature is determined through attributes such as `coef_` or `feature_importances_`. The least important features are pruned, and the procedure is repeated until the desired number of features is reached. `RFECV` performs RFE in a cross-validation loop, ensuring robust and generalizable feature selection.

Comparison and Reporting We compare the cross-validation accuracy of the `SVC` classifier before and after feature elimination. The better-performing classifier is used to report the test accuracy. Additionally, the indices of the selected features after RFE are reported.

3.6.2 Results

Random Forest Feature Importances The feature importances determined by the `RandomForestClassifier` are shown in Figure 26. This plot indicates which features are most important for classification.

SVC Performance The mean cross-validation accuracy of the chosen `SVC` classifier before feature elimination is reported as follows:

Mean CV accuracy of the chosen SVC: 0.6347

SVC Performance after RFE After performing RFE, the mean cross-validated accuracy of the `SVC` classifier is reported:

Mean CV accuracy of the SVC after RFE: 0.7493

Test Accuracy and Selected Features The test accuracy of the `SVC` classifier after RFE, which showed better performance, is reported. Additionally, the indices of the selected features are listed.

Test accuracy of the SVC after RFE: 0.7560
Selected features: [3, 11, 13, 17]

3.6.3 Conclusion

The feature importances identified by the `RandomForestClassifier` provided valuable insights into which features contribute most to the classification task. Using RFE to select the most important features significantly improved the performance of the `SVC` classifier. The mean cross-validation accuracy increased from 0.6347 to 0.7493, and the test accuracy improved to 0.7560. The selected features were indices [3, 11, 13, 17], which were used to train the final `SVC` classifier.

This task demonstrates the importance of feature selection in improving model performance and highlights the effectiveness of tree-based methods and recursive feature elimination for this purpose.