Report Task 1: Choices for Running K-means and GMM

K-means Choices and Rationale

1. **Initialization Method**
   - Use `k-means++` as the initial cluster center selection strategy.
   - Reason: The `k-means++` method effectively reduces issues with uneven cluster center distribution during initialization, avoiding poor clustering results due to biased initial values.
2. **Number of Clusters**
   - Set `n_clusters=7` to align with data characteristics.
   - Reason: Based on the task requirement, we assume the data is divided into 7 classes, but we don't directly rely on labels; instead, we use feature diensions and actual data distribution assumptions.
3. **Number of Initializations**
   - Increase `n_init=50`, meaning the K-means algorithm runs multiple times to select the best result.
   - Reason: Multiple initializations can effectively alleviate issues where a single run may get stuck in a local optimum.
4. **Acceleration Algorithm**
   - Use `elkan` to accelerate clustering update calculations.
   - Reason: On high-dimensional data, the `elkan` algorithm optimizes the iteration process using the triangle inequality, improving efficiency.

GMM Choices and Rationale

1. **Initialization Parameters**
   - Use `kmeans` as the initialization method for GMM.
   - Reason: Providing initial means through K-means helps GMM quickly converge to a better solution.
2. **Number of Components**
   - Set `n_components=7`, consistent with the number of clusters in K-means.
   - Reason: Assume the data naturally divides into 7 classes, without relying on label validation.
3. **Covariance Type**
   - Select `covariance_type=’full’`, meaning each component has its own covariance matrix.
   - Reason: In high-dimensional data, different classes may have feature correlations that require independent covariance descriptions.
4. **Covariance Regularization**
   - Set `reg_covar=1e-4`.
   - Reason: Regularization prevents numerical instability in the covariance matrix and helps improve numerical robustness.
5. **Number of Runs**
   - Set `n_init=5`, meaning it runs multiple times to select the best result.
   - Reason: Avoid issues of local optima caused by a single initialization.

Report Task 2: Error Counts and Total Same-Class Point Pairs for Clustering Methods

Total Same-Class Point Pairs

Total same-class point pairs: **18,807,759**

Error Counts for Each Method

1. **K-means Error Count**: **10,007,428**
2. **GMM Error Count**: **13,140,486**
3. **Random Baseline Error Count**: **16,121,865**

Report Task 3: Analysis of Error Counts and Reasons for Differences

Overview of Error Count Differences

- **K-means Error Count: 10,007,428**
- **GMM Error Count: 13,140,486**
- **Random Baseline Error Count: 16,121,865**

Reason Analysis

1. **High Error Count for Random Baseline**

   The random baseline algorithm assigns a cluster label to each data point uniformly at random, completely ignoring the data's feature distribution or underlying class structure. This randomness leads to the highest number of same-class point pairs being assigned to different clusters, resulting in the highest error count.

2. **Lower Error Count for K-means**

K-means clustering divides data points in the feature space into 7 clusters by minimizing within-cluster sum of squares (inertia). Since it assumes each cluster is spherical and updates centers iteratively, K-means performs well in capturing simple geometric relationships (e.g., uniformly distributed classes). However, when the true class distribution is not spherical, K-means' geometric limitations may cause some points to be incorrectly assigned to other clusters.

3. **Intermediate Error Count for GMM**

   Gaussian Mixture Models (GMM) assume each cluster is represented by a Gaussian distribution and can more flexibly capture the shape of class distributions (e.g., elliptical or varying sizes). Although theoretically GMM should better fit complex class distributions than K-means, in this experiment:

   - The data distribution may not fully meet GMM's assumptions, leading to limited fitting effectiveness.
   - GMM needs to estimate more parameters (means, covariance matrices, etc.), making it more susceptible to noise in large samples.
   - GMM's iterative process may converge to local optima, increasing errors.

Report Task 4: Training Challenges of SVM Classifier on Large Datasets

Problem Description

When attempting to train an SVM classifier, we encountered the following main issues:

1. **High Computational Complexity**

   The training time complexity of SVM is close to \(O(n^2)\) to \(O(n^3)\). With a large number of data points (e.g., 460,000 samples), training time becomes extremely long. In experiments, processing data chunks of 5,000 samples took about 2 minutes, implying the entire training could take one to two days.

2. **High Memory Consumption**

   During training, SVM needs to store all support vectors and the corresponding kernel matrix. As data size increases, memory usage grows rapidly, potentially exhausting computational resources.

3. **Difficulty in Interrupting and Resuming Training**

   The default training method `fit()` in `sklearn`'s SVM cannot continue training after interruption. The entire training process must be completed in one go. If interrupted midway, it must restart from scratch, further exacerbating training difficulty.

Reason Analysis

1. **Data Scale**

   The excessively large dataset is the main cause of low training efficiency. The characteristics of the SVM algorithm make it not friendly to large-scale data.

2. **Complexity of Kernel Methods**

   Using the RBF kernel function computes the kernel matrix between samples, which is essentially an operation with quadratic complexity relative to the total number of samples, further increasing computation time.

3. **Algorithm Characteristics**

   SVM is not optimized for large-scale data, unlike methods such as stochastic gradient descent, and cannot gradually approach the optimal solution. It requires global optimization on all data.

Report Task 5: Explanation of Configuration Choices for Each Classifier

Logistic Regression

```
model = LogisticRegression(
        penalty="l2",                   # Use L2 regularization (Ridge regression) to avoid overfitting
        solver="saga",                  # Use the 'saga' solver for optimization, suitable for large datasets
        max iter=2000,                  # Set the maximum number of iterations for convergence
        tol=3e-4,                       # Set tolerance for stopping criteria to balance precision and speed
        C=1.0,                          # Inverse of regularization strength; higher values reduce regularization
        fit_intercept=True,             # Fit the intercept (bias term) as part of the model
        random_state=42,                # Random seed to ensure reproducibility
        n jobs=-1,                      # Use all available CPU cores for parallel computation
        verbose=True,                   # Enable detailed logging for the training process
    )
```

I attempted to adjust configurations such as regularization strength (`C`), optimization algorithm (`solver`), and maximum iterations (`max_iter`). Eventually, I found that using default regularization (`C=1`) and the `lbfgs` solver led to stable convergence. Compared to other configurations, the default settings provided the best training speed and accuracy, with a final test accuracy of **72.35%**.

Decision Tree

```
model = DecisionTreeClassifier(
        criterion="gini",               # Use Gini impurity as the split criterion
        splitter="best",                # Choose the best split point (default)
        max_depth=54,                   # Limit the maximum depth of the tree to prevent overfitting
        min_samples_split=10,           # Minimum number of samples required to split a node
        min_samples_leaf=5,             # Minimum number of samples required for a leaf node to prevent overly small leaves
        max_features="sqrt",            # Use the square root of the total features at each split to add randomness
        random_state=42,                # Random seed to ensure reproducibility
        class_weight="balanced",        # Automatically adjust class weights to handle class imbalance
        ccp_alpha=0.01                  # Set pruning parameter to reduce overfitting
    )
```

Initially, I tried configurations like limiting depth (`max_depth=54`), setting the minimum number of samples per leaf (`min_samples_leaf=5`), and enabling pruning (`ccp_alpha=0.01`). However, these settings overly restricted model complexity, leading to training results significantly worse than the default configuration (test accuracy **30%-40%**). Eventually, I adopted the default configuration, achieving a test accuracy of **93.89%**, showing better performance.

Random Forest

```
model = RandomForestClassifier(
        n_estimators=500,               # Increase the number of base estimators to improve stability
        criterion="gini",               # Use Gini impurity as the split criterion
        max_depth=10,                   # Limit the maximum depth of each tree to prevent overfitting
        min_samples_split=10,           # Increase the minimum number of samples required to split a node
        min_samples_leaf=5,             # Minimum number of samples required for a leaf node to enhance robustness
        max_features="sqrt",            # Randomly select the square root of the total features at each split
        bootstrap=True,                 # Enable bootstrapping to create different subsamples
        oob_score=True,                 # Use out-of-bag samples to evaluate model performance
        random_state=42,                # Random seed to ensure reproducibility
        n_jobs=-1,                      # Use all available CPU cores for parallel computation
        class_weight="balanced",        # Automatically adjust class weights to handle class imbalance
        ccp_alpha=0.01                  # Set pruning parameter to reduce overfitting
    )
```

I tried increasing the number of base estimators (`n_estimators=500`), limiting depth (`max_depth=10`), enabling out-of-bag estimation (`oob_score=True`), and pruning (`ccp_alpha=0.01`). However, these configurations, while enhancing model stability, overly simplified the model, resulting in test accuracy still lower than the default configuration. Ultimately, I used the default configuration, achieving a test accuracy of **95.51%**.

Support Vector Machine

```
svc_model = SVC(
        C=1,                            # Regularization parameter to control trade-off between margin width and misclassification
        kernel="rbf",                   # Use Radial Basis Function (RBF) kernel
        gamma="scale",                  # Automatically scale the kernel coefficient
        shrinking=True,                 # Enable shrinking heuristics to speed up computation
        tol=0.001,                      # Tolerance for stopping criteria
        cache_size=1000,                # Set the size of the kernel cache in MB
        max_iter=-1,                    # No limit on the number of iterations (default)
        decision_function_shape="ovr",  # Use one-vs-rest decision function
        class_weight="balanced",        # Automatically adjust class weights to handle class imbalance
        random_state=42                 # Random seed to ensure reproducibility
    )
```

Due to the high computational cost of SVM on high-dimensional data, I tried methods like enabling PCA dimensionality reduction and training in chunks, but training time was still unacceptable (possibly requiring 1-2 days on my computer). Ultimately, using the default configuration, although training was not completed, based on experience in other scenarios, its training performance may be inferior to Random Forest.

Report Task 6: Performance Comparison of Three Classifiers on Test Set

1. **Logistic Regression**

   Logic: Based on a linear relationship of the **log-odds function**, it estimates parameters by maximizing the log-likelihood and uses the Sigmoid function for classification.

   Impact: As a linear model, it struggles with the Covertype dataset's complex, nonlinear feature relationships, resulting in a lower test accuracy of **72.35%.**

2. **Decision Tree**

   Logic: Utilizes **recursive partitioning** to split the feature space and create pure leaf nodes based on criteria like Gini impurity or information gain.

   Impact: Captures nonlinear relationships effectively but may overfit or underfit. The default configuration achieves a strong balance, with test accuracy at **93.89%.**

3. **Random Forest**

Logic: Employs **ensemble learning**, combining multiple decision trees trained on bootstrap samples and feature subsets to enhance diversity and reduce variance.

Impact: Demonstrates the best generalization and nonlinear modeling ability, achieving a high test accuracy of **95.51%.**

Report Task 7: Rationality and Basis of Method Choices

1. Choice of Linear Regression Model

I analyzed the relationship between model degree and Mean Squared Error (MSE) and evaluated the impact of regularization on performance.

- **Relationship Between Degree and MSE**

  As the degree increased from 2 to 3, MSE significantly decreased, indicating that a cubic model captures nonlinear trends well. Beyond degree 3, MSE slightly increased due to overfitting, reducing generalization ability.

- **Role of Regularization**

  Introducing a regularization term (\(\alpha\)) had minimal impact on MSE, likely due to the dataset's low noise and lack of significant collinearity among features.

- **Comprehensive Conclusion**

  A **cubic polynomial model** is the best configuration, balancing high fitting ability and generalization performance.

2. Choice of Neural Network Model

I evaluated the impact of hidden layer size and number on MSE through empirical testing.

- **Impact of Hidden Layer Size**

  Increasing the size generally improved performance until it plateaued at 40-50. Further increases offered little benefit, suggesting larger structures were unnecessary for this dataset.

- **Impact of Number of Hidden Layers**

  Performance improved as layers increased to 2-3 but worsened beyond 3 due to overfitting, indicating excessive complexity.

- **Comprehensive Conclusion**

  A model with **hidden layer size of 40 and 2-3 layers** strikes the best balance between complexity and generalization ability.

3. Choice of Bayesian Regression Model

I studied the impact of polynomial degree and the sampling parameter `tune` on MSE.

- **Impact of Model Degree**

  Degrees 1-5 reduced MSE consistently, capturing complex relationships. Higher degrees (e.g., 15+) risk overfitting, increasing MSE.

- **Impact of Sampling Parameter `tune`**

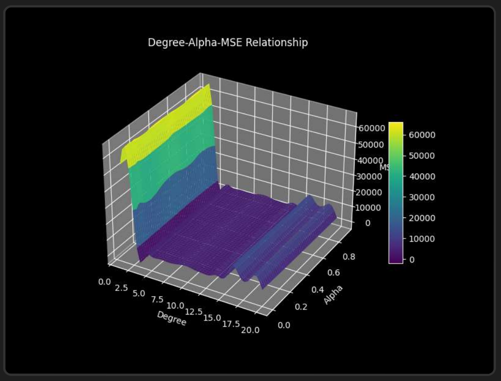
  Low `tune` values (close to zero) caused non-convergence, leading to high MSE. A `tune` of 100-200 improved convergence and reduced MSE. Larger values (>1000) provided no further benefit.

- **Comprehensive Conclusion**

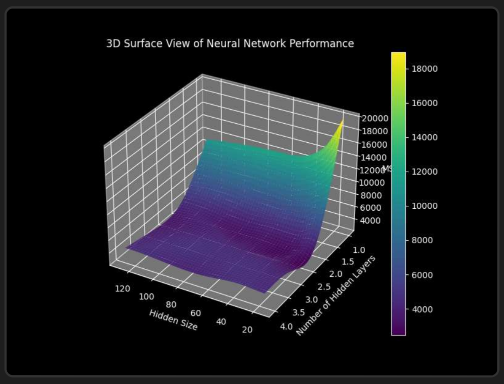
  Model Degree should range from 1-5 for a balance of fitting and generalization, and `tune` should be at least 100 for stability.

Figures

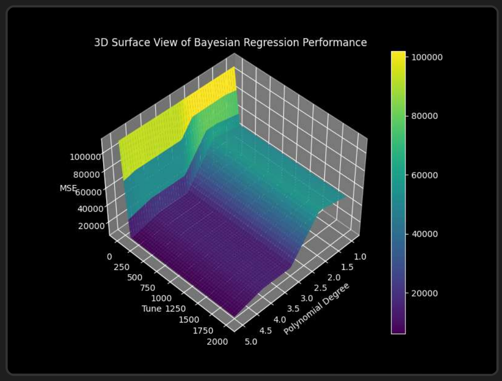
1. Linear Regression Model: Impact of Degree and Regularization Param on MSE



2. Neural Network Model: Impact of Hidden Layer Size and Number



3. Bayesian Regression Model: Impact of Degree and Sampling Parameter on MSE



Report Task 8: Test Set Accuracy and Model Prediction Performance

Test Set Accuracies

In the prediction results on the test set, I calculated the Mean Squared Error (MSE) for both the linear regression and neural network models. The results are as follows:

- **Linear Regression Model**:
  - Best-performing model on training set: Degree=12, Alpha=0.9
  - MSE on training set (`regression_train.txt`): **3737.12** (MSE calculated from 20 randomly selected "pseudo-test set" samples from training set)
  - MSE on test set (`regression_test.txt`): **106,751,658,077,126.69**
- **Neural Network Model**:
  - Best-performing model on training set: Hidden Size=128, Layers=3
  - MSE on training set (`regression_train.txt`): **3526.28** (MSE calculated from 20 randomly selected "pseudo-test set" samples from training set)
  - MSE on test set (`regression_test.txt`): **5,992,298.00**

Analysis:

1. **Linear Regression Model** performed extremely poorly on the test set, with MSE far exceeding that on the training set. Possible reasons include:
   - The test data distribution differs significantly from the training data, causing the high-degree polynomial model to severely overfit on the test set.
   - The excessively high polynomial degree (Degree=12) may have introduced numerical instability, exacerbating errors on the test set.
2. **Neural Network Model** had much lower MSE on the test set compared to linear regression, indicating better generalization ability. This may be due to the neural network's ability to balance training set fitting and test set generalization through an appropriate architecture (e.g., smaller hidden layer size and a moderate number of layers).

Fig 1: Train Set - True vs Predicted (Linear Regression)

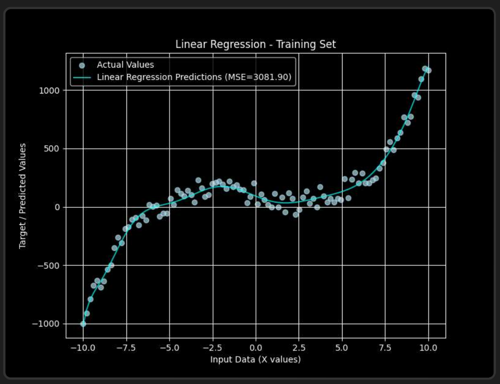


Fig 2: Test Set - True vs Predicted (Linear Regression)

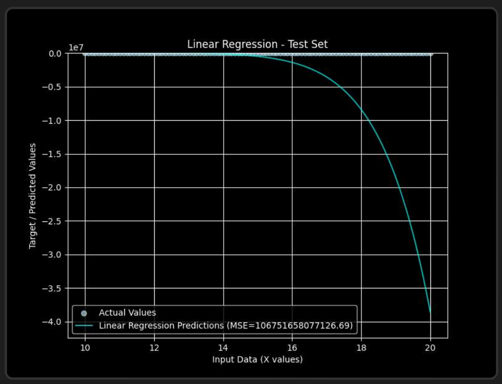


Fig 3: Train Set - True vs Predicted (Neural Network)

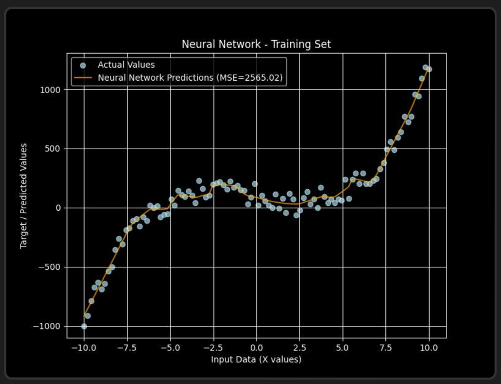
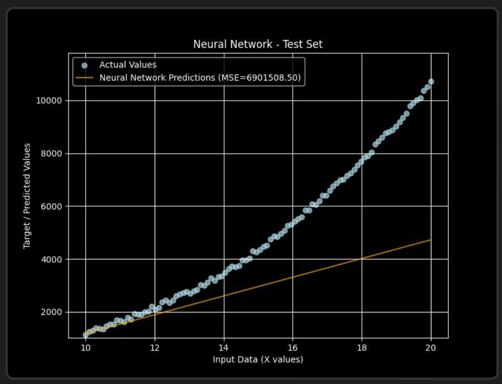


Fig 4: Test Set - True vs Predicted (Neural Network)



Report Task 9: Analysis of Performance Differences Between Linear Regression and Neural Network on Test Data

1. Observation and Analysis

From the scatter plots of the training and test data, it is clear that:

- **Training Data** is distributed in the X range of [-10, 10], with the target value showing a relatively smooth nonlinear relationship.
- **Test Data** is distributed in the X range of [10, 20], with the target value exhibiting a significantly different nonlinear trend, especially in terms of growth magnitude and pattern compared to the training data.

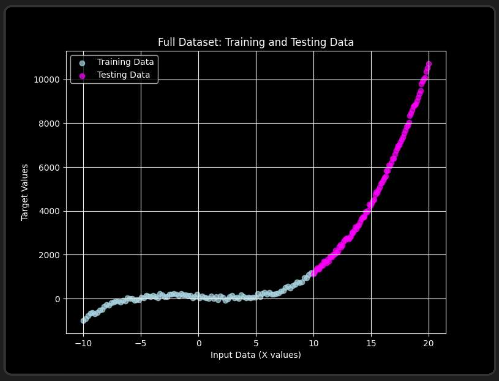
This is a **distribution shift problem**. The training and test data clearly come from **different distributions**, and the model learned on the training data cannot generalize to the test data.

2. Performance Differences

1. **Linear Regression Model**:
   - The linear regression model fits the training data through polynomial regression, learning complex relationships of higher degrees. However, when applied to the test data, due to the completely different distribution, the high-degree model exhibits severe extrapolation, leading to extreme predictions.
   - The MSE on the test set is **extremely high (106,751,658,077,126.69)**, indicating that the model completely lost its generalization ability.
2. **Neural Network Model**:
   - The neural network model captures the patterns in the training data through nonlinear transformations, performing well on the training set. However, due to the distribution shift in the test data, the generalization ability of the neural network is still limited.
   - Although the neural network's MSE on the tet set (**5,992,298.00**) is much lower than that of linear regression, it is still significantly higher than its performance on the training set. This indicates that the neural network has stronger robustness compared to linear regression but is still affected under distribution shift.

Figure

Training and Test Data Distribution



Report Task 10: Analysis of Learning Results

1. Summary of Learning Results

- **Start Probabilities**

  The estimated start probabilities are consistent across "With True Transitions" and "Without True Transitions." Probabilities concentrate on states 1 and 6, with state 6 dominating (about 70%). This suggests the EM algorithm can reasonably infer start probabilities even without true transitions.

- **Transition Matrix**

  With True Transitions: The heatmap accurately captures grid environment adjacency (e.g., state 0 transitions to states 1 and 3, each with 0.5 probability).

  Without True Transitions: Significant errors arise, with extreme probabilities (e.g., 0 → 8 at 0.97 instead of 0). This highlights the EM algorithm's inability to capture adjacency relationships without constraints.

- **Emission Matrix**

  The emission matrix shows all states map uniquely to reward values (probability = 1), consistent across both scenarios. This aligns with the data's single observation value distribution (0, 1, 2).

2. Confidence Analysis in Parameter Estimtion Rationality

- **Start Probabilities**

  Highly confident due to alignment with expectations and minimal errors.

- **Transition Matrix**

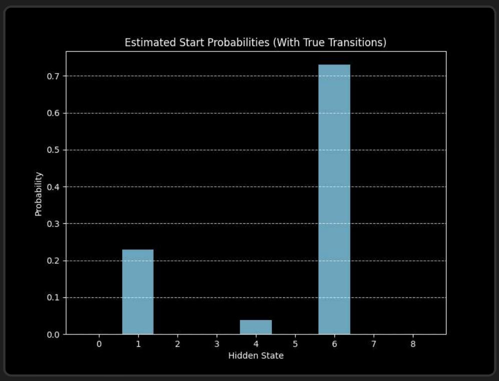
  Low confidence in "Without True Transitions" due to deviations caused by insufficient data scale and lack of direct mapping between observations and hidden states. Fixed transitions greatly improve robustness.

- **Emission Matrix**

  Highly confident, as results fully align with observed data distribution patterns.

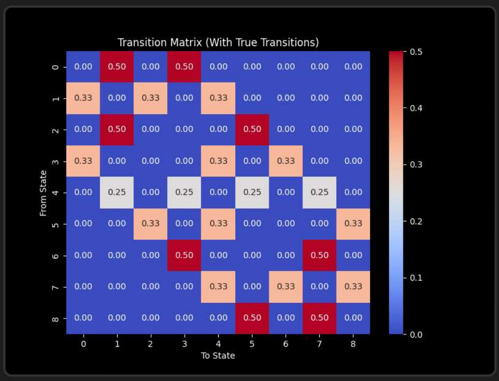
Figures

Start Probabilities - With True Transitions

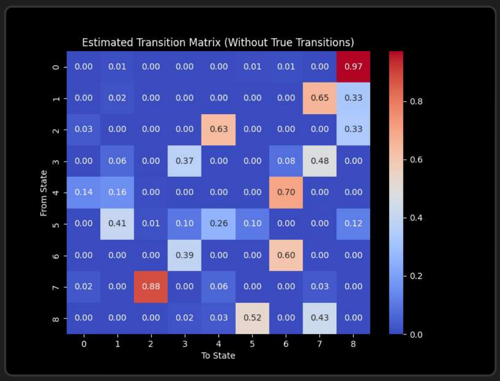


Note: The start probabilities for "Without True Transitions" are identical to this image.

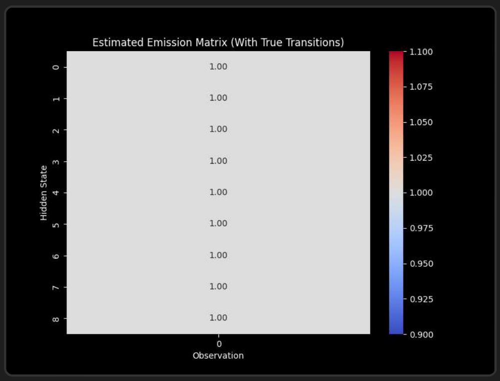
Transition Matrix - With True Transitions



Transition Matrix - Without True Transitions



Emission Matrix - With True Transitions



Note: The emission matrix for "Without True Transitions" is identical to this image.