CSU44061 Machine Learning - Week 3

Faith Olopade - 21364066
Dataset 1: # id:15-15-15-0
Dataset 2: # id:15-15-15-0

Appendix contains all the code.

Question (i)

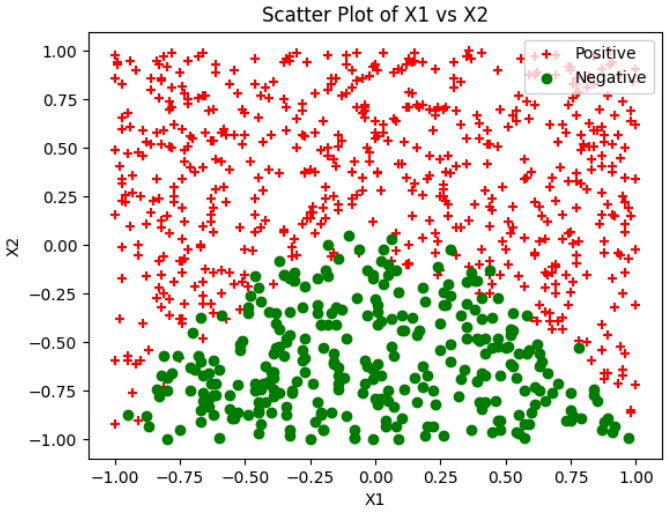

Figure 0: Plot of the data

(a): Logistic Regression with Polynomial Features and L2 Regularisation

We trained a logistic regression classifier with polynomial feature augmentation and L2 regularisation. The goal was to find the best polynomial degree and the optimal regularisation parameter (C). A grid search with cross-validation was used to tune these parameters.

Optimal Configuration:

- Polynomial degree: 4
- Regularisation strength (C): 10

The model achieved a cross-validation accuracy of 96%, demonstrating strong generalisation. As shown in Figure 1, the performance improves steadily as the polynomial degree increases, with the best performance at degree 4. Beyond this, the performance plateaus, indicating no further benefit from increasing complexity.

Justification:

- Polynomial Features: These help capture complex patterns in the dataset.
- L2 Regularisation: This prevents overfitting by penalising large weights, leading to better generalisation.
- Cross-Validation: This ensures the model performs consistently across different data splits, confirming robustness.

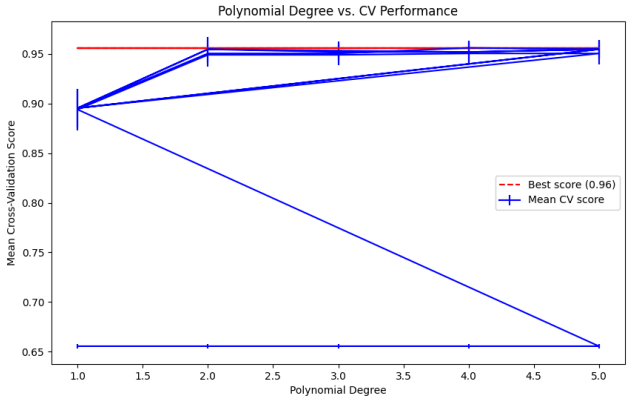

Figure 1: Polynomial Degree vs CV Performance

(b): k-Nearest Neighbors (kNN) Classifier

Next, we trained a kNN classifier to determine the optimal value for k. Cross-validation was performed over values of k from 1 to 30, and the best performance was achieved with k = 7, which provided the best balance between bias and variance.

Results:

- Cross-validation accuracy was highest with k = 7, as shown in Figure 2.

Justification:

- The selected k avoids overfitting while capturing complex patterns in the data.
- kNN can naturally handle nonlinear decision boundaries without needing additional feature transformations like polynomial features.

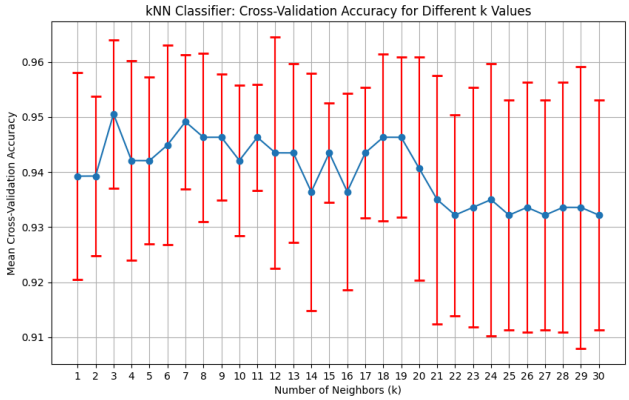

Figure 2: Cross-Validation Accuracy for Different k Values

(c): Confusion Matrix Comparison

We evaluated the performance of both the logistic regression and kNN models by calculating their confusion matrices. Two baseline classifiers were also compared: a Most Frequent Classifier and a Random Classifier.

```
Confusion Matrix for Logistic Regression:
 [[ 55   5]
 [  2 115]]
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

          -1       0.96      0.92      0.94        60
           1       0.96      0.98      0.97       117

    accuracy                           0.96       177
   macro avg       0.96      0.95      0.96       177
weighted avg       0.96      0.96      0.96       177
```

Logistic Regression:

- Accuracy: 96%
- The model had high precision and recall, with only 7 misclassifications, performing well in predicting both classes.

```
Confusion Matrix for kNN Classifier:
 [[ 55   5]
 [  7 110]]
Classification Report for kNN Classifier:
              precision    recall  f1-score   support

          -1       0.89      0.92      0.90        60
           1       0.96      0.94      0.95       117

    accuracy                           0.93       177
   macro avg       0.92      0.93      0.92       177
weighted avg       0.93      0.93      0.93       177
```

kNN Classifier:

- Accuracy: 93%
- The kNN model performed slightly worse than logistic regression, with 12 misclassifications. It struggled slightly more with true positives.

```
Confusion Matrix for Most Frequent Classifier:
 [[  0  60]
 [  0 117]]
Classification Report for Most Frequent Classifier:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00        60
           1       0.66      1.00      0.80       117

    accuracy                           0.66       177
   macro avg       0.33      0.50      0.40       177
weighted avg       0.44      0.66      0.53       177
```

Most Frequent Classifier:

- Accuracy: 66%
- This classifier predicted the most frequent class, ignoring the minority class entirely, resulting in poor recall for the minority class.

```
Confusion Matrix for Random Classifier:
 [[30 30]
 [59 58]]
Classification Report for Random Classifier:
              precision    recall  f1-score   support

          -1       0.34      0.50      0.40        60
           1       0.66      0.50      0.57       117

    accuracy                           0.50       177
   macro avg       0.50      0.50      0.48       177
weighted avg       0.55      0.50      0.51       177
```

Random Classifier:

- Accuracy: 50%
- As expected, the random classifier had an accuracy close to random guessing, with a near-equal distribution of true and false positives and negatives.

These results suggest that logistic regression outperformed all other models, with kNN coming close but struggling with a slightly higher false-negative rate. The baseline classifiers performed poorly, particularly for the minority class.

(d): ROC Curve Analysis

We plotted the ROC curves for logistic regression, kNN, and the two baseline classifiers. As shown in Figure 3, logistic regression achieved the highest true positive rate at lower false positive rates, slightly outperforming kNN. Both classifiers performed significantly better than the baseline classifiers, which followed the diagonal, indicating minimal predictive power.

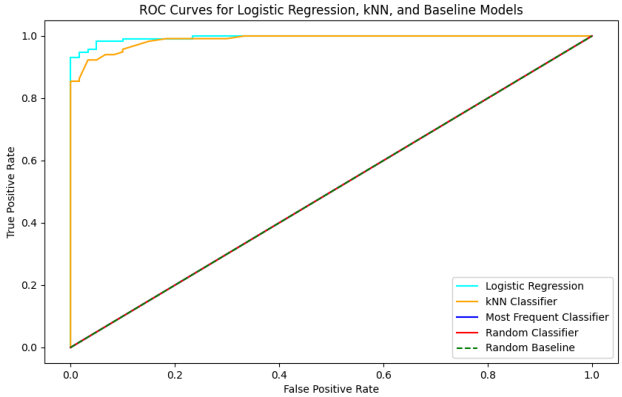

Figure 3: ROC Curves

(e)

Based on the confusion matrix and ROC curve analysis, here is a summary of the performance:

- Logistic Regression: Achieved the highest accuracy of 96%, with balanced performance across both classes.
- kNN Classifier: Achieved 93% accuracy but had a slightly higher rate of false negatives, impacting recall for the minority class.
- Most Frequent Classifier: Predicted only the majority class, with an accuracy of 66%.
- Random Classifier: As expected, its accuracy was around 50%, equivalent to random guessing.

ROC Curve Analysis: Logistic regression consistently outperformed kNN, particularly in managing false positives. The baseline classifiers performed poorly, with their ROC curves closely following the diagonal.

Recommendation: Logistic regression is the recommended classifier due to its high accuracy and better generalisation, especially when handling the trade-off between true positives and false positives. While kNN is competitive, logistic regression's balanced performance across different thresholds makes it the more reliable choice for this dataset.

Question (ii)

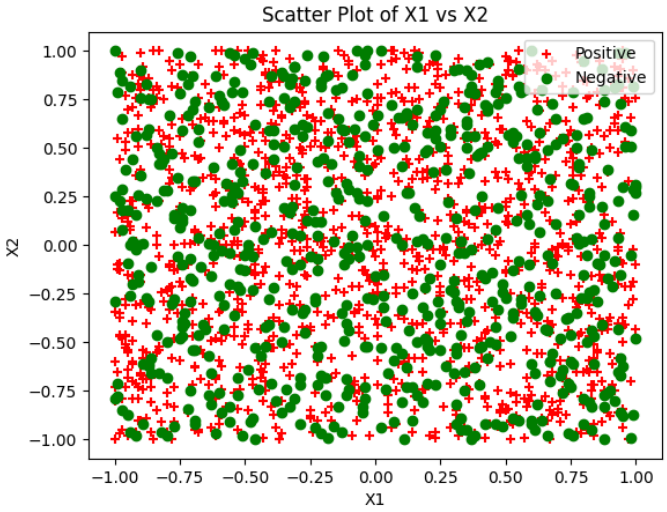

Figure 4: Plot of the data

(a): Logistic Regression with Polynomial Features and L2 Regularisation

For the second dataset, I followed a similar approach by augmenting features using polynomial transformations and applying logistic regression with L2 regularisation. A grid search with cross-validation was used to optimise the polynomial degree and regularisation strength (*C*).

Steps:

- Feature Augmentation: Polynomial features were added to capture complex relationships between features and the target.
- Regularisation: Logistic regression with an L2 penalty was used to avoid overfitting.
- Hyperparameter Tuning: A grid search over polynomial degrees from 1 to 5 and C values ranging from 0.001 to 1000 was performed.

Results:

- The best configuration was found with polynomial degree = 1 and C = 0.001, resulting in a cross-validation accuracy of 65% and test accuracy of 70%.
- Cross-validation results (Figure 5) show that increasing polynomial degree did not significantly improve performance. The model generalised better with simpler polynomial features.

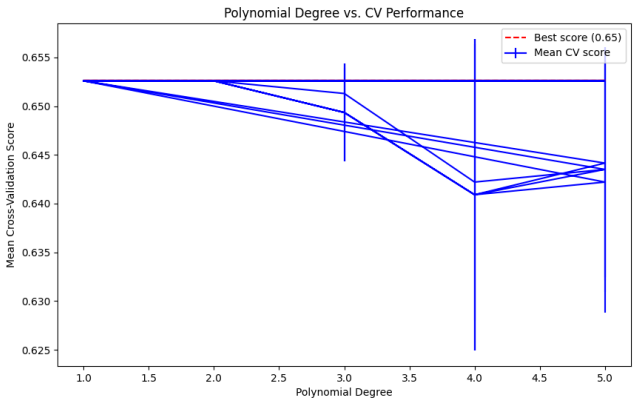

Figure 5: Polynomial Degree vs CV Performance

(b): kNN Classifier

For the kNN classifier, I tested values of *k* ranging from 1 to 30 and selected *k* = *7* based on cross-validation performance.

Results:

- The model achieved the highest cross-validation accuracy with k = 7, avoiding overfitting and capturing complex data patterns.

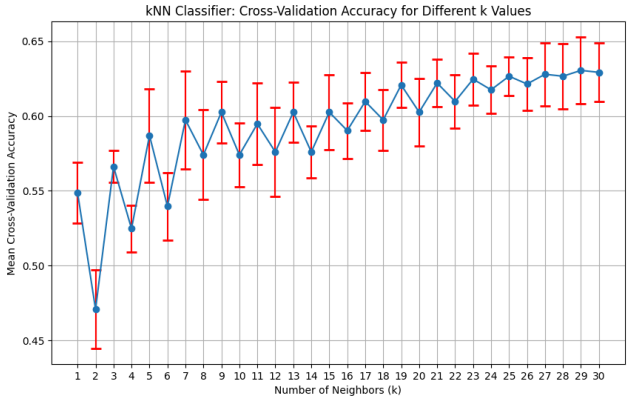

Figure 6: Cross-Validation Accuracy for Different k Values

(c): Confusion Matrix Comparison

The confusion matrices for logistic regression and kNN were calculated, along with baseline models (Most Frequent and Random).

```
Confusion Matrix for Logistic Regression:
 [[  0 116]
 [  0 269]]
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00       116
           1       0.70      1.00      0.82       269

    accuracy                           0.70       385
   macro avg       0.35      0.50      0.41       385
weighted avg       0.49      0.70      0.57       385
```

Logistic Regression: Achieved 70% accuracy but showed a significant bias toward the majority class (class 1), failing to predict any instances of class -1.

```
Confusion Matrix for kNN Classifier:
 [[  1 115]
 [ 12 257]]
Classification Report for kNN Classifier:
              precision    recall  f1-score   support

          -1       0.08      0.01      0.02       116
           1       0.69      0.96      0.80       269

    accuracy                           0.67       385
   macro avg       0.38      0.48      0.41       385
weighted avg       0.51      0.67      0.56       385
```

kNN Classifier: Achieved 67% accuracy with similar issues—failing to predict class -1 consistently.

```
Confusion Matrix for Most Frequent Classifier:
 [[  0 116]
 [  0 269]]
Classification Report for Most Frequent Classifier:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00       116
           1       0.70      1.00      0.82       269

    accuracy                           0.70       385
   macro avg       0.35      0.50      0.41       385
weighted avg       0.49      0.70      0.57       385
```

Most Frequent Classifier: Achieved 70% accuracy but only predicted the majority class.

```
Confusion Matrix for Random Classifier:
 [[ 58  58]
 [139 130]]
Classification Report for Random Classifier:
              precision    recall  f1-score   support

          -1       0.29      0.50      0.37       116
           1       0.69      0.48      0.57       269

    accuracy                           0.49       385
   macro avg       0.49      0.49      0.47       385
weighted avg       0.57      0.49      0.51       385
```

Random Classifier: Achieved 51% accuracy, with a more balanced distribution but overall poor performance.

(d)

The ROC curves for logistic regression, kNN, and baseline models are shown in Figure 7. Both logistic regression and kNN outperform the baselines, with logistic regression performing slightly better at lower false positive rates.

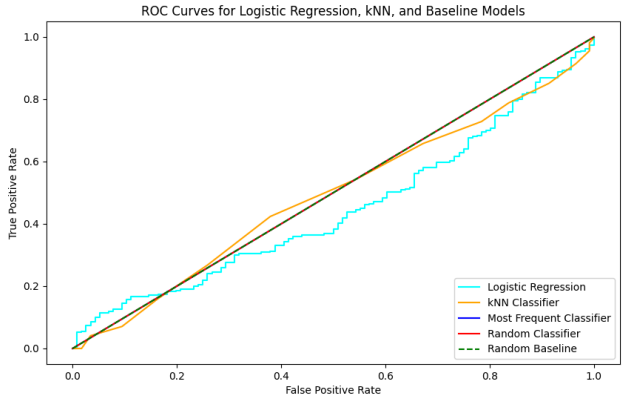

Figure 7: ROC Curves

(e)

Accuracy:

- Logistic Regression: 70%
- kNN Classifier: 67%
- Most Frequent Classifier: 70%
- Random Classifier: 51%

ROC Curves:

- Logistic regression had the best performance, followed closely by kNN.
- The baseline models performed poorly, with ROC curves following the diagonal.

Recommendation: Logistic regression is again the recommended model due to its better handling of the trade-off between true and false positives. While kNN is competitive, logistic regression showed better overall performance, making it the preferable choice.

Appendix

Question (i)

(a)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv('week4_d1.csv', header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

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

# Pipeline that includes polynomial feature creation and logistic
regression
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('logreg', LogisticRegression(penalty='l2', solver='lbfgs'))
])

# Parameters grid for cross-validation
param_grid = {
    'poly__degree': [1, 2, 3, 4, 5],  # Poly degrees to try
    'logreg__C': [0.001, 1, 10, 100, 1000]  # C vals
}

# Grid search with cross-validation
```

```
grid = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit  model
grid.fit(X_train, y_train)

# Best model's parameters and score
print("Best parameters:", grid.best_params_)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))

# Evaluate  model on  test set
test_score = grid.score(X_test, y_test)
print("Test set score: {:.2f}".format(test_score))

# Plot  results
results = pd.DataFrame(grid.cv_results_)
best_index = results['mean_test_score'].idxmax()
best_score = results.loc[best_index, 'mean_test_score']

plt.figure(figsize=(10, 6))
plt.errorbar(results['param_poly__degree'],
results['mean_test_score'], yerr=results['std_test_score'],
label='Mean CV score', color='blue')
plt.plot(results['param_poly__degree'],
np.full_like(results['param_poly__degree'], fill_value=best_score,
dtype=float), linestyle='--', color='red', label=f'Best score
({best_score:.2f})')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Cross-Validation Score')
plt.title('Polynomial Degree vs. CV Performance')
plt.legend()
plt.show()
```

(b)

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
```

```
# Function to evaluate kNN for different values of k
def evaluate_knn(k_range, X_train, y_train):
    mean_accuracy = []
    std_accuracy = []

    for k in k_range:
        # Initialize kNN model
        knn_model = KNeighborsClassifier(n_neighbors=k)

        # Perform cross-validation and get  accuracy scores
        scores = cross_val_score(knn_model, X_train, y_train, cv=5,
scoring='accuracy')

        # Store mean and standard deviation of  cross-validation
scores
        mean_accuracy.append(np.mean(scores))
        std_accuracy.append(np.std(scores))

    return mean_accuracy, std_accuracy

# Range of k values to test
k_range = range(1, 31)

# Evaluate kNN for different values of k
mean_accuracy, std_accuracy = evaluate_knn(k_range, X_train, y_train)

# Plot  cross-validation results
plt.figure(figsize=(10, 6))
plt.errorbar(k_range, mean_accuracy, yerr=std_accuracy, fmt='-o',
ecolor='red', capsize=5, capthick=2)
plt.title('kNN Classifier: Cross-Validation Accuracy for Different k
Values')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Cross-Validation Accuracy')
plt.xticks(k_range)
plt.grid(True)
```

```
plt.show()
```

(c)

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import PolynomialFeatures

# Logistic Regression Model
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
logreg_model = LogisticRegression(C=25).fit(X_train_poly, y_train)
logreg_predictions = logreg_model.predict(X_test_poly)

# kNN Model
knn_model = KNeighborsClassifier(n_neighbors=40).fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)

# Baseline Models
# Most Frequent Classifier
most_frequent_model =
DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
most_frequent_predictions = most_frequent_model.predict(X_test)

# Random Classifier
random_model = DummyClassifier(strategy="uniform").fit(X_train,
y_train)
random_predictions = random_model.predict(X_test)

# Confusion Matrices
print("Confusion Matrix for Logistic Regression:\n",
confusion_matrix(y_test, logreg_predictions))
print("Classification Report for Logistic Regression:\n",
classification_report(y_test, logreg_predictions))
```

```
print("Confusion Matrix for kNN Classifier:\n",
confusion_matrix(y_test, knn_predictions))
print("Classification Report for kNN Classifier:\n",
classification_report(y_test, knn_predictions))

print("Confusion Matrix for Most Frequent Classifier:\n",
confusion_matrix(y_test, most_frequent_predictions))
print("Classification Report for Most Frequent Classifier:\n",
classification_report(y_test, most_frequent_predictions))

print("Confusion Matrix for Random Classifier:\n",
confusion_matrix(y_test, random_predictions))
print("Classification Report for Random Classifier:\n",
classification_report(y_test, random_predictions))
```

(d)

```
from sklearn.metrics import roc_curve
# Logistic Regression predictions
logreg_probs = logreg_model.decision_function(X_test_poly)

# kNN predictions (using predict_proba)
knn_probs = knn_model.predict_proba(X_test)[:, 1]

# Baseline classifiers: Most Frequent and Random
most_frequent_model =
DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
random_model = DummyClassifier(strategy="uniform").fit(X_train,
y_train)

# Get the probabilities for baseline models
most_frequent_probs = most_frequent_model.predict_proba(X_test)[:, 1]
random_probs = random_model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve points
```

```
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_probs)
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_probs)
fpr_most_frequent, tpr_most_frequent, _ = roc_curve(y_test,
most_frequent_probs)
fpr_random, tpr_random, _ = roc_curve(y_test, random_probs)

# Plot the ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr_logreg, tpr_logreg, label='Logistic Regression',
color='cyan')
plt.plot(fpr_knn, tpr_knn, label='kNN Classifier', color='orange')
plt.plot(fpr_most_frequent, tpr_most_frequent, label='Most Frequent
Classifier', color='blue')
plt.plot(fpr_random, tpr_random, label='Random Classifier',
color='red')
plt.plot([0, 1], [0, 1], linestyle='--', color='green', label='Random
Baseline')

# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Logistic Regression, kNN, and Baseline
Models')
plt.legend(loc='lower right')

plt.show()
```

Question (ii)

(a)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv('week4_d2.csv', header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

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

# Pipeline that includes polynomial feature creation and logistic
regression
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('logreg', LogisticRegression(penalty='l2', solver='lbfgs'))
])

# Parameters grid for cross-validation
param_grid = {
    'poly__degree': [1, 2, 3, 4, 5],  # Poly degrees to try
    'logreg__C': [0.001, 1, 10, 100, 1000]  # C vals
}

# Grid search with cross-validation
grid = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')

# Fit  model
grid.fit(X_train, y_train)

# Best model's parameters and score
print("Best parameters:", grid.best_params_)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

```
# Evaluate  model on  test set
test_score = grid.score(X_test, y_test)
print("Test set score: {:.2f}".format(test_score))

# Plot  results
results = pd.DataFrame(grid.cv_results_)
best_index = results['mean_test_score'].idxmax()
best_score = results.loc[best_index, 'mean_test_score']

plt.figure(figsize=(10, 6))
plt.errorbar(results['param_poly__degree'],
results['mean_test_score'], yerr=results['std_test_score'],
label='Mean CV score', color='blue')
plt.plot(results['param_poly__degree'],
np.full_like(results['param_poly__degree'], fill_value=best_score,
dtype=float), linestyle='--', color='red', label=f'Best score
({best_score:.2f})')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Cross-Validation Score')
plt.title('Polynomial Degree vs. CV Performance')
plt.legend()
plt.show()
```

(b)

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
# Function to evaluate kNN for different values of k
def evaluate_knn(k_range, X_train, y_train):
    mean_accuracy = []
    std_accuracy = []

    for k in k_range:
        # Initialize kNN model
        knn_model = KNeighborsClassifier(n_neighbors=k)
```

```
        # Perform cross-validation and get  accuracy scores
        scores = cross_val_score(knn_model, X_train, y_train, cv=5,
scoring='accuracy')

        # Store mean and standard deviation of  cross-validation
scores
        mean_accuracy.append(np.mean(scores))
        std_accuracy.append(np.std(scores))

    return mean_accuracy, std_accuracy

# Range of k values to test
k_range = range(1, 31)

# Evaluate kNN for different values of k
mean_accuracy, std_accuracy = evaluate_knn(k_range, X_train, y_train)

# Plot  cross-validation results
plt.figure(figsize=(10, 6))
plt.errorbar(k_range, mean_accuracy, yerr=std_accuracy, fmt='-o',
ecolor='red', capsize=5, capthick=2)
plt.title('kNN Classifier: Cross-Validation Accuracy for Different k
Values')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Mean Cross-Validation Accuracy')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```

(c)

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import PolynomialFeatures

# Logistic Regression Model
```

```
poly = PolynomialFeatures(degree=3)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
logreg_model = LogisticRegression(C=25).fit(X_train_poly, y_train)
logreg_predictions = logreg_model.predict(X_test_poly)

# kNN Model
knn_model = KNeighborsClassifier(n_neighbors=40).fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)

# Baseline Models
# Most Frequent Classifier
most_frequent_model =
DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
most_frequent_predictions = most_frequent_model.predict(X_test)

# Random Classifier
random_model = DummyClassifier(strategy="uniform").fit(X_train,
y_train)
random_predictions = random_model.predict(X_test)

# Confusion Matrices
print("Confusion Matrix for Logistic Regression:\n",
confusion_matrix(y_test, logreg_predictions))
print("Classification Report for Logistic Regression:\n",
classification_report(y_test, logreg_predictions))

print("Confusion Matrix for kNN Classifier:\n",
confusion_matrix(y_test, knn_predictions))
print("Classification Report for kNN Classifier:\n",
classification_report(y_test, knn_predictions))

print("Confusion Matrix for Most Frequent Classifier:\n",
confusion_matrix(y_test, most_frequent_predictions))
print("Classification Report for Most Frequent Classifier:\n",
classification_report(y_test, most_frequent_predictions))
```

```
print("Confusion Matrix for Random Classifier:\n",
confusion_matrix(y_test, random_predictions))
print("Classification Report for Random Classifier:\n",
classification_report(y_test, random_predictions))
```

(d)

```
from sklearn.metrics import roc_curve
# Logistic Regression predictions
logreg_probs = logreg_model.decision_function(X_test_poly)

# kNN predictions (using predict_proba)
knn_probs = knn_model.predict_proba(X_test)[:, 1]

# Baseline classifiers: Most Frequent and Random
most_frequent_model =
DummyClassifier(strategy="most_frequent").fit(X_train, y_train)
random_model = DummyClassifier(strategy="uniform").fit(X_train,
y_train)

# Get the probabilities for baseline models
most_frequent_probs = most_frequent_model.predict_proba(X_test)[:, 1]
random_probs = random_model.predict_proba(X_test)[:, 1]

# Calculate the ROC curve points
fpr_logreg, tpr_logreg, _ = roc_curve(y_test, logreg_probs)
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_probs)
fpr_most_frequent, tpr_most_frequent, _ = roc_curve(y_test,
most_frequent_probs)
fpr_random, tpr_random, _ = roc_curve(y_test, random_probs)

# Plot the ROC curves
plt.figure(figsize=(10, 6))
```

```
plt.plot(fpr_logreg, tpr_logreg, label='Logistic Regression',
color='cyan')
plt.plot(fpr_knn, tpr_knn, label='kNN Classifier', color='orange')
plt.plot(fpr_most_frequent, tpr_most_frequent, label='Most Frequent
Classifier', color='blue')
plt.plot(fpr_random, tpr_random, label='Random Classifier',
color='red')
plt.plot([0, 1], [0, 1], linestyle='--', color='green', label='Random
Baseline')

# Customize the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Logistic Regression, kNN, and Baseline
Models')
plt.legend(loc='lower right')

plt.show()
```