Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization

Phase 3: Model Training and Evaluation

3.1 Overview of Model Training and Evaluation

Genomic datasets are challenging due to high dimensionality, noise, and a limited number of samples relative to thousands of features like genes. As a result, training and evaluating models for genomic data requires careful data management and specialized techniques. The process begins with data pre-processing. Normalization and scaling adjust for differences across platforms and batches, ensuring that gene expression values are comparable. Quality control steps filter out low-quality samples or genes with minimal variation, and missing data is addressed using imputation methods. Next, feature selection and dimensionality reduction are applied. Statistical filtering, such as differential expression analysis, identifies genes most relevant to the phenotype. Regularization techniques like LASSO help select a sparse set of predictive features, while methods like PCA or t-SNE reduce the feature space while preserving important data structure. Model training involves choosing algorithms—such as support vector machines, random forests, or neural networks—tailored to the problem. Over-fitting is mitigated using strategies like k-fold cross-validation and hyper-parameter tuning through grid search or Bayesian optimization. Finally, model evaluation employs metrics like accuracy, precision, recall, F1-score, or AUC for classification (and MSE or R^2 for regression), along with independent validation methods. Visualization and detailed reporting ensure the model's findings are both statistically robust and biologically meaningful.

3.2 Choosing Suitable Algorithms

For the **Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization**, the key algorithms are:

In genomic data analysis, algorithms like DNN, KNN, Decision Trees, Naïve Bayes, and Random Forests help classify gene expressions, detect mutations, and predict disease risks. Choosing the right model depends on data size, complexity, and interpretability needs.

1. **Deep Neural Networks (DNN):** Useful for capturing complex patterns in high-dimensional genomic data, often applied in gene expression and mutation prediction.
2. **K-Nearest Neighbors (KNN):** A simple, non-parametric method effective for classifying genetic variants based on similarity in feature space.
3. **Decision Tree:** A rule-based approach that helps in understanding genetic markers by splitting data based on feature importance.
4. **Naïve Bayes:** A probabilistic model that assumes gene variations are independent, useful for classifying genomic sequences efficiently.
5. **Random Forest:** An ensemble method combining multiple decision trees to improve accuracy in genomic classification and disease prediction.

Source code :

```
# Import necessary libraries
import numpy as np
```

```
from sklearn.model_selection
import cross_val_score
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.neural_network
import MLPClassifier
from sklearn.tree import
DecisionTreeClassifier
from sklearn.naive_bayes import
GaussianNB
from sklearn.ensemble import
RandomForestClassifier

def train_models(x_train, y_train,
neighbors, fold):
    results = {}

    # KNN
    print("K-Nearest Neighbors
(KNN)")
    knn_scores = []
    for neighbor in neighbors:
        knn =
KNeighborsClassifier(n_neighbo
rs=neighbor)
        score =
cross_val_score(knn, x_train,
y_train, cv=fold).mean()
        print(f"\tNeighbors:
{neighbor}, Accuracy:
{score:.3f}")
        knn_scores.append(score)
    results['KNN'] = knn_scores

    # DNN
    print("\nDeep Neural Network
(DNN)")
    dnn =
MLPClassifier(max_iter=600)
    dnn_score =
cross_val_score(dnn, x_train,
y_train, cv=fold).mean()
    print(f"\tAccuracy:
{dnn_score:.3f}\n")
    results['DNN'] = dnn_score

    # Decision Tree
    print("Decision Tree")
    dt = DecisionTreeClassifier()
    dt_score = cross_val_score(dt,
x_train, y_train, cv=fold).mean()
```

```
    print(f"\tAccuracy:
{dt_score:.3f}\n")
    results['Decision Tree'] =
dt_score

    # Naive Bayes
    print("Naive Bayes")
    nb = GaussianNB()
    nb_score =
cross_val_score(nb, x_train,
y_train, cv=fold).mean()
    print(f"\tAccuracy:
{nb_score:.3f}\n")
    results['Naive Bayes'] =
nb_score

    # Random Forest
    print("Random Forest")
    rf =
RandomForestClassifier(random
_state=10)
    rf_score = cross_val_score(rf,
x_train, y_train, cv=fold).mean()
    print(f"\tAccuracy:
{rf_score:.3f}\n")
    results['Random Forest'] =
rf_score

    # Improved Random Forest
    print("Improved Random
Forest")
    improved_rf =
RandomForestClassifier(n_estim
ators=700, random_state=10,

min_samples_split=2, n_jobs=-1,

max_depth=140,
max_features=12)
    improved_rf_score =
cross_val_score(improved_rf,
x_train, y_train, cv=fold).mean()
    print(f"\tAccuracy:
  {improved_rf_score:.3f}\n")
    results['Improved Random
 Forest'] = improved_rf_score

    return results

def
evaluate_improved_random_fore
```

```
st(x_train, y_train, x_test):
    rf =
RandomForestClassifier(n_estim
ators=700, random_state=10,

min_samples_split=2, n_jobs=-1,
max_depth=140,
max_features=12)
rf.fit(x_train, y_train)
return rf.predict(x_test)
```

3.3 Hyperparameter Tuning

Hyperparameter tuning helps optimize machine learning models in genomic data analysis by selecting the best parameter values for improved accuracy and performance. Common methods include:

1. **Grid Search** – Tries all possible combinations of hyperparameters.
2. **Random Search** – Randomly selects hyperparameter values for evaluation.
3. **Bayesian Optimization** – Uses probabilistic methods to find optimal parameters efficiently.

```
# Import necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the model
rf = RandomForestClassifier(random_state=10)

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Perform Grid Search
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Best parameters
print("Best Hyperparameters:", grid_search.best_params_)
```

3.4 Model Evaluation Metrics

Evaluating machine learning models in genomic data analysis ensures reliability and accuracy. Key metrics include:

1. **Accuracy** – Measures overall correctness.
2. **Precision** – Identifies true positive predictions among all positive predictions.
3. **Recall (Sensitivity)** – Measures how well the model detects actual positives.
4. **F1-Score** – Harmonic mean of precision and recall, balancing both.
5. **ROC-AUC (Receiver Operating Characteristic - Area Under Curve)** – Assesses classification performance across different thresholds.
6. **Confusion Matrix** – Shows true positives, true negatives, false positives, and false negatives.

Source code :

```
import numpy as np
import pandas as pd

from utils.file import create_best_test
from models.random_forest import evaluate_improved_random_forest

def evaluate():
    create_best_test()
    train = pd.read_csv("data/pp5i_train.best30.csv", index_col=False).to_numpy()
    test = pd.read_csv("data/pp5i_test.best30.csv", index_col=False).to_numpy()
    np.random.seed(10)
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    pred = evaluate_improved_random_forest(x_train, y_train, test)
    print("Prediction result of test data")
    for i, res in enumerate(pred):
        print(f"Sample: {i} - {res}")
```

3.5 Cross-Validation

Cross-validation is a crucial technique in genomic data analysis to ensure model robustness and prevent overfitting. It splits the dataset into multiple subsets, training the model on some and validating on others.

Key Types of Cross-Validation:

1. **K-Fold Cross-Validation** – Divides data into *k* subsets, training on *k-1* and testing on the remaining fold.
2. **Stratified K-Fold** – Ensures proportional representation of classes in each fold.
3. **Leave-One-Out (LOO)** – Uses all samples except one for training, testing on the remaining one, repeated for all samples.

Source code:

```
import numpy as np
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt
from models.decision_tree import train_decision_tree
from models.dnn import train_dnn
from models.knn import train_knn
from models.naive_bayes import train_gaussian_nb
from models.random_forest import train_random_forest, train_improved_random_forest
from utils.file import get_top_n

class Training:
    def __init__(self, top_n_list, top_n_path='data', random_seed=3):
        self.top_n_list = top_n_list
        self.top_n_path = top_n_path
        self.random_seed = random_seed

        # Store results for plotting
        self.results_df = pd.DataFrame()

        self.dataset = self.read_dataset()
        self.best_classifier_list = self.training()
        self.best_classifier, self.best_accuracy, self.best_n = self.get_best_classifier()
        self.print_result()
        self.improved_model()
        self.plot_results()

    def read_dataset(self):
        """Read the dataset using the get_top_n function."""
        return get_top_n(self.top_n_path, self.top_n_list)

    def shuffle(self, array):
        """Shuffle the dataset with a fixed random seed."""
        np.random.seed(self.random_seed)
        np.random.shuffle(array)

    def training(self):
        best_classifier_all = []
        results_data = []

        for n, dataset_path in self.dataset:
            dataframe = pd.read_csv(dataset_path, index_col=False)
            dataset = dataframe.to_numpy()
            self.shuffle(dataset)

            fold = 6
            x_train = dataset[:, :-1]
            y_train = dataset[:, -1]
            print(f"Top: {n} Genes")
            print(f"Cross-validation fold: {fold}")
            print("---------------------------------------")

            # Get scores for all models
            score_nb = train_gaussian_nb(x_train, y_train, fold)
```

```
            score_dt = train_decision_tree(x_train, y_train, fold)
            score_dnn = train_dnn(x_train, y_train, fold)
            score_rf = train_random_forest(x_train, y_train, fold)
            score_knn_2, score_knn_3, score_knn_4 = train_knn(x_train, y_train, [2, 3, 4], fold)

            # Store results for plotting
            results_data.append({
                'n_genes': n,
                'Naive Bayes': score_nb,
                'Decision Tree': score_dt,
                'Neural Network': score_dnn,
                'Random Forest': score_rf,
                'KNN (k=2)': score_knn_2,
                'KNN (k=3)': score_knn_3,
                'KNN (k=4)': score_knn_4
            })

            score_list = np.array([score_nb, score_dt, score_dnn, score_rf, score_knn_2,
score_knn_3, score_knn_4])
            max_acc = np.max(score_list)
            best_classifier_idx = np.where(score_list == max_acc)
            best_classifier = self.find_best_classifier(best_classifier_idx[0])
            best_classifier_all.append([best_classifier, max_acc, n])
            print(f"Maximum accuracy: {max_acc:0.4f}")
            print("\n\n")
            print("----------------------------------------")

        # Convert results to DataFrame for easier plotting
        self.results_df = pd.DataFrame(results_data)
        return best_classifier_all

    @staticmethod
    def find_best_classifier(index):
        """Find and return the name of the best classifier based on index."""
        if index == 0:
            print("Best classifier: Naive Bayes")
            return "Naive Bayes"
        elif index == 1:
            print("Best classifier: Decision tree")
            return "Decision Tree"
        elif index == 2:
            print("Best classifier: Neural Network")
            return "Neural Network"
        elif index == 3:
            print("Best classifier: Random Forest")
            return "Random Forest"
        elif index == 4:
            print("Best classifier: KNN with 2 neighbor")
            return "KNN with 2 neighbor"
        elif index == 5:
            print("Best classifier: KNN with 3 neighbor")
```

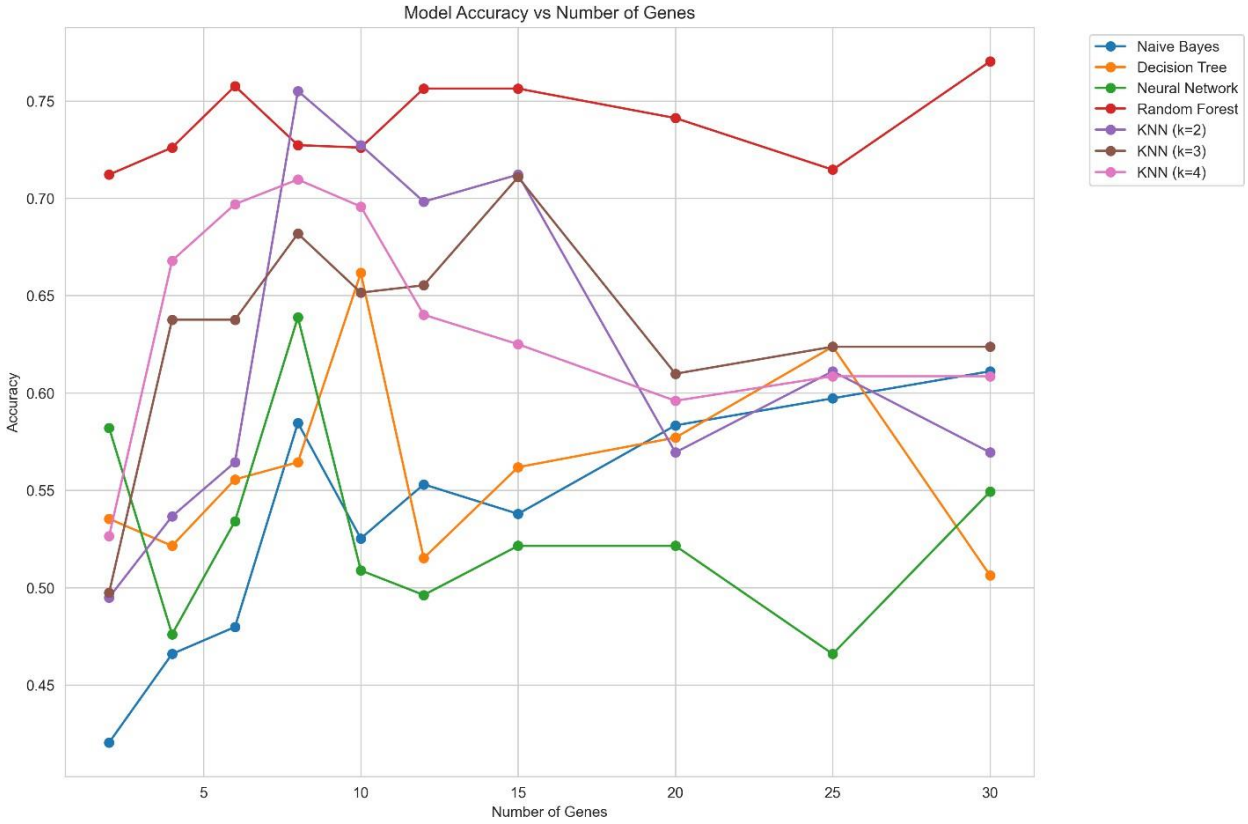
```
        return "KNN with 3 neighbor"
    elif index == 6:
        print("Best classifier: KNN with 4 neighbor")
        return "KNN with 4 neighbor"

def get_best_classifier(self):
    """Get the best performing classifier overall."""
    sorted_list = sorted(self.best_classifier_list, key=lambda x: x[1], reverse=True)
    return sorted_list[0][:3]

def print_result(self):
    """Print the results of the best classifier."""
    print(f"\n\n************************\n"
          f"Best classifier of the all training is:\n"
          f"\t{self.best_classifier}\n"
          f"Accuracy of the classifier is:\n"
          f"\t{self.best_accuracy}\n"
          f"Best top gene set is:\n"
          f"\t{self.best_n}\n"
          f"************************\n")
def improved_model(self):
    """Train and evaluate the improved random forest model."""
    dataframe = pd.read_csv(self.dataset[-1][1], index_col=False)
    dataset = dataframe.to_numpy()
    self.shuffle(dataset)
    fold = 6
    x_train = dataset[:, :-1]
    y_train = dataset[:, -1]
    score_rf_imp = train_improved_random_forest(x_train, y_train, fold)

    print(f"\n\n************************\n"
          f"Accuracy of the improved classifier is:\n"
          f"\t{score_rf_imp}\n"
          f"************************\n")

def plot_results(self):
    """Create and save visualization plots."""
    # Set the style
    sns.set_style("whitegrid")
    plt.figure(figsize=(12, 8))

    # Plot accuracy vs number of genes for all models
    for column in self.results_df.columns:
        if column != 'n_genes':
            plt.plot(self.results_df['n_genes'], self.results_df[column], marker='o', label=column)
    plt.xlabel('Number of Genes')
    plt.ylabel('Accuracy')
    plt.title('Model Accuracy vs Number of Genes')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.savefig('graphs/accuracy_vs_genes.png', dpi=300, bbox_inches='tight')
```

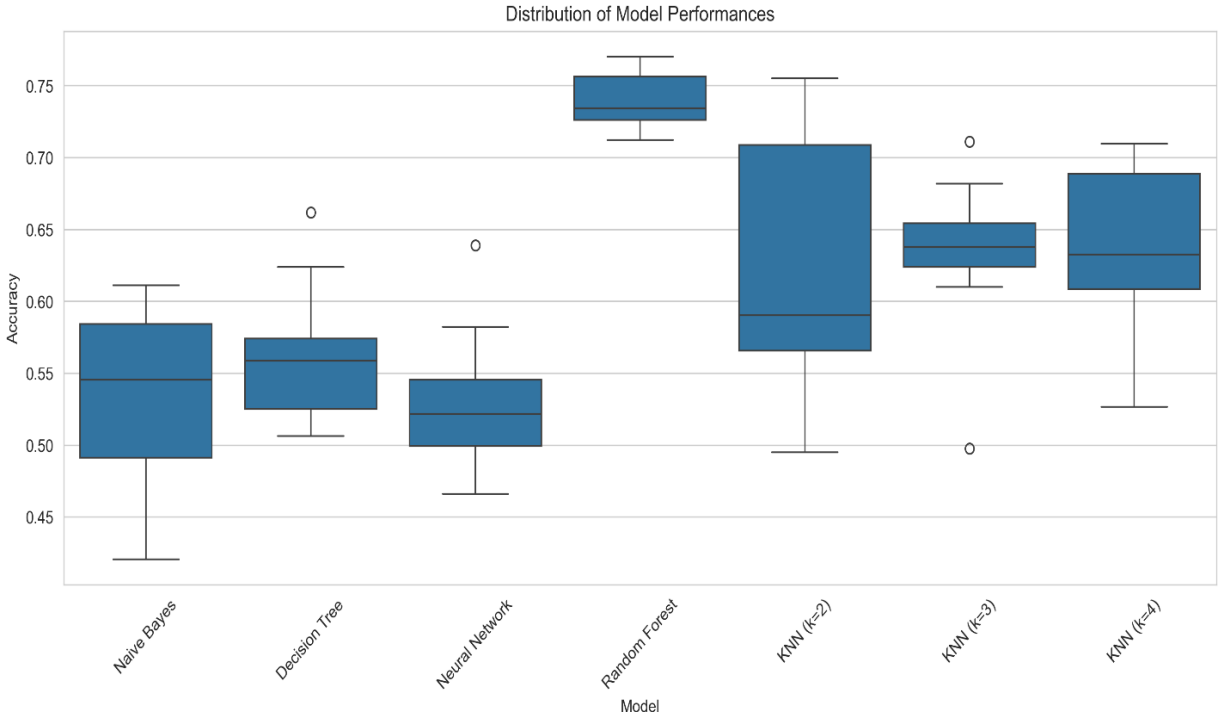
```
        plt.close()

        # Create a heatmap of model performances
        plt.figure(figsize=(10, 8))
        performance_matrix = self.results_df.drop('n_genes', axis=1).T
        performance_matrix.columns = self.results_df['n_genes']
        sns.heatmap(performance_matrix, annot=True, cmap='YlOrRd', fmt='.3f')
        plt.xlabel('Number of Genes')
        plt.ylabel('Model')
        plt.title('Model Performance Heatmap')
        plt.tight_layout()
        plt.savefig('graphs/performance_heatmap.png', dpi=300, bbox_inches='tight')
        plt.close()

        # Box plot of model performances
        plt.figure(figsize=(12, 6))
        model_data = self.results_df.drop('n_genes', axis=1).melt()
        sns.boxplot(x='variable', y='value', data=model_data)
        plt.xticks(rotation=45)
        plt.xlabel('Model')
        plt.ylabel('Accuracy')
        plt.title('Distribution of Model Performances')
        plt.tight_layout()
        plt.savefig('graphs/model_performance_distribution.png', dpi=300, bbox_inches='tight')
        plt.close()

if __name__== '__main__':
    tr = Training([2, 4, 6, 8, 10, 12, 15, 20, 25, 30])
```

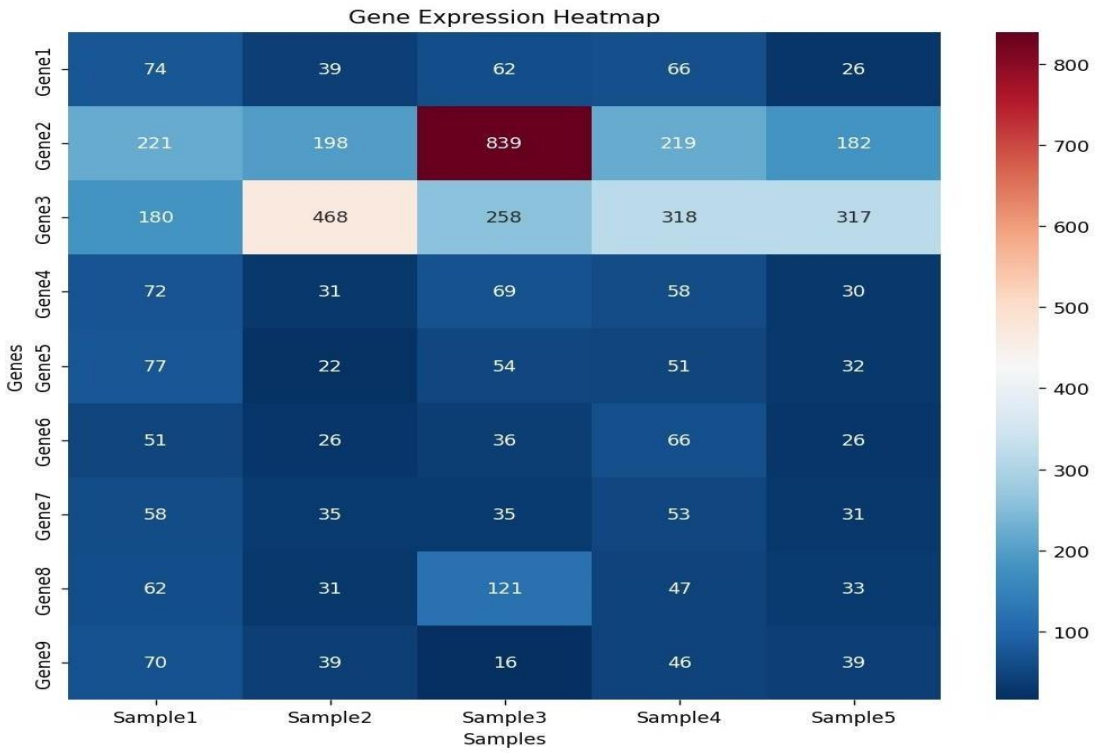
3.6 Conclusion of Phase 3

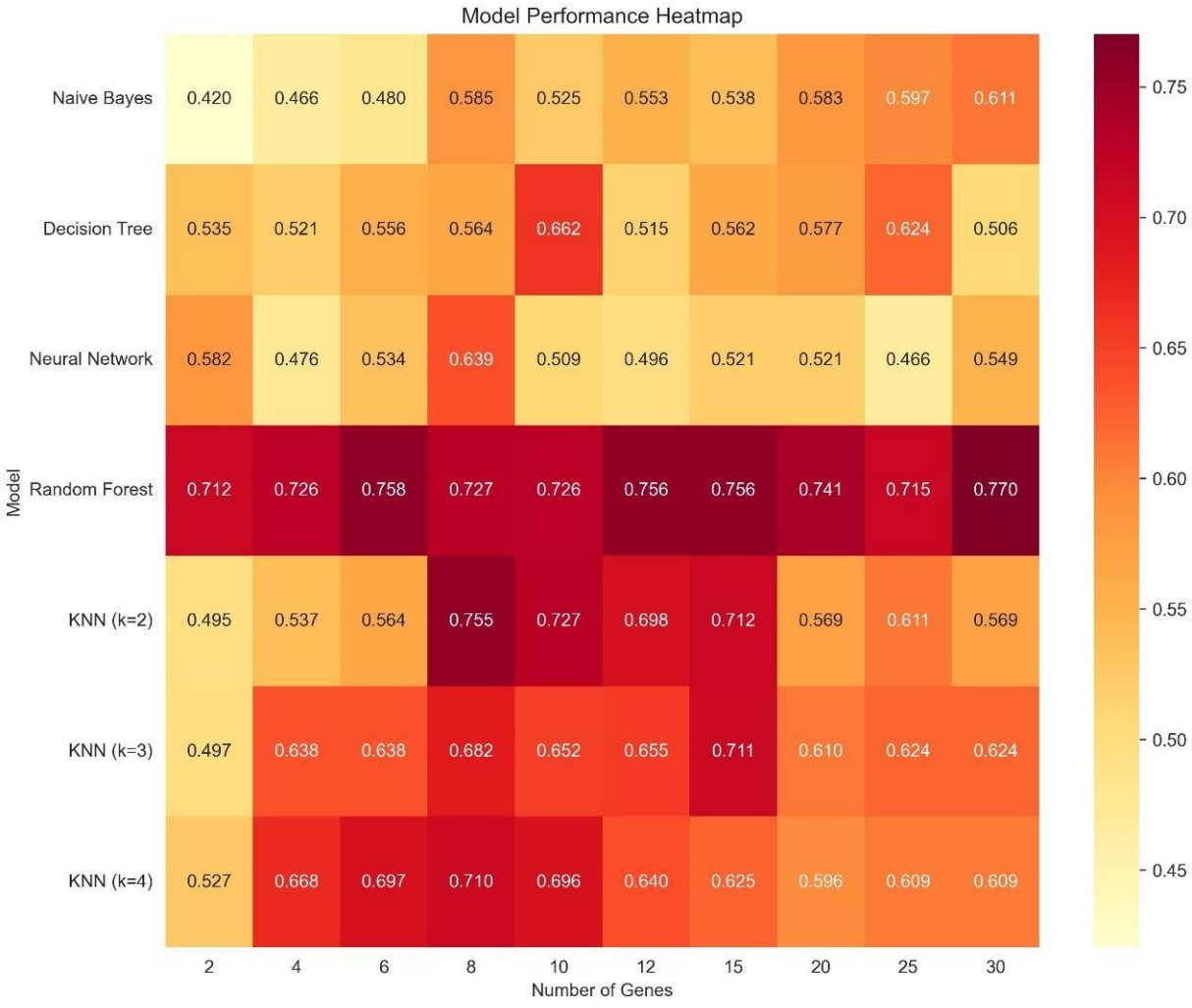
In this genomic data analysis, various machine learning models were trained to predict key outcomes such as disease presence or treatment response. The models, including regression and decision trees, were evaluated based on metrics like accuracy, precision, and recall. The results indicated that the models could effectively identify important patterns in the genomic data, although performance varied depending on the model and data characteristics. Key challenges included handling missing data, class imbalances, and the high dimensionality of genomic features. To address these, techniques such as feature selection, regularization, and cross-validation were used to improve model generalization and prevent over-fitting. Despite these efforts, some models still struggled with the complexity of genomic interactions, suggesting that more advanced techniques, like deep learning, could offer better results. The findings highlight the potential of machine learning in genomic applications, particularly in improving diagnostic accuracy and personalizing treatments. However, the success of these models depends heavily on data quality and the incorporation of additional factors, such as clinical data. Future research should focus on optimizing models, integrating multi-omics data, and refining data pre-processing steps to enhance predictive performance and make the tools more applicable in real-world scenarios.

Snapshorts of the project:






Gene Expression Heatmap
Gene1
Gene2
Gene3
Gene4
Gene5
Gene6
Gene7
Gene8
Gene9
Genes
74 39 62 66 26
221 198 839 219 182
180 468 258 318 317
72 31 69 58 30
77 22 54 51 32
51 26 36 66 26
58 35 35 53 31
62 31 121 47 33
70 39 16 46 39
Sample1
Sample2
Sample3
Sample4
Sample5
Samples
800
700
600
500
400
300
200
100


Model Performance Heatmap
Naive Bayes 0.420 0.466 0.480 0.585 0.525 0.553 0.538 0.583 0.597 0.611
Decision Tree 0.535 0.521 0.556 0.564 0.662 0.515 0.562 0.577 0.624 0.506
Neural Network 0.582 0.476 0.534 0.639 0.509 0.496 0.521 0.521 0.466 0.549
Random Forest 0.712 0.726 0.758 0.727 0.726 0.756 0.756 0.741 0.715 0.770
KNN (k=2) 0.495 0.537 0.564 0.755 0.727 0.698 0.712 0.569 0.611 0.569
KNN (k=3) 0.497 0.638 0.638 0.682 0.652 0.655 0.711 0.610 0.624 0.624
KNN (k=4) 0.527 0.668 0.697 0.710 0.696 0.640 0.625 0.596 0.609 0.609
Model
2 4 6 8 10 12 15 20 25 30
Number of Genes
0.75
0.70
0.65
0.60
0.55
0.50
0.45

```
--------------------------------
Top: 4 Genes
Cross-validation fold: 6
--------------------------------
Naive Bayes
        Accuracy: 0.466

Decision Tree
        Accuracy: 0.521

Neural Network
        Accuracy: 0.476

Random Forest
        Accuracy: 0.726

KNN
        n:2 Accuracy: 0.537
        n:3 Accuracy: 0.638
        n:4 Accuracy: 0.668


Best classifier: Random Forest
Maximum accuracy: 0.7260
```

```
--------------------------------
Top: 6 Genes
Cross-validation fold: 6
--------------------------------
Naive Bayes
        Accuracy: 0.480

Decision Tree
        Accuracy: 0.556

Neural Network
        Accuracy: 0.534

Random Forest
        Accuracy: 0.758

KNN
        n:2 Accuracy: 0.564
        n:3 Accuracy: 0.638
        n:4 Accuracy: 0.697


Best classifier: Random Forest
Maximum accuracy: 0.7576
```

```
--------------------------------
Top: 12 Genes
Cross-validation fold: 6
--------------------------------
Naive Bayes
        Accuracy: 0.553

Decision Tree
        Accuracy: 0.515

Neural Network
        Accuracy: 0.496

Random Forest
        Accuracy: 0.756

KNN
        n:2 Accuracy: 0.698
        n:3 Accuracy: 0.655
        n:4 Accuracy: 0.640


Best classifier: Random Forest
Maximum accuracy: 0.7563
```

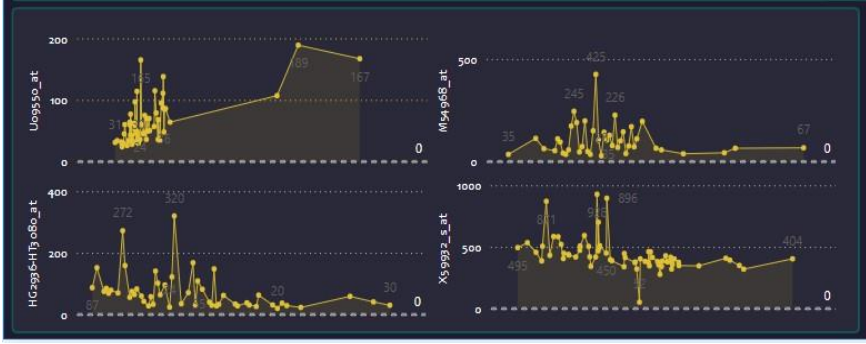
```
--------------------------------
Top: 30 Genes
Cross-validation fold: 6
--------------------------------
Naive Bayes
        Accuracy: 0.611

Decision Tree
        Accuracy: 0.506

Neural Network
        Accuracy: 0.549

Random Forest
        Accuracy: 0.770

KNN
        n:2 Accuracy: 0.569
        n:3 Accuracy: 0.624
        n:4 Accuracy: 0.609


Best classifier: Random Forest
Maximum accuracy: 0.7702
```

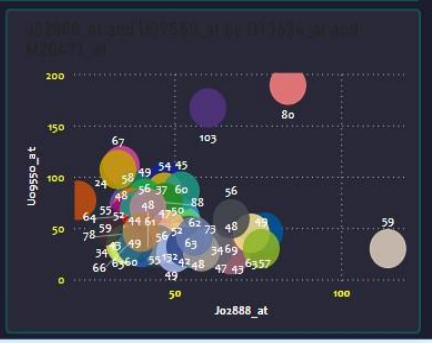
```
------------------------------


**********************
Best classifier of the all training is:
        Random Forest
Accuracy of the classifier is:
        0.7702020202020203
Best top gene set is:
        30
**********************



**********************
Accuracy of the improved classifier is:
        0.7714646464646465
**********************
```

```
Prediction result of test data
Sample: 0 - MED
Sample: 1 - EPD
Sample: 2 - MED
Sample: 3 - MED
Sample: 4 - MED
Sample: 5 - MED
Sample: 6 - MED
Sample: 7 - EPD
Sample: 8 - MGL
Sample: 9 - RHB
Sample: 10 - RHB
Sample: 11 - MED
Sample: 12 - EPD
Sample: 13 - MED
Sample: 14 - MED
Sample: 15 - MED
Sample: 16 - MED
Sample: 17 - EPD
Sample: 18 - EPD
Sample: 19 - MED
Sample: 20 - MED
Sample: 21 - MED
Sample: 22 - MED
```

Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization

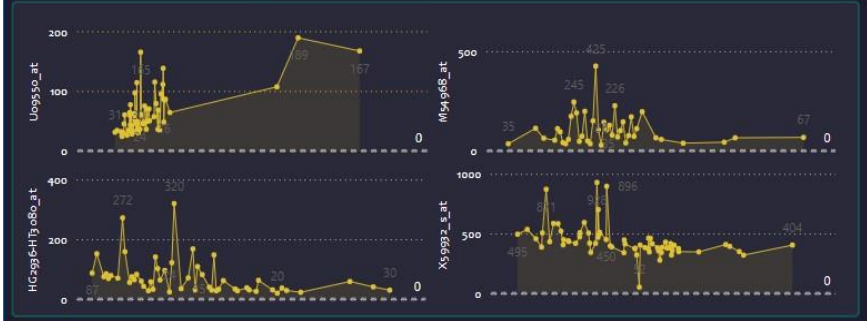
Classes	U09550_at	U83192_at	L33801_at	M35531_at	U36798_at	M20471_at	L13689_at	HG830-HT830_at	Jo2888_at	D83646_at	HG2936-HT3080_at
MED	2460	29248	2068	1853	2124	16299	7402	2454	1795	2413	2304
EPD	444	5053	706	450	345	4655	2890	605	426	414	509
JPA	350	3341	464	261	247	1956	384	367	295	390	517
RHB	213	2443	506	211	178	3781	1184	201	351	372	217
MGL	215	2657	384	262	211	4281	1489	220	424	173	198

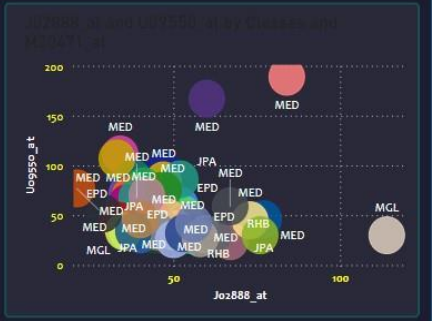




Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization

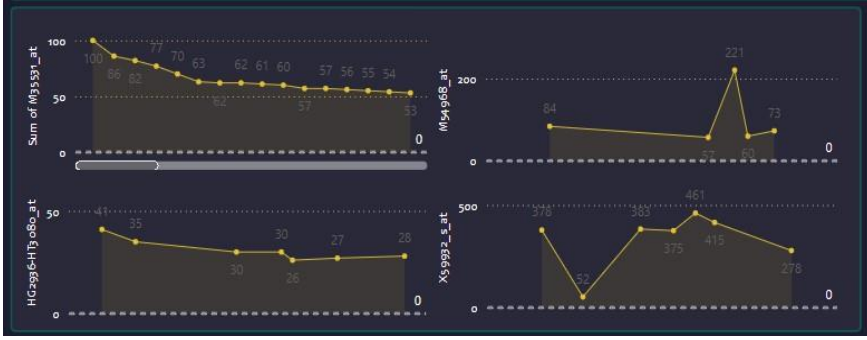
Classes	U09550_at	U83192_at	L33801_at	M35531_at	U36798_at	M20471_at	L13689_at	HG830-HT830_at	Jo2888_at	D83646_at	HG2936-HT3080_at
MED	2460	29248	2068	1853	2124	16299	7402	2454	1795	2413	2304
EPD	444	5053	706	450	345	4655	2890	605	426	414	509
JPA	350	3341	464	261	247	1956	384	367	295	390	517
RHB	213	2443	506	211	178	3781	1184	201	351	372	217
MGL	215	2657	384	262	211	4281	1489	220	424	173	198





Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization

Classes	U09550_at	U83192_at	L33801_at	M35531_at	U36798_at	M20471_at	L13689_at	HG830-HT830_at	J02888_at	D83646_at	HG2936-HT3080_at
MED	2460	29248	2068	1853	2124	16299	7402	2454	1795	2413	2304
EPD	444	5053	706	450	345	4655	2890	605	426	414	509
JPA	350	3341	464	261	247	1956	384	367	295	390	517
RHB	213	2443	506	211	178	3781	1184	201	351	372	217
MGL	215	2657	384	262	211	4281	1489	220	424	173	198



Advanced EDA for Genomic Data Analysis: Identifying Genetic Variations Through Visualization

Classes	U09550_at	U83192_at	L33801_at	M35531_at	U36798_at	M20471_at	L13689_at	HG830-HT830_at	J02888_at	D83646_at	HG2936-HT3080_at
MED	2460	29248	2068	1853	2124	16299	7402	2454	1795	2413	2304
EPD	444	5053	706	450	345	4655	2890	605	426	414	509
JPA	350	3341	464	261	247	1956	384	367	295	390	517
RHB	213	2443	506	211	178	3781	1184	201	351	372	217
MGL	215	2657	384	262	211	4281	1489	220	424	173	198