Artificial Intelligence
Project Report

Hammad Suleman Khan (2012292)
Bilal Pervez (2012291)
Ali Asar Khowaja (2012285)
Muhammad Sarim Effendi (2012304)

Algorithms Implemented:

Logistic Regression

Logistic Regression is a popular statistical model used for binary classification tasks, where the goal is to predict the probability of an event belonging to one of two possible classes. It is a supervised learning algorithm that is widely used in various fields, including machine learning, statistics, and social sciences. Here's a detailed explanation of Logistic Regression:

1. **Introduction:**
   Logistic Regression is based on the concept of linear regression, but with a different approach to handle binary classification problems. It models the relationship between the input features and the probability of the target variable belonging to a particular class.

2. **Assumptions:**
   Logistic Regression makes several assumptions:
   - Binary Outcome: The target variable must be binary, meaning it can take only two values or be transformed into a binary form.
   - Linearity: There should be a linear relationship between the input features and the log-odds of the target variable.
   - Independence of Errors: The errors in the model should be independent of each other.
   - No Multicollinearity: The input features should not be highly correlated with each other.

3. **Logistic Function (Sigmoid Function):**
   Logistic Regression uses the logistic function, also known as the sigmoid function, to model the relationship between the input features and the target variable. The logistic function takes any real-valued number and maps it to a value between 0 and 1, representing the probability of the event.

4. **Model Representation:**
   In Logistic Regression, the model representation can be written as:
   p(X) = 1 / (1 + e^(-z))
   where p(X) is the predicted probability of the event, X is the input feature vector, and z is the linear combination of the input features and their corresponding coefficients.

5. **Training the Model:**
   The goal of training the Logistic Regression model is to estimate the coefficients (weights) that best fit the data. This is typically done using maximum likelihood estimation or optimization algorithms such as gradient descent. The model is trained by minimizing the error between the predicted probabilities and the actual class labels.

6. **Decision Boundary:**
   Logistic Regression uses a decision boundary to classify the instances into one of the two classes. The decision boundary is a threshold probability value (usually 0.5), above which the instance is classified as one class, and below which it is classified as the other class.

7. **Model Evaluation:**
   Logistic Regression can be evaluated using various performance metrics, including:
   - Accuracy: The proportion of correctly classified instances.
   - Precision: The ability of the model to correctly identify positive instances.
   - Recall (Sensitivity): The ability of the model to correctly identify all positive instances.
   - F1-score: The harmonic mean of precision and recall, providing a balanced measure between the two.

8. **Regularization:**
   Logistic Regression can be regularized to prevent overfitting. Regularization techniques like L1 regularization (Lasso) and L2 regularization (Ridge) can be applied to penalize large coefficient values and reduce the complexity of the model.

9. **Multiclass Logistic Regression:**
   While Logistic Regression is primarily used for binary classification, it can be extended to handle multiclass classification problems using techniques like One-vs-Rest (OvR) or Multinomial Logistic Regression.

10. **Interpretability:**
    One advantage of Logistic Regression is its interpretability. The coefficients of the model indicate the impact of each input feature on the predicted probability. Positive coefficients indicate a positive association, while negative coefficients indicate a negative association.

Overall, Logistic Regression is a powerful and interpretable algorithm for binary classification tasks. It is widely used due to its simplicity, efficiency, and effectiveness in various domains.

How can logistic regression be improved?

When applying Logistic Regression on the Titanic dataset, there are several ways to improve the performance of the algorithm:

1. **Data Preprocessing:**
    - Missing Data: Handle missing values in the dataset by imputing them with appropriate techniques like mean, median, or mode.
    - Feature Scaling: Scale the numerical features to a similar range to avoid dominance of one feature over others. Common scaling techniques include standardization (mean=0, variance=1) or normalization (0-1 range).
    - Categorical Variables: Encode categorical variables into numerical form using techniques like one-hot encoding or label encoding.

2. **Feature Engineering:**
    - Create new features: Extract meaningful information from existing features or create new features based on domain knowledge. For example, extract the title from the passenger's name or create a feature indicating whether a passenger is traveling alone or with family.
    - Feature Selection: Select the most relevant features that contribute significantly to the target variable and remove irrelevant or redundant features. This helps reduce dimensionality and improve model efficiency.

3. **Handling Imbalanced Data:**
   If the dataset has imbalanced classes (e.g., a significant difference in the number of survivors and non-survivors), consider techniques like oversampling the minority class (e.g., using SMOTE) or undersampling the majority class to balance the dataset.

4. **Regularization:**
   Apply regularization techniques like L1 or L2 regularization to prevent overfitting. Regularization helps in reducing the impact of irrelevant features and prevents the model from memorizing noise in the data.

5. **Hyperparameter Tuning:**
   Experiment with different hyperparameter values like the regularization parameter (C), solver type, and maximum number of iterations. Use techniques like cross-validation to find the optimal set of hyperparameters that yield the best performance.

6. **Model Evaluation and Comparison:**
   Evaluate the model's performance using appropriate evaluation metrics like accuracy, precision, recall, and F1-score. Compare the performance of Logistic Regression with other classification algorithms to determine if there are alternative models that provide better results for the Titanic dataset.

It's important to note that the performance improvement techniques may vary based on the characteristics of the dataset and the specific requirements of the problem. A thorough analysis of the data and iterative experimentation with different approaches will help in optimizing the performance of Logistic Regression on the Titanic dataset.

K Nearest Neighbor

K-Nearest Neighbors (KNN) is a simple yet powerful supervised learning algorithm used for both classification and regression tasks. It operates on the principle that similar instances or data points tend to belong to the same class or have similar output values.

1. **Working Principle:**
   - KNN determines the class or value of a new data point by analyzing its k nearest neighbors in the training dataset.
   - It calculates the distance between the new data point and all other points in the training set using a distance metric such as Euclidean distance or Manhattan distance.
   - The k nearest neighbors are identified based on the smallest distances, and the majority class or average value of these neighbors is assigned to the new data point.

2. **Choosing the Value of K:**
   - The parameter 'k' represents the number of nearest neighbors to consider.
   - The selection of k is crucial as it influences the algorithm's behavior. A smaller k value can make the model more sensitive to noise, while a larger k value can result in a smoother decision boundary but may overlook local patterns.
   - The optimal value of k is typically determined through experimentation and cross-validation.

3. **Distance Metrics:**
   - KNN relies on distance metrics to measure the similarity between data points.
   - Euclidean distance is the most commonly used metric, computed as the square root of the sum of squared differences between corresponding features of two points.
   - Manhattan distance is an alternative metric that calculates the sum of absolute differences between corresponding features.

- Other distance metrics, such as Minkowski distance, can be used depending on the problem domain and data characteristics.

4. **Data Preprocessing:**
    - Like most machine learning algorithms, KNN benefits from data preprocessing steps.
    - Feature scaling is essential to ensure that all features contribute equally to the distance calculation. Scaling techniques like standardization (mean=0, variance=1) or normalization (0-1 range) can be applied.
    - Handling categorical variables may require encoding techniques such as one-hot encoding or label encoding.

5. **Decision Rule:**
    - For classification problems, the majority class among the k neighbors is assigned to the new data point.
    - In regression problems, the average or weighted average of the target values of the k neighbors is assigned as the predicted value.

6. **Computational Complexity:**
    - KNN has a relatively high computational cost compared to other algorithms.
    - During prediction, it requires calculating distances for each new data point against all training data points, which can be time-consuming for large datasets.
    - Techniques like KD-trees or Ball trees can be used to optimize the searching process and improve the algorithm's efficiency.

7. **Choosing the Optimal K:**
    - The selection of the optimal k value depends on the dataset and problem at hand.
    - It can be determined using techniques like cross-validation, where different values of k are evaluated, and the one with the best performance is chosen.
    - It's important to consider the trade-off between bias and variance when selecting k. Smaller values of k tend to result in low bias and high variance, while larger values of k lead to high bias and low variance.

KNN is a versatile algorithm that is relatively easy to understand and implement. However, it has certain limitations, such as sensitivity to irrelevant features, high memory usage for large datasets, and lack of interpretability. Understanding the principles and considerations discussed above will help in effectively applying and optimizing the KNN algorithm for various tasks.

How can KNN be improved?

1. Feature Selection:
    - Analyze and select relevant features that have a strong correlation with the target variable (survival in this case) and remove irrelevant or redundant features.
    - Perform exploratory data analysis (EDA) to identify features that provide meaningful insights and contribute significantly to the prediction.

2. Feature Engineering:
    - Create new features that might capture important patterns or relationships in the data. For example, you could extract the title from the passenger names or create a feature indicating whether a passenger is traveling alone or with family.
    - Consider binning or discretizing continuous variables to transform them into categorical features if it improves the performance.

3. Data Cleaning:
    - Handle missing values appropriately. Depending on the amount and nature of missing data, you can either remove instances with missing values, impute missing values using techniques like mean, median, or regression imputation, or create an additional indicator variable to represent missing values.
    - Deal with outliers that might negatively impact the algorithm's performance. You can remove outliers or apply transformation techniques to reduce their influence.

4. Feature Scaling:
    - Normalize or standardize the feature values to ensure that each feature contributes equally to the distance calculation in KNN.
    - Scaling can prevent features with larger ranges from dominating the distance metric and affecting the algorithm's performance.

5. Optimal Value of K:
    - Experiment with different values of k to find the optimal choice. Perform cross-validation to evaluate the model's performance for different k values.
    - Be cautious of overfitting (using a very small k) or underfitting (using a very large k). Choose a value of k that balances bias and variance and provides the best generalization performance.

6. Model Evaluation and Comparison:
    - Besides accuracy, consider evaluating the KNN model's performance using other metrics such as precision, recall, and F1-score.
    - Compare the performance of KNN with other classification algorithms suitable for the Titanic dataset, such as logistic regression, decision trees, random forests, or support vector machines. This can help identify which algorithm performs better for this specific dataset.

7. Ensemble Techniques:

- Implement ensemble methods like bagging or boosting, which combine multiple KNN models to improve overall performance.
- Bagging techniques, such as Random Forests, create an ensemble of KNN models trained on different subsets of the data to reduce variance and improve prediction accuracy.
- Boosting techniques, such as AdaBoost or Gradient Boosting, sequentially train KNN models, giving more weight to misclassified instances, thereby improving overall prediction performance.

Remember that the choice of improvements depends on the characteristics of the Titanic dataset and the specific goals of the analysis. It's recommended to iterate through these steps, evaluate the impact of each improvement, and assess the model's performance to ultimately achieve the best possible results.

Decision Trees

Decision trees are a popular machine learning algorithm used for both classification and regression tasks. They are intuitive to understand and interpret, making them valuable for decision-making. Here's a detailed explanation of decision trees:

1. **Overview:**
    - A decision tree is a hierarchical structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome or prediction.
    - The tree is built by recursively partitioning the data based on the values of different features, aiming to create homogeneous subsets of data at each node.
    - Decision trees can handle both categorical and numerical features and can handle missing values.

2. **Splitting Criteria:**
    - At each internal node, the decision tree algorithm selects the best feature to split the data. This is done based on a splitting criterion, which measures the quality of the split.
    - Popular splitting criteria include Gini impurity and entropy for classification tasks, and mean squared error or mean absolute error for regression tasks.
    - The splitting criterion aims to maximize the homogeneity of the subsets created by the split, resulting in pure nodes or nodes with minimal impurity.

3. **Building the Tree:**
    - Decision trees are typically built using a top-down, recursive process known as recursive binary splitting.

- The algorithm starts with the root node containing the entire dataset and recursively splits the data based on the selected features until a stopping criterion is met.
- Stopping criteria can include reaching a maximum depth, minimum number of samples per leaf, or achieving a certain level of purity.

4. **Pruning:**
    - Decision trees are prone to overfitting, which means they can learn the training data too well and perform poorly on unseen data.
    - Pruning is a technique used to reduce overfitting by simplifying the tree structure. It involves removing nodes or branches that do not significantly improve the tree's predictive performance.
    - Pruning can be based on various metrics, such as cost complexity pruning (using a cost function) or reduced error pruning (using a validation set).

5. **Interpretability:**
    - One of the major advantages of decision trees is their interpretability. The tree structure allows easy understanding of the decision-making process.
    - Decision trees can be visualized, showing the path from the root to each leaf and the decision rules at each node.
    - Feature importance can be determined by assessing the impact of each feature on the tree's performance, such as measuring the decrease in impurity or the number of times a feature is used for splitting.

6. **Handling Overfitting:**
    - Decision trees are prone to overfitting, especially if the tree grows too deep or the splitting criteria are too strict.
    - Techniques such as pruning, limiting the maximum depth, setting a minimum number of samples per leaf, or using ensemble methods like random forests can help mitigate overfitting.

7. **Handling Categorical and Numerical Features:**
    - Decision trees can handle both categorical and numerical features.
    - For categorical features, the tree considers each category as a separate branch, and for numerical features, the tree selects a threshold to split the data into two subsets.
    - Various algorithms exist to determine the best splitting point for numerical features, such as binary splitting or multiway splitting.

8. **Handling Missing Values:**
    - Decision trees can handle missing values in the data.
    - Different strategies can be employed, such as ignoring missing values, treating missing values as a separate category, or imputing missing values using techniques like mean, median, or regression imputation.

9. **Advantages of Decision Trees:**
    - Easy to understand and interpret, making them useful for decision-making and providing insights into the data.
    - Can handle both categorical and numerical features, as well as missing values.
    - Nonlinear relationships and interactions between features can be captured by the tree structure.
    - Can handle irrelevant features without affecting the overall performance.
    - Can be used for both classification and regression tasks.

10. **Limitations of Decision Trees:**
    - Decision trees are prone to overfitting, especially when the tree grows too deep or the dataset has noisy or irrelevant features.
    - Decision trees can be sensitive to small variations in the data, leading to different trees and potentially overfitting.
    - Decision trees can struggle with capturing complex relationships or patterns that require multiple levels of splitting.
    - Decision trees can be biased towards features with more levels or categories.
    - Decision trees are not very robust to small changes in the data and may produce different results.

Overall, decision trees are a versatile and interpretable algorithm that can provide valuable insights and perform well on various datasets. However, careful tuning and handling of overfitting are crucial to ensure their optimal performance.

How can the Decision Tree be improved?

1. **Feature Engineering:**
   Analyze and preprocess the existing features to create new informative features. For example:
    - Extract titles from the "Name" feature to capture social status (e.g., Mr., Mrs., Miss).
    - Create a new feature by combining "SibSp" and "Parch" to represent family size.
    - Group or categorize continuous features like "Age" or "Fare" into meaningful intervals.

2. **Handling Missing Values:**
   Address missing values in the dataset. Some options include:
    - Imputing missing values with statistical measures like mean, median, or mode.
    - Creating a separate category for missing values, if applicable.
    - Utilizing advanced imputation techniques like regression imputation.

3. **Feature Selection:**
   Identify and select the most relevant features for training the decision tree. Consider using techniques like:
   - Univariate feature selection based on statistical tests or feature importance scores.
   - Recursive feature elimination to iteratively select the best subset of features.

4. **Handling Categorical Variables:**
   Encode categorical variables in a suitable format for the decision tree algorithm. Options include:
   - One-Hot Encoding: Create binary columns for each category.
   - Label Encoding: Assign unique integer labels to each category.

5. **Pruning:**
   Apply pruning techniques to reduce overfitting and improve generalization. This can involve:
   - Setting a maximum depth for the tree to prevent it from growing too deep.
   - Specifying a minimum number of samples required for further splitting.
   - Using cost complexity pruning to find an optimal trade-off between tree complexity and performance.

6. **Ensemble Methods:**
   Consider using ensemble methods, such as Random Forests, which combine multiple decision trees to improve predictive accuracy and reduce overfitting. This can be done by:
   - Training an ensemble of decision trees on random subsets of the data.
   - Aggregating predictions from multiple trees to make final predictions.

7. **Hyperparameter Tuning:**
   Experiment with different hyperparameter settings of the decision tree algorithm. Perform a grid search or random search to find the optimal combination of hyperparameters. Key parameters to consider include:
   - Maximum depth of the tree.
   - Minimum number of samples required for a split.
   - Splitting criteria (e.g., Gini impurity or entropy).

8. **Cross-Validation:**
   Use cross-validation techniques to evaluate the performance of the decision tree algorithm on different subsets of the data. This helps assess the model's generalization ability and detect potential issues with overfitting.

9. **Evaluate Different Metrics:**
   Besides accuracy, consider evaluating the decision tree model using other metrics like precision, recall, and F1-score. This is especially relevant for imbalanced datasets like

the Titanic dataset, where the number of survivors and non-survivors is not evenly distributed.

By implementing these strategies, you can enhance the performance of the decision tree algorithm on the Titanic dataset and improve its predictive capabilities. Remember that it's important to experiment and iterate on these approaches based on the specific characteristics of the dataset and the goals of your analysis.

Neural Networks

Neural networks, also known as artificial neural networks (ANNs), are a class of machine learning algorithms inspired by the structure and functioning of biological brains. They consist of interconnected nodes, called artificial neurons or units, organized in layers. Neural networks are powerful models capable of learning complex patterns and relationships in data.

Here is an explanation of each component and concept in a neural network algorithm:

1. **Artificial Neuron (Node):**
   An artificial neuron, also known as a node or unit, represents a simplified model of a biological neuron. It takes multiple input signals, applies weights to them, performs a computation, and produces an output signal.

2. **Activation Function:**
   An activation function determines the output of a neuron based on its input. It introduces non-linearity into the network, allowing it to learn and represent complex relationships. Common activation functions include sigmoid, tanh, ReLU, and softmax.

3. **Layers:**
   Neural networks are organized into layers of interconnected neurons. The three main types of layers are:
   - Input Layer: The initial layer that receives the input data.
   - Hidden Layers: Intermediate layers between the input and output layers. They extract and transform features from the input data.
   - Output Layer: The final layer that produces the network's output, such as class probabilities in classification tasks.

4. **Weights and Biases:**
   Each connection between neurons in the network has an associated weight. Weights determine the strength of the connection and influence the impact of each input on the neuron's computation. Biases are additional parameters that adjust the output of a neuron, allowing it to learn different representations.

5. **Forward Propagation:**
   In the forward propagation step, input data flows through the network from the input layer to the output layer. Each neuron performs a computation using its inputs, weights, and biases, and passes the result to the next layer. This process is repeated layer by layer until the output is generated.

6. **Loss Function:**
   A loss function quantifies the difference between the predicted output of the network and the actual target output. It measures the network's performance and provides a signal for adjusting the weights during training. Common loss functions include mean squared error (MSE) for regression and categorical cross-entropy for classification.

7. **Backpropagation:**
   Backpropagation is the key algorithm used to train neural networks. It involves computing the gradient of the loss function with respect to the network's weights. The gradient is then used to update the weights in the opposite direction of the gradient, enabling the network to learn from its mistakes.

8. **Training:**
   During training, the network learns to adjust its weights and biases by iteratively feeding the input data, propagating the signals forward, computing the loss, and backpropagating the gradients. This process aims to minimize the loss and improve the network's predictions.

9. **Optimization Algorithms:**
   Optimization algorithms, such as stochastic gradient descent (SGD) and its variants (e.g., Adam, RMSprop), are used to update the network's weights during training. These algorithms adjust the weights based on the gradients computed by backpropagation, gradually guiding the network towards better performance.

10. **Hyperparameters:**
    Hyperparameters are parameters set before training that control the behavior of the network. They include the number of hidden layers, the number of neurons in each layer, the learning rate, batch size, and regularization techniques. Tuning hyperparameters is crucial to achieve optimal performance.

11. **Overfitting:** Overfitting occurs when a neural network performs well on the training data but fails to generalize to new, unseen data. Regularization techniques like dropout and L1/L2 regularization can be applied to prevent overfitting and improve generalization.

12. **Evaluation:** Once trained, the network is evaluated on a separate test set to assess its performance. Metrics such as accuracy, precision, recall, and F1-score are used to

evaluate classification models, while regression models may use metrics like mean squared error or R-squared.

Neural networks are versatile models that can be applied to various tasks, including image recognition, natural language processing, time series analysis, and more. However, building an effective neural network often requires experimentation, fine-tuning, and considering the characteristics of the specific problem and dataset at hand.

How can Neural Networks be improved?

1. **Feature Engineering:** Carefully analyze and preprocess the dataset to extract meaningful features that can better represent the underlying patterns. This may involve creating new features, combining existing ones, or transforming variables. For example, you can extract titles from passenger names or create binary indicators for missing values.

2. **Handling Missing Data:** Develop strategies to handle missing data appropriately. Depending on the extent of missingness, you can choose to drop columns with high missing values, impute missing values using statistical methods, or create separate categories for missing data.

3. **Scaling and Normalization:** Scale numerical features to a common range, such as 0 to 1, using techniques like min-max scaling or standardization. This ensures that features with different scales do not unduly influence the model's learning process.

4. **Handling Categorical Variables:** Convert categorical variables into numerical representations that the algorithm can understand. This can be achieved through one-hot encoding, label encoding, or ordinal encoding. Be cautious when encoding ordinal variables, as the chosen encoding should reflect the variable's inherent order.

5. **Feature Selection:** Identify and select the most relevant features for training the model. Consider using techniques such as correlation analysis, stepwise selection, or regularization methods to exclude irrelevant or redundant features that may introduce noise.

6. **Cross-Validation:** Employ cross-validation techniques to evaluate the algorithm's performance more robustly. Cross-validation helps estimate how well the model generalizes to unseen data by splitting the dataset into multiple train-test folds and averaging the evaluation metrics across them.

7. **Hyperparameter Tuning:** Optimize the hyperparameters of the logistic regression algorithm. This can be done using techniques like grid search, random search, or Bayesian optimization. Hyperparameters to tune may include regularization strength, learning rate, maximum number of iterations, or convergence criteria.

8. **Ensemble Methods**: Consider using ensemble methods, such as bagging or boosting, to combine multiple logistic regression models for improved performance. Ensemble methods can help reduce variance, increase model stability, and capture diverse patterns in the dataset.

9. **Addressing Class Imbalance:** If the dataset exhibits class imbalance, where one class dominates the other, apply techniques to address this issue. Methods like oversampling the minority class, undersampling the majority class, or using algorithms designed for imbalanced datasets, such as SMOTE, can help improve the model's ability to predict the minority class.

10. **Regularization:** Apply regularization techniques, such as L1 or L2 regularization, to prevent overfitting and improve the model's generalization. Regularization helps control the complexity of the model by penalizing large weights or restricting the number of features used.

It is important to note that the effectiveness of these strategies may vary depending on the specific characteristics of the Titanic dataset. Experimentation and iterative refinement are key to finding the best combination of techniques for improving algorithm performance.

```
In [72]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=11)

In [73]: knn.fit(X_train, y_train)

In [74]: knn.score(X_test, y_test)
Out[74]: 0.6883116883116883

In [72]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=10)

In [73]: knn.fit(X_train, y_train)

In [74]: knn.score(X_test, y_test)
Out[74]: 0.7142857142857143
```

```
[99]: from sklearn.neighbors import KNeighborsClassifier
      knn = KNeighborsClassifier(n_neighbors=12)

100]: knn.fit(X_train, y_train)

      C:\Users\rajab\anaconda3\lib\site-packages\sklearn\
      as passed when a 1d array was expected. Please chan
        return self._fit(X, y)

100]: KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=12)

101]: knn.score(X_test, y_test)

101]: 0.7835497835497836

[124]: from sklearn.neighbors import KNeighborsClassifier
       knn = KNeighborsClassifier(n_neighbors=14)

[125]: knn.fit(X_train, y_train)

[126]: knn.score(X_test, y_test)

[126]: 0.8051948051948052

[127]: from sklearn.neighbors import KNeighborsClassifier
       knn = KNeighborsClassifier(n_neighbors=15)

[128]: knn.fit(X_train, y_train)

[129]: knn.score(X_test, y_test)

[129]: 0.7835497835497836
```

Note: accuracy goes down at 15 neighbors while is the best at 14

```
[142]: from sklearn.neighbors import KNeighborsClassifier
       knn = KNeighborsClassifier(n_neighbors=16)
```

```
[143]: knn.fit(X_train, y_train)
```

```
[144]: knn.score(X_test, y_test)
[144]: 0.8225108225108225
```

Note: improves again at 16 neighbors

Logistic Images

```
In [151]: report = classification_report(y_test, lr_predict)
          print(report)
```

	precision	recall	f1-score	support
0	0.90	0.82	0.86	171
1	0.59	0.75	0.66	60
accuracy			0.80	231
macro avg	0.75	0.78	0.76	231
weighted avg	0.82	0.80	0.81	231

```
lr = LogisticRegression(C=2)
```

```
In [160]: report = classification_report(y_test, lr_predict)
          print(report)
```

	precision	recall	f1-score	support
0	0.91	0.81	0.86	171
1	0.59	0.77	0.67	60
accuracy			0.80	231
macro avg	0.75	0.79	0.76	231
weighted avg	0.83	0.80	0.81	231

```
: accuracy=accuracy_score(y_test,lr_predict)
  accuracy
: 0.8008658008658008
```

Comparison between traditional machine learning algorithms and reinforcement learning algorithms:

It's important to note that while traditional machine learning algorithms focus on data prediction and classification tasks based on given input-output pairs, reinforcement learning algorithms deal with sequential decision-making problems where an agent learns to maximize rewards by interacting with an environment. Reinforcement learning involves the concept of exploring different actions and learning from delayed rewards to achieve long-term goals.

1. **Learning Paradigm:**

- Traditional Machine Learning Algorithms: These algorithms fall under the paradigm of supervised or unsupervised learning. In supervised learning, the algorithms learn from labeled training data to make predictions or classifications. In unsupervised learning, the algorithms find patterns or structure in unlabeled data.
- Reinforcement Learning Algorithms: Reinforcement learning is a trial-and-error learning paradigm. The algorithms learn by interacting with an environment, receiving feedback in the form of rewards or penalties based on their actions.

2. **Objective:**

- Traditional Machine Learning Algorithms: The objective of traditional machine learning algorithms is to predict or classify data accurately. They aim to find patterns or relationships in the given data.
- Reinforcement Learning Algorithms: The objective of reinforcement learning algorithms is to maximize cumulative rewards over time. The algorithms learn to make a series of actions that lead to the highest possible rewards in a given environment.

3. **Feedback:**

- Traditional Machine Learning Algorithms: Traditional algorithms rely on labeled training data or unsupervised techniques to learn from the data patterns.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms receive feedback in the form of scalar rewards or penalties based on their actions taken in the environment. The rewards indicate the desirability of the actions, while penalties indicate unfavorable outcomes.

4. **Training Process:**
    -
- Traditional Machine Learning Algorithms: Traditional algorithms typically involve offline batch learning. They learn from a fixed dataset and optimize their models based on that data.

- Reinforcement Learning Algorithms: Reinforcement learning algorithms involve online sequential learning. They interact with an environment in a sequential manner, updating their models based on the feedback received after each action.

5. Environment:

- Traditional Machine Learning Algorithms: Traditional algorithms do not explicitly model the environment. They focus on learning patterns in the data provided.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms explicitly model an environment. The algorithms make decisions based on the state of the environment and learn from the consequences of their actions.

6. **Decision Making**:

- Traditional Machine Learning Algorithms: Traditional algorithms make static decisions based on the input data they are provided. Once the model is trained, it can predict or classify new data based on the learned patterns.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms make dynamic decisions based on the current state of the environment. They consider the potential long-term consequences of their actions and aim to maximize their cumulative rewards over time.

7. Actions:

- Traditional Machine Learning Algorithms: Traditional algorithms do not typically involve explicit actions taken by the algorithms themselves.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms take a sequence of actions in the environment based on the learned policies. The actions can affect the state of the environment and subsequent rewards.

8. Exploration:

- Traditional Machine Learning Algorithms: Traditional algorithms do not typically involve explicit exploration of different actions or strategies.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms often include an exploration-exploitation tradeoff. They explore different actions to gather information about the environment and learn better strategies for maximizing rewards.

9. Feedback Delay:

- Traditional Machine Learning Algorithms: Traditional algorithms receive immediate feedback in the form of labeled training data or unsupervised learning techniques.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms receive delayed rewards or penalties that depend on the consequences of their actions. The feedback is often delayed as the agent learns from the long-term outcomes of its actions.

10. Reinforcement:

- Traditional Machine Learning Algorithms: The concept of reinforcement is not applicable in traditional machine learning algorithms.
- Reinforcement Learning Algorithms: Reinforcement learning algorithms learn through positive reinforcement (rewards) and negative reinforcement (penalties). The agent adjusts its actions based on the feedback received from the environment, which includes rewards or penalties. By maximizing the cumulative reward over time, the algorithm aims to learn the optimal policy or value function that guides its decision-making process.

Difference between RL and ML algos and how ML algorithms can be used for RL

The four models you mentioned, logistic regression, decision trees, K-nearest neighbors (KNN), and neural networks, are primarily used in traditional machine learning settings for supervised or unsupervised learning tasks. While they are not typically used directly for reinforcement learning, they can be incorporated into reinforcement learning frameworks in certain ways. Here's how they can be used:

1. **Logistic Regression:** Logistic regression is commonly used for binary classification problems in traditional machine learning. In the context of reinforcement learning, logistic regression can be used as a policy function to determine the probability of taking certain actions given a state in the environment. The logistic regression model can be trained using reinforcement learning techniques such as policy gradients or actor-critic methods to optimize the policy.

2. **Decision Trees:** Decision trees are versatile models that can be used for both classification and regression tasks. In reinforcement learning, decision trees can be employed as a function approximator for the value function or the policy. The decision tree can learn to make decisions based on the state of the environment and the potential rewards associated with different actions.

3. **K-nearest Neighbors (KNN)**: K-nearest neighbors is a non-parametric algorithm used for classification and regression tasks. In reinforcement learning, KNN can be used in a similar way as decision trees, acting as a function approximator for the value function or policy. KNN can capture the relationships between states and actions by finding the nearest neighbors in the state space and generalizing their associated rewards or policies.

4. **Neural Networks:** Neural networks are powerful models capable of learning complex patterns and relationships. In reinforcement learning, neural networks are commonly used as function approximators for value functions, policy functions, or both. The neural network can take the current state of the environment as input and output the predicted values or probabilities of different actions. Through reinforcement learning algorithms like

deep Q-learning or policy gradients, the neural network can be trained to improve decision-making in the environment.

It's important to note that while these models can be used within reinforcement learning frameworks, they are often combined with specific reinforcement learning algorithms and techniques to handle the sequential decision-making process, reward feedback, and exploration-exploitation tradeoffs inherent in reinforcement learning problems.

PPO (Proximal Policy Optimization) and A2C (Advantage Actor-Critic) are reinforcement learning algorithms that differ from traditional machine learning algorithms in several ways. Here are the main differences:

1. **Training approach:**

- Traditional Machine Learning Algorithms: Traditional machine learning algorithms typically learn from a static dataset, where the data is independently and identically distributed (i.i.d.).
- PPO and A2C: These algorithms employ an online training approach, where the agent interacts with the environment in real-time, gathering experience and updating its policy or value function accordingly.

2. **Feedback mechanism:**

- Traditional Machine Learning Algorithms: Traditional algorithms receive feedback in the form of labeled training data, typically provided by humans.
- PPO and A2C: These reinforcement learning algorithms receive feedback from the environment through rewards or penalties. The agent learns by trial and error, adjusting its policy or value function based on the outcomes of its actions.

3. **Exploration vs. Exploitation:**

- Traditional Machine Learning Algorithms: Traditional algorithms focus on optimizing performance on the given dataset, without explicitly considering the trade-off between exploration and exploitation.
- PPO and A2C: These algorithms address the exploration-exploitation dilemma by actively exploring the environment to discover new strategies while also exploiting known successful actions.

4. **Policy representation:**

- Traditional Machine Learning Algorithms: Traditional algorithms often rely on fixed, predefined models or representations for decision-making.
- PPO and A2C: These reinforcement learning algorithms typically use parameterized policies that can be continuously updated and refined during training.

5. **Temporal aspect:**

- Traditional Machine Learning Algorithms: Traditional algorithms do not explicitly consider the temporal aspect of decision-making or the sequential nature of actions.
- PPO and A2C: These algorithms take into account the sequential and temporal nature of actions, as they aim to optimize long-term cumulative rewards.

Overall, PPO and A2C are specialized reinforcement learning algorithms designed to learn from interacting with the environment, receiving feedback in the form of rewards, and optimizing decision-making policies over time. They offer a different approach compared to traditional machine learning algorithms that operate on static datasets with labeled examples.