Week 4 - Feature selection, Model selection, Model training and Evaluation Machine Learning

David Esteso Calatrava

October 19, 2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Scinece and Statics

Introduction

In this assignment, we will analyze two datasets to evaluate their usefulness for machine learning tasks. Not all datasets effectively capture important relationships, and some may contain too much noise, making it difficult to extract meaningful insights. Through this analysis, we will explore whether the datasets at hand are suitable for building reliable models, or if their limitations prevent them from delivering accurate results.

Dataset 1 - id:6–12-6-0

(i) Logistic Regression with Polynomial Features

We will train a logistic regression classifier with polynomial features and L2 penalty. The maximum polynomial order and regularization weight C will be chosen using cross-validation.

```
for max_poly_order in max_poly_order_range:
    poly = PolynomialFeatures(degree=max_poly_order)
    X_poly = poly.fit_transform(X)

    for C in C_range:
        model = LogisticRegression(C=C, penalty='l2', max_iter=100000)
        scores = cross_val_score(model, X_poly, y, cv=5, scoring='accuracy')
        mean_score = scores.mean()
        std_score = scores.std()

        results.append((max_poly_order, C, mean_score, std_score))

        if mean_score > best_score:
            best_score = mean_score
            best_model = model
            best_poly = poly
```

This code snippet evaluates logistic regression models with polynomial feature transformations of varying orders and regularization strengths C. It iterates through a specified range of polynomial degrees, generating polynomial combinations of the input features X. For each combination of polynomial degree and C, a logistic regression model is trained and assessed using 5-fold cross-validation. The mean accuracy and standard deviation of the scores are calculated, and the best-performing model is updated accordingly. The 'max_iter=100000' parameter ensures sufficient iterations for convergence, particularly for high polynomial orders.

We selected $C = \{0.01, 0.1, 1, 10, 100, 1000\}$ as these are typical values commonly used to tune regularization in logistic regression models, with the optimal value often found within this range. For polynomial features, degrees from 1 to 9 were chosen to represent different levels of nonlinearity, which balances model complexity and reduces the risk of overfitting. Although these ranges may seem broad and could affect execution time, they allow for a better understanding of the model's behavior and the trend in accuracy.

We have also split the dataset into training and testing sets and made predictions on the test set, as shown in Figure 2.

The dataset is divided using the 'train_test_split' function, with 80% allocated for training and 20% for testing, ensuring reproducibility with 'random_state=42'. If a polynomial transformation object ('poly') is provided, the training features ('X_train') are transformed using 'fit_transform', and the same transformation is applied to the test features ('X_test').

The logistic regression model is then fitted to the transformed training data ('X_train_poly'). Predictions are subsequently made on the transformed test set ('X_test_poly').

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Transform the features if polynomial features are provided
X_train_poly = poly.fit_transform(X_train) if poly is not None else X_train
X_test_poly = poly.fit_transform(X_test) if poly is not None else X_test

model.fit(X_train_poly, y_train)

predictions = model.predict(X_test_poly)
```

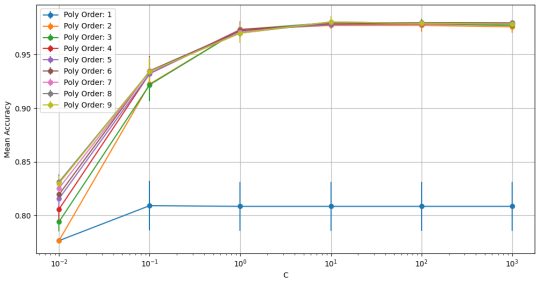

Figure 1: Cross validation LR - Dataset 1

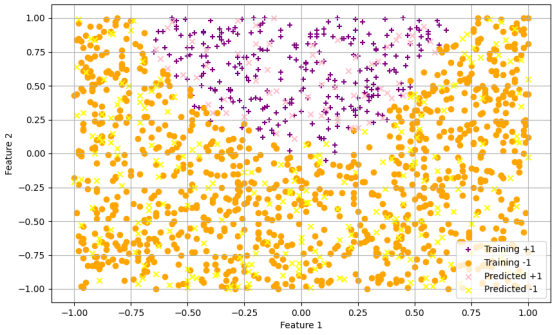

Figure 2: Predictions LR - Dataset 1

C	**Polynomial Degree**	**Intercept**	**Most Influential Features**
10	3	-0.37	$X2 : 11.18, X1^2 : -14.81$

Table 1: Best LR model - Dataset 1

As shown in Table 1, the most influencial feaures align with our expectations based on the data observed in Figure 2. The data points for the positive class $(+1)$ are concentrated in the upper part of the graph, indicating a significant positive influence of the second feature (X_2) on predictions, as reflected in its positive and high coefficient. Additionally, the influence of X_1^2 is notable because the term $-X_1^2$ resembles the decision boundary we would draw, suggesting a strong relationship between this feature and the classification task. Conversely, data along the the first feature axis shows a more balanced spread, leading to a smaller influence on predictions and a lower coefficient. Furthermore, since the majority of the points represent class -1, we could have anticipated a negative intercept.

The values of $C = 10$ and the polynomial feature degree of 7 fall within the manually established ranges we set, indicating that we have made appropriate choices for our parameter search. This selection aligns with the expected trend where increasing the magnitude of both initially improves model performance, eventually reaching an optimal point before performance declines with further increases in both. Our results suggest that we have successfully identified this peak performance point, confirming that our parameter selection was effective in optimizing the model's predictive capability.

We chose *accuracy* as the scoring metric because it provides a straightforward evaluation of the overall correctness of the model's predictions. Other metrics such as *precision*, *recall*, F1-*score*, or *ROC* and *AUC* focus on specific aspects, like minimizing false positives or false negatives. Since we do not have any specific objective or prior information about class imbalance, accuracy serves as a balanced and appropriate choice for this classification task. However,

we will analyze these other metrics later to gain a deeper understanding of the model's performance. Additionally, the mean squared error would not be suitable for a dataset with only two classes, as it is designed for continuous values, not discrete ones.

(ii) k-Nearest Neighbors (kNN)

We are going to train a kNN classifier and select the best value of k via cross-validation. This Python code performs 5-fold cross-validation to evaluate the accuracy of k-Nearest Neighbors (kNN) models for different values of k. It iterates through a specified range of k values in 'k_values', creating a 'KNeighborsClassifier' for each k. The model's accuracy is assessed using 'cross_val_score', which calculates the scores across the folds. The mean accuracy and standard deviation of the scores are computed, and the best-performing k is updated based on the highest mean score.

```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
    mean_score = np.mean(scores)
    std_score = np.std(scores)
    results.append((k, mean_score, std_score))

for k, score, _ in results:
    if score > best_score:
        best_k = k
        best_score = score
```

The k-Nearest Neighbors algorithm is a straightforward and powerful method we will be using to carry out classification tasks, although it can also be used for regression. In classifying a new feature vector x, the algorithm calculates the distances from x to every point in the training data using some distance metric, often Euclidean distance. After calculating these distances, the k-nearest neighbors algorithm then picks the k closest training examples. In classification, it elects the label through a majority vote of the k nearest neighbors, while for regression problems, it simply calculates the average of the labels. This kind of instance-based learning enables the model to predict directly from the training dataset. Since kNN can naturally capture nonlinear decision boundaries through its instance-based learning and flexible neighborhood structure, we did not use polynomial features, as they are unnecessary for this method, as we will see.

The choice of the parameter k will be crucial in determining the effectiveness of the algorithm. A small value of k may make the model very sensitive to noise in the data and cause overfitting, since it would pick the label of the closest neighbor. On the other hand, when the parameter k is set to an extremely high value approaching infinity, the model will end up predicting the majority class for the entire dataset and may fail to consider the individual characteristics of a new instance. This could be considered an underfitting situation in which the model is too simple and fails to model the underlying complexities of the data. Consequently, a proper choice of k is critical.

The choice of the range for k values is based on the earlier discussion. It is a range where we can typically find the best value for k. Additionally, this range is typically broad enough to show how performance declines after reaching the optimal value for k, as shown in Figure 3. This reduction in performance is related to overfitting, which cross-validation helps us detect.

The results in Table 2 show that the model achieved an optimal $k = 31$ with an impressive accuracy of 0.98, indicating strong performance in classification. As shown in Figure 4, k exhibits the same trend as the hyperparameter C and polynomial degree from the previous section: after reaching the optimum, increasing the number of neighbors leads to a decline in

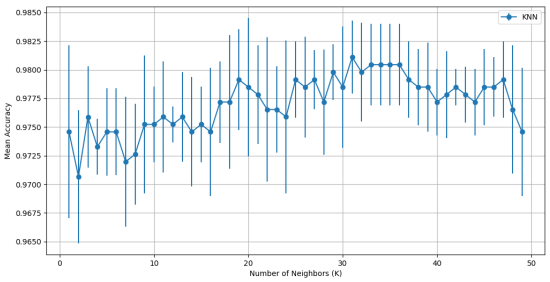

Figure 3: KNN Cross validation - Dataset 1

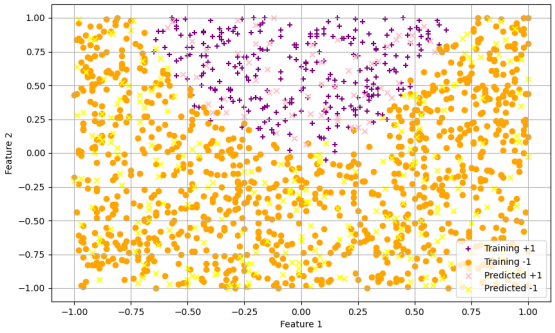

Figure 4: KNN predictions - Dataset 1

Best k	Accuracy
31	0.98

Table 2: Best KNN model - Dataset 1

performance. Moreover, Figure 4 demonstrates that the predictions closely resemble those from the previous section (in Figure 2), successfully classifying almost all instances. Moreover, it is noteworthy that the standard deviation of the accuracy is on the order of hundredths, as we can see in Figure 1 and Figure 3, meaning that the performance of the model remains quite consistent across the folds with only minor variations. However, in upcoming sections, we will examine additional metrics like precision, recall, and F1 score to gain a more comprehensive understanding of both model's effectiveness.

(iii) Confusion Matrices

The confusion matrices for the logistic regression and kNN classifiers will be calculated, along with a baseline classifiers that always predicts the most frequent class. The confusion matrix is important because it provides a clear visual representation of a model's performance, allowing us to identify specific types of errors and understand how well the model distinguishes between different classes.

The following Python code creates a confusion matrix and calculates various evaluation metrics using the respective functions imported from 'sklearn.metrics'.

A confusion matrix is generated by comparing the true labels y_{test} with the predicted labels. The accuracy, precision, recall, and F1 score are then computed using the corresponding imported functions. Finally, the confusion matrix is displayed.

```
# Create a confusion matrix
cm = confusion_matrix(y_test, predictions, labels=[-1, 1])

# Calculate metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='binary', pos_label=1,
    zero_division=0)
recall = recall_score(y_test, predictions, average='binary', pos_label=1, zero_division=0)
f1 = f1_score(y_test, predictions, average='binary', pos_label=1, zero_division=0)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[-1, 1])
plt.figure(figsize=(8, 6))
```

```
disp.plot(cmap='Blues', values_format='d')
```

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	239	0
Actual Positive	68	0

Metric	Value
Accuracy	0.7785
Precision	0.0000
Recall	0.0000
F1 Score	0.0000

Table 3: Confusion Matrix and Metrics baseline - Dataset 1

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	237	2
Actual Positive	5	63

Metric	Value
Accuracy	0.9772
Precision	0.9692
Recall	0.9265
F1 Score	0.9474

Table 4: Confusion Matrix and Metrics LR - Dataset 1

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	237	2
Actual Positive	8	60

Metric	Value
Accuracy	0.9674
Precision	0.9677
Recall	0.8824
F1 Score	0.9231

Table 5: Confusion Matrix and Metrics KNN - Dataset 1

Regarding the metrics next to each confusion matrix, *Accuracy* is the ratio of correctly predicted instances among all the instances predicted. It is useful in cases where the classes are somewhat balanced, but could misleading if one class dominates significantly. The formula is $\frac{TP+TN}{TP+TN+FP+FN}$. Here, TP stands for true positives, TN stands for true negatives, FP stands for false positives, and FN denotes false negatives.

Precision is important when we want to minimize false positives, such as in spam detection. It indicates the proportion of positive predictions that were actually correct: $\frac{TP}{TP+FP}$.

Recall is the measure which we want to be high when we want to avoid false negatives, for example, when we are interested in finding a disease: $\frac{TP}{TP+FN}$.

The *F1 Score* represents the harmonic average between Precision and Recall, therefore being a useful measure when one is dealing with class imbalance or if both are relevant: $2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.

These metrics will be important because our dataset is slightly imbalanced, with the negative class significantly outnumbering the positive class. This is reflected in the accuracy of the dummy regressor, which approaches 80. It highlights the need to pay attention to other metrics that provide a more complete view of the model's performance, especially in the context of imbalanced classes.

(iv) ROC Curves

We will plot the ROC curves for the logistic regression and kNN classifiers, along with the baseline classifier ROC. In this assignment, we will use a baseline classifier that always predicts the most common class.

The following snippet generates ROC curves for multiple models and calculates the AUC for each using the respective imported functions.

A figure is created with a specified size, and for each model, the false positive rate (FPR) and true positive rate (TPR) are computed using 'roc_curve'. The AUC is then calculated with 'auc', and each ROC curve is plotted with its corresponding label, including the AUC value for easy comparison.

```
plt.figure(figsize=(10, 6))

# predictions and y_tests are lists of model predictions and true labels
for i in range(len(predictions)):
    fpr, tpr, _ = roc_curve(y_tests[i], predictions[i])
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve for each model
    plt.plot(fpr, tpr, label=f'{names[i]} (AUC = {roc_auc:.2f})')
```

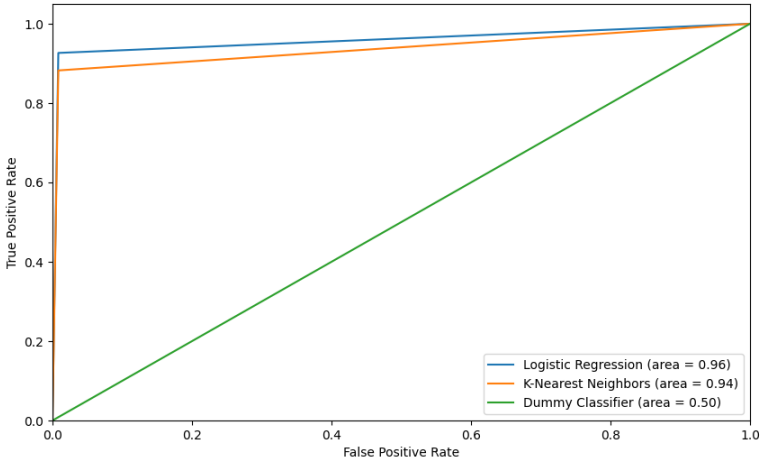

Figure 5: ROC curves - Dataset 1

The ROC (Receiver Operating Characteristic) is a curve that plots the true positive rate against the false positive rate at different threshold settings for a binary classifier. Each point of the ROC represents a different threshold. It starts with a high threshold where no positives are returned. As we decrease the threshold, the points start to rise upwards and to the right, indicating that more true positives appear but also some false positives begin to appear. The best curve clings to the upper left corner. It means high true positive rates at a few false positive rates. The ROC curve is particularly useful when dealing with imbalanced datasets, because it offers a complete view of the model's performance, not just of its accuracy.

For interpretation of the ROC curve, one should seek the AUC; the closer to 1 it is, the better a model is performing. In contrast, a value of approximately 0.5 would be obtained if performance is no better than random guessing or, in our case, predicting the most common class.

(v) Model Comparison

We will compare the performance of the logistic regression, kNN, and a baseline classifier, which predicts the most probable class, based on confusion matrices, metrics and ROC curves.

We see that logistic regression is slightly superior in all the metrics compared to the k-nearest neighbors model. To understand the reason for this difference, we need to examine the confusion matrix. In it, we notice that LR has correctly classified three samples from the test subset in the positive class that KNN has incorrectly classified as negative. This highlights LR's ability to better identify true positives in this specific case.

This superiority is also reflected in the ROC curve, where we observe that the AUC for LR is higher. This is evident from LR's ability to achieve a greater true positive rate more quickly than KNN, as indicated by its curve being closer to the top-left corner of the graph, which we said earlier is desirable.

What could account for this slight superiority? One possible explanation lies in the nature of the dataset, which appears to be separable with a well-defined decision boundary. In such cases, logistic regression is particularly effective, as it can accurately model the relationship between the features and the outcome variable.

Additionally, both the logistic regression model and the k-nearest neighbors model have outperformed the baseline classifier in all metrics, indicating that they have effectively captured the underlying patterns in the dataset and are performing well for this specific task. We observe that all metrics approach 1, which is desirable, further emphasizing their effectiveness in this context.

It is important to highlight that the metrics of the dummy regressor are 0 because, in all calculations, the number of true positives (TP) is in the numerator, and the most probable class is negative. As a result, it has not predicted any true positives. However, we see that this also translates into a precision close to 80%, which can be misleading if we do not closely examine the other metrics.

On the whole, logistic regression represents the best model in this given dataset, despite their small difference in performances. Whereas from the metrics, it's crystal clear that it outperformed kNN though the margin is so small. That probably might be explained by the fact that it correctly approximated the underlying decision boundary hence fit for the problem in question. Therefore, I conclude that logistic regression is the preferred choice in this case, especially when dealing with datasets characterized by well-defined separability.

Dataset 2 - id:6-6-6-0

(i) Logistic Regression with Polynomial Features

We will repeat the logistic regression training process with polynomial features and L2 penalty for the second dataset, using cross-validation for model selection.

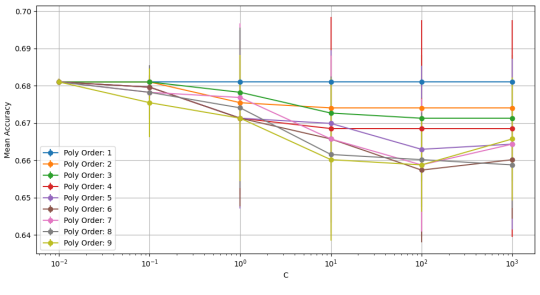

Figure 6: LR Cross validation - Dataset 2

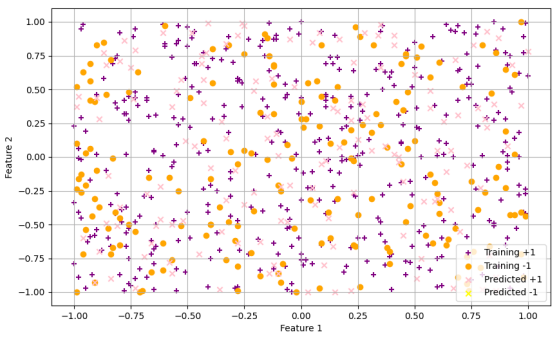

Figure 7: LR predictions - Dataset 2

C	Polynomial Degree	Intercept	Most Influential Features	Accuracy
0.01	1	0.78	$X1 : 0.02, X2 : 0.09$	0.68

Table 6: Best LR model - Dataset 2

The results in table 6 show that the model always predicts the positive class, which may be due to more than one reason: First, the large intercept value of 0.7831 suggests a strong bias toward predicting the positive class given negligible feature contributions. Second, the feature coefficients are very small and hence limit the features from making large contributions to the predictions. As Figure 7 shows, the data set is highly scattered, and it also does not have any clear linear separability, which worsen the prevailing scenario. It therefore follows that logistic regression would likely find it hard to separate the classes properly and hence would be biased toward predicting the positive class for the majority of the cases. This behavior is like the baseline model that predicts the most likely class, considering that the logistic regression model could be incapable of establishing any significant decision boundary within this dataset. In fact, if we were to draw it, we would see that the equation of that line does not intersect the dataset but rather passes underneath it, which explains why it always predicts the same class. Moreover, the low value of C and the polynomial degree of 1 indicate that fitting to the dataset is not beneficial, reinforcing the idea that the model cannot effectively capture the underlying patterns due to the lack of separability, as illustrated by Figure 6.

(ii) k-Nearest Neighbors (kNN)

For the second dataset, we will train a kNN classifier and select the optimal k through cross-validation.

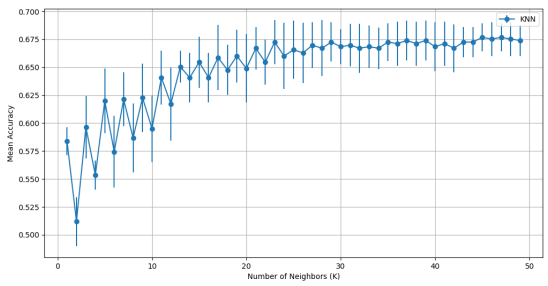

Figure 8: KNN Cross validation - Dataset 2

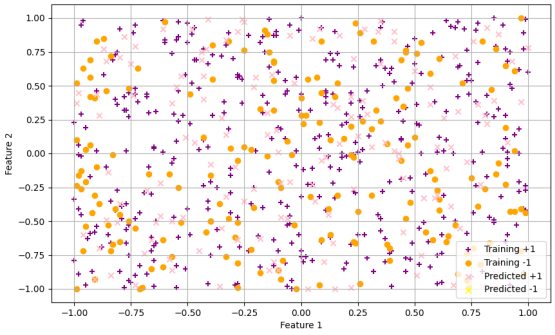

Figure 9: KNN predictions - Dataset 2

Best k	Accuracy
45	0.68

Table 7: Best KNN model - Dataset 2

The kNN model faces similar challenges, as the number of neighbors considered is significantly higher than what we previously explored. As discussed in our kNN analysis, increasing the number of neighbors can lead to less emphasis on the training data, resulting, in this case, in the model identifying optimal performance without necessarily focusing on the nearest samples. This behavior resembles that of logistic regression, where the model consistently predicts

the positive class, which is the majority class in this dataset, and does not take the dataset's variability into account. It is understandable that, with more neighbors taken into account, the likelihood of those neighbors belonging to the majority class increases. In Figure 8, we can observe this trend of improved performance as more neighbors are considered. Once again, we see that neither model effectively captures the underlying essence of the dataset, underscoring the difficulties presented by its characteristics.

(iii) Confusion Matrices

The confusion matrices for the logistic regression and kNN classifiers will be calculated, along with baseline classifier for comparison.

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	0	49
Actual Positive	0	96

Metric	**Value**
Accuracy	0.6621
Precision	0.6621
Recall	1.0000
F1 Score	0.7967

Table 8: Confusion Matrix and Metrics Baseline - Dataset 2

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	0	49
Actual Positive	0	96

Metric	**Value**
Accuracy	0.6621
Precision	0.6621
Recall	1.0000
F1 Score	0.7967

Table 9: Confusion Matrix and Metrics KNN - Dataset 2

Confusion Matrix		
	Predicted Negative	Predicted Positive
Actual Negative	0	49
Actual Positive	0	96

Metric	**Value**
Accuracy	0.6621
Precision	0.6621
Recall	1.0000
F1 Score	0.7967

Table 10: Confusion Matrix and Metrics LR - Dataset 2

(iv) ROC Curves

Then, we plot the ROC curves for the logistic regression and kNN classifiers, along with points corresponding to the baseline classifier.

(v) Model Comparison

Finally, we will compare the classifiers' performances based on the second dataset using confusion matrices and ROC curves. What has been discussed in the previous analyses is clearly reflected in the confusion matrices and the metrics in Table 8, 9 and 10, where we observe that all three models yield exactly the same results. This consistency among the models indicates a fundamental limitation in their ability to differentiate between the classes in the dataset. However, it is essential to note that this is not a fault of the models themselves, but rather

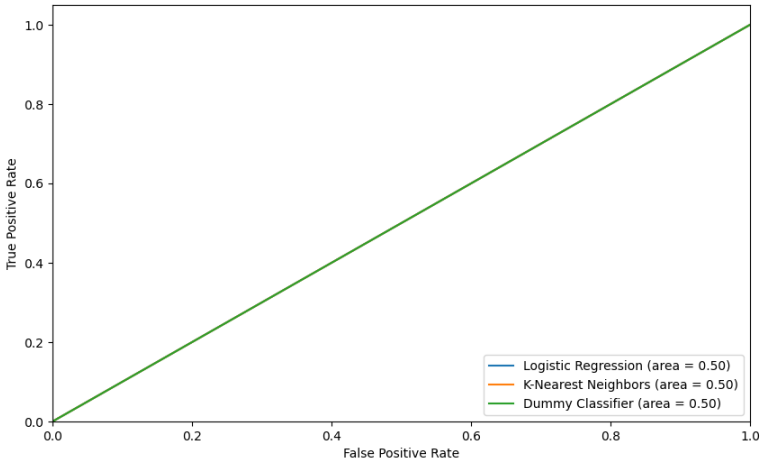

Figure 10: ROC curves - Dataset 2

a consequence of the dataset's inherent dispersion. After all, the ROC curve in Figure 10 presents an overlap for all three models, hence reinforcing the above assertion that none of them succeeded in catching the fundamental structure of the data.

To conclude, these properties of the dataset render the appliance of machine learning models hardly possible, since we cannot separate the classes from each other. That could also mean that the dataset is not very good for machine learning applications unless further preprocessing or feature engineering is performed in order to improve its separability and provide more informative patterns for the models.

Appednix

exercise.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay,
    precision_score, recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.dummy import DummyClassifier


def load_data(filepath):
    """Load the dataset from a CSV file."""
    df = pd.read_csv(filepath)
    return df

def cross_val_nested_loops(df, max_poly_order_range, C_range, dataset):
    """Train and evaluate a Logistic Regression model using nested loops. Return the best
        model, best polynomial features, and results."""
    X = df.iloc[:, :2].values
    y = df.iloc[:, 2].values

    best_score = 0
    results = []
    best_model = None
    best_poly = None

    for max_poly_order in max_poly_order_range:
        poly = PolynomialFeatures(degree=max_poly_order)
        X_poly = poly.fit_transform(X)

        for C in C_range:
            model = LogisticRegression(C=C, penalty='l2', max_iter=100000)
            scores = cross_val_score(model, X_poly, y, cv=5, scoring='accuracy')
            mean_score = scores.mean()
            std_score = scores.std()

            results.append((max_poly_order, C, mean_score, std_score))

            if mean_score > best_score:
                best_score = mean_score
                best_model = model
                best_poly = poly

    # Save the best results to a file
    results_filename = f"../output/{dataset}/best_results_lr.txt"
    with open(results_filename, 'w') as file:
        file.write(f"Best Polynomial Order: {best_poly.degree}\n")
        file.write(f"Best  C: {best_model.C}\n")
        file.write(f"Best Cross-Validation Score: {best_score:.4f}\n")

    return best_model, best_poly, results
```

```
def plot_cross_val_results(results, name, dataset):
    """Plot cross-validation results and save to a file."""
    if name == 'lr':
        results_df = pd.DataFrame(results, columns=['Max Poly Order', 'C', 'Mean Score', '
            Std Score'])
    elif name == 'knn':
        results_df = pd.DataFrame(results, columns=['K', 'Mean Score', 'Std Score'])

    plt.figure(figsize=(12, 6))

    # Handle plotting differently based on the model name
    if name == 'lr':
        for max_order in results_df['Max Poly Order'].unique():
            subset = results_df[results_df['Max Poly Order'] == max_order]
            plt.errorbar(subset['C'], subset['Mean Score'], yerr=subset['Std Score'],
                         label=f'Poly Order: {max_order}', marker='o')
            plt.xscale('log')

        plt.xlabel('C')

    elif name == 'knn':
        plt.errorbar(results_df['K'], results_df['Mean Score'], yerr=results_df['Std Score'
            ],
                     label='KNN', marker='o')
        plt.xlabel('Number of Neighbors (K)')

    plt.ylabel('Mean Accuracy')
    plt.legend()
    plt.grid()

    filename = f"../output/{dataset}/cv_{name}.png"
    plt.savefig(filename)
    plt.close()


def plot_and_save_results(df, model, poly, name, dataset):
    """Visualize training data and predictions of the model, and save the confusion matrix
        and metrics. Retunr the true labels and predictions."""
    features = df.shape[1] - 1
    X = df.iloc[:, :features].values
    y = df.iloc[:, features].values

    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
        =42)

    # Transform the features if polynomial features are provided
    X_train_poly = poly.fit_transform(X_train) if poly is not None else X_train
    X_test_poly = poly.fit_transform(X_test) if poly is not None else X_test

    model.fit(X_train_poly, y_train)

    predictions = model.predict(X_test_poly)

    plt.figure(figsize=(10, 6))

    # Plot training data
    plt.scatter(X_train[:, 0][y_train == 1], X_train[:, 1][y_train == 1], marker='+', color=
        'purple', label='Training +1')
```

```
plt.scatter(X_train[:, 0][y_train == -1], X_train[:, 1][y_train == -1], marker='o',
    color='orange', label='Training -1')

# Plot predictions on the test set
plt.scatter(X_test[:, 0][predictions == 1], X_test[:, 1][predictions == 1], marker='x',
    color='pink', label='Predicted +1')
plt.scatter(X_test[:, 0][predictions == -1], X_test[:, 1][predictions == -1], marker='x'
    , color='yellow', label='Predicted -1')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.legend(loc='lower right')
plt.grid(True)

filename = f"../output/{dataset}/predictions_{name}.png"
plt.savefig(filename)
plt.close()

# Create a confusion matrix
cm = confusion_matrix(y_test, predictions, labels=[-1, 1])

# Calculate metrics
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions, average='binary', pos_label=1,
    zero_division=0)
recall = recall_score(y_test, predictions, average='binary', pos_label=1, zero_division
    =0)
f1 = f1_score(y_test, predictions, average='binary', pos_label=1, zero_division=0)


disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[-1, 1])
plt.figure(figsize=(8, 6))

disp.plot(cmap='Blues', values_format='d')

# Save the confusion matrix plot to a file
cm_filename = f"../output/{dataset}/{name}_confusion_matrix.png"
plt.savefig(cm_filename)
plt.close()

# Save the confusion matrix and metrics to a text file
metrics_filename = f"../output/{dataset}/{name}_metrics.txt"
with open(metrics_filename, 'w') as f:
    f.write("Confusion Matrix:\n")
    np.savetxt(f, cm, fmt='%d')

    f.write("\nMetrics:\n")
    f.write(f"Accuracy: {accuracy:.4f}\n")
    f.write(f"Precision: {precision:.4f}\n")
    f.write(f"Recall: {recall:.4f}\n")
    f.write(f"F1 Score: {f1:.4f}\n")


# Save LR information to a file
if name == 'lr':
    filename = f"../output/{dataset}/logistic_regression_model_info.txt"
    with open(filename, 'w') as file:
        file.write("intercept: " + str(model.intercept_[0]) + "\n")
        for feature, coef in zip(poly.get_feature_names_out(['X1', 'X2']), model.coef_):
```

```
                file.write(f"{feature}: {coef}\n")
    return y_test, predictions


def train_knn_with_cross_validation(df , k_values, dataset):
    """Train a kNN classifier on the data and use cross-validation to select the best k and
        return the model and results."""

    features = df.shape[1] - 1
    X = df.iloc[:, :features].values
    y = df.iloc[:, features].values
    results = []
    best_k = 0
    best_score = 0

    for k in k_values:
        knn = KNeighborsClassifier(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=5, scoring='accuracy')
        mean_score = np.mean(scores)
        std_score = np.std(scores)
        results.append((k, mean_score, std_score))

    for k, score, _ in results:
        if score > best_score:
            best_k = k
            best_score = score


    # Train the final model with the best k on the full dataset
    final_knn = KNeighborsClassifier(n_neighbors=best_k)

    # Save the best results to a file
    results_filename = f"../output/{dataset}/best_results_knn.txt"
    with open(results_filename, 'w') as file:
        file.write(f"Best Number of Neighbors (k): {best_k}\n")
        file.write(f"Best Cross-Validation Score: {best_score:.4f}\n")

    return final_knn, results


def train_dummy_classifier(df):
    """Train a DummyClassifier on the provided DataFrame and return the model"""
    X = df.iloc[:, :2].values
    y = df.iloc[:, 2].values


    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
        =42)

    # Create and train the DummyClassifier with the 'most_frequent' strategy
    dummy_classifier = DummyClassifier(strategy='most_frequent')
    dummy_classifier.fit(X_train, y_train)


    return dummy_classifier


def plot_roc_curve(y_tests, predictions, names, dataset):
    """Plot ROC curves for the models and save the plot to a file."""
    plt.figure(figsize=(10, 6))
```

```
    # predictions and y_tests are lists of model predictions and true labels
    for i in range(len(predictions)):
        fpr, tpr, _ = roc_curve(y_tests[i], predictions[i])
        roc_auc = auc(fpr, tpr)

        # Plot ROC curve for each model
        plt.plot(fpr, tpr, label=f'{names[i]} (AUC = {roc_auc:.2f})')

    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.savefig(f"../output/{dataset}/roc_curves.png")
    plt.close()
```

main.py

```
import exercise as e
import pandas as pd

if __name__ == "__main__":
    for i in range(1, 3):
        data_filepath = f'../data/data{i}.csv'
        df = pd.read_csv(data_filepath)


        model_lr, poly, results = e.cross_val_nested_loops(df, range(1, 10), [0.01, 0.1, 1,
            10, 100, 1000], i)
        e.plot_cross_val_results(results, 'lr', i)
        test_lr, predictions_lr = e.plot_and_save_results(df, model_lr, poly, 'lr', i)


        model_knn, results = e.train_knn_with_cross_validation(df, range(1, 50), i)
        e.plot_cross_val_results(results, 'knn', i)
        test_knn, predictions_knn = e.plot_and_save_results(df, model_knn, None, 'knn', i)


        model_dummy = e.train_dummy_classifier(df)
        test_dummy, predictions_dummy= e.plot_and_save_results(df, model_dummy, None, 'dummy
            ', i)

        e.plot_roc_curve(
            [test_lr, test_knn, test_dummy],
            [predictions_lr, predictions_knn, predictions_dummy],
            ['Logistic Regression', 'K-Nearest Neighbors', 'Dummy Classifier'],
            i
        )
```