Dipartimento di Informatica dell'Università di Bologna Corso di Laurea Magistrale in Informatica – Curriculum A

Final Report of Data Analytics

MovieLens Data Analytics

Project exam

Tirocinante: Zhiguang Li

Matricola: 0001029938

Professor.: Prof. Marco Di Felice

Professor: Prof. Giuseppe Lisanti

Index

1. Introduction

The objective of this project is to predict the average ratings of movies based on their features using the MovieLens dataset, a recommendation system containing over 60,000 movies. Operated by GroupLens Research at the University of Minnesota, MovieLens serves as the primary dataset for analysis. The project employs a variety of methods, including traditional techniques, neural network-based approaches, and the deep learning framework TabNet. The research process follows stages seen in the course:

- Data acquisition
- Data visualization
- Data preprocessing
- Modeling
- Performance evaluation

To facilitate coding, data acquisition and visualization are combined into one code file, while data preprocessing, modeling, and performance evaluation are separated into traditional machine learning methods (modelTML), neural networks (modelDNN), and TabNet (modelTabNet), respectively. Preprocessing and modeling are prerequisites for model training, with performance evaluation metrics outputted for each method. The models explored include:

- Traditional Machine Learning:
  - Linear Regression
  - Random Forest Regression
  - Ridge Regression
  - KNN Regression
  - Lasso Regression
  - Support Vector Regression
- Neural Networks
- TabNet

2. Data Acquisition

Initially, we downloaded the relevant dataset from a website. The dataset used is MovieLens, a movie recommendation system featuring over 60,000 movies along with their ratings and tags. Additionally, each movie is associated with a genome that identifies a feature and its relevance.

- genome-scores: Each line in this file represents a movie's relevance, containing the movie ID, tag ID, and the tag's relevance. It comprises 15,584,448 rows.
- genome-tags: Contains tags used to evaluate movies in the genome-scores.csv file. Each line represents a tag, including the tag ID and its description, totaling 1,128 rows.
- movies: Includes movie information, such as their titles, release years, and genres. Each line represents a movie, containing the movie ID, title with release year, and genres, with 62,423 rows in total.
- ratings: Contains user ratings for movies. Each line represents a rating, including the user ID, movie ID, rating, and rating date, with a total of 25,000,095 rows.

For analysis, not all available files were used; only the movies, genome-scores, genome-tags, and ratings CSV files were utilized.

Due to the data being distributed across multiple files, a decision was made to create a merged file, resulting in a new dataset where each movie is associated with the relevancy of each tag related to it. For each movie, the average rating from user reviews was calculated, yielding a continuous value.

3.Data Visualization

Data visualization aims to clearly and effectively communicate data through charts, aiding in efficient data exploration and the discovery of potential relationships. This section introduces several different charts.

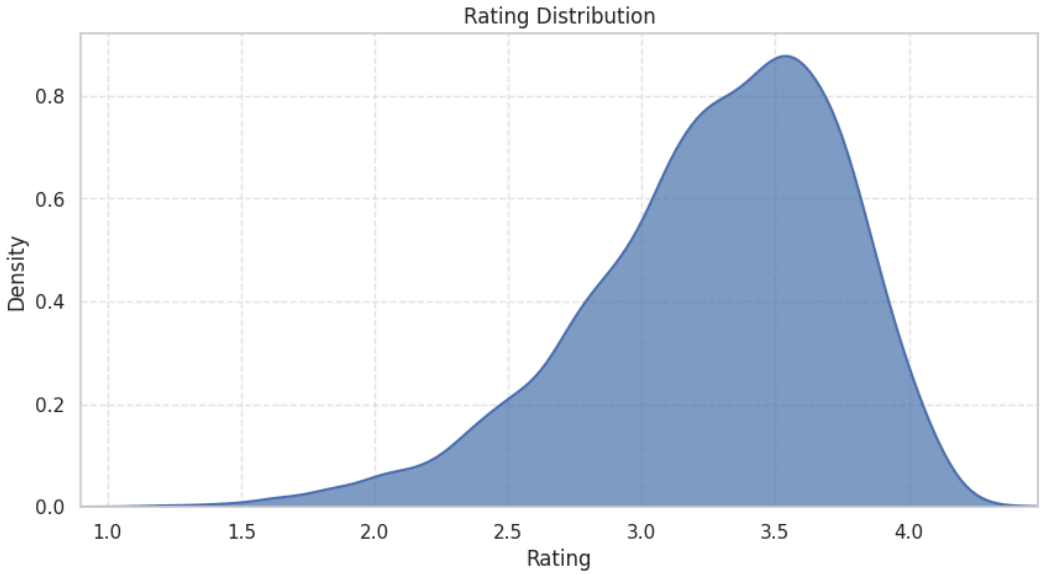

Figure 1: Distribution of Ratings

It shows the distribution of ratings within the dataset. The chart reveals a higher density area around the rating value of 3.5.

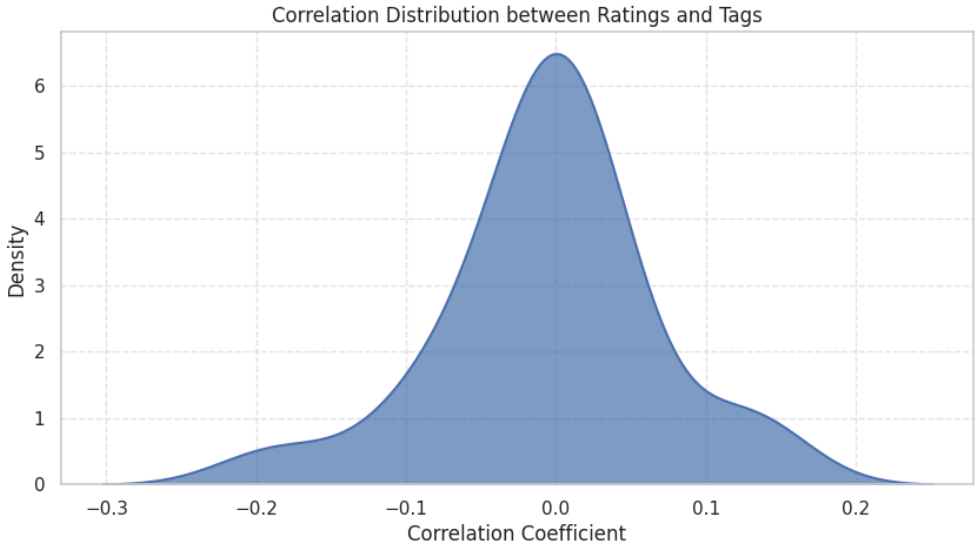

Figure 2: Correlation Distribution Between Tags and Ratings

It illustrates the correlation between tags and ratings. Some tags have a higher correlation with ratings than others.

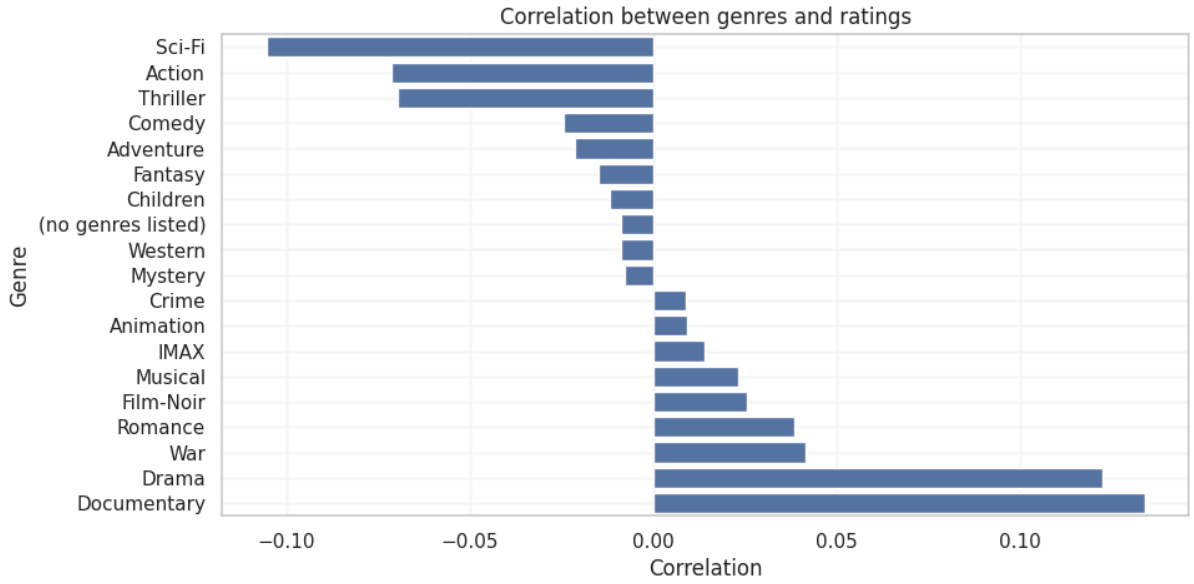

Figure 3: Correlation Between Movie Genres and Ratings

It displays the correlation between movie genres and ratings. Documentary, Drama, and War genres show a higher positive correlation with ratings, whereas Horror, Sci-Fi, and Action genres exhibit a higher negative correlation.

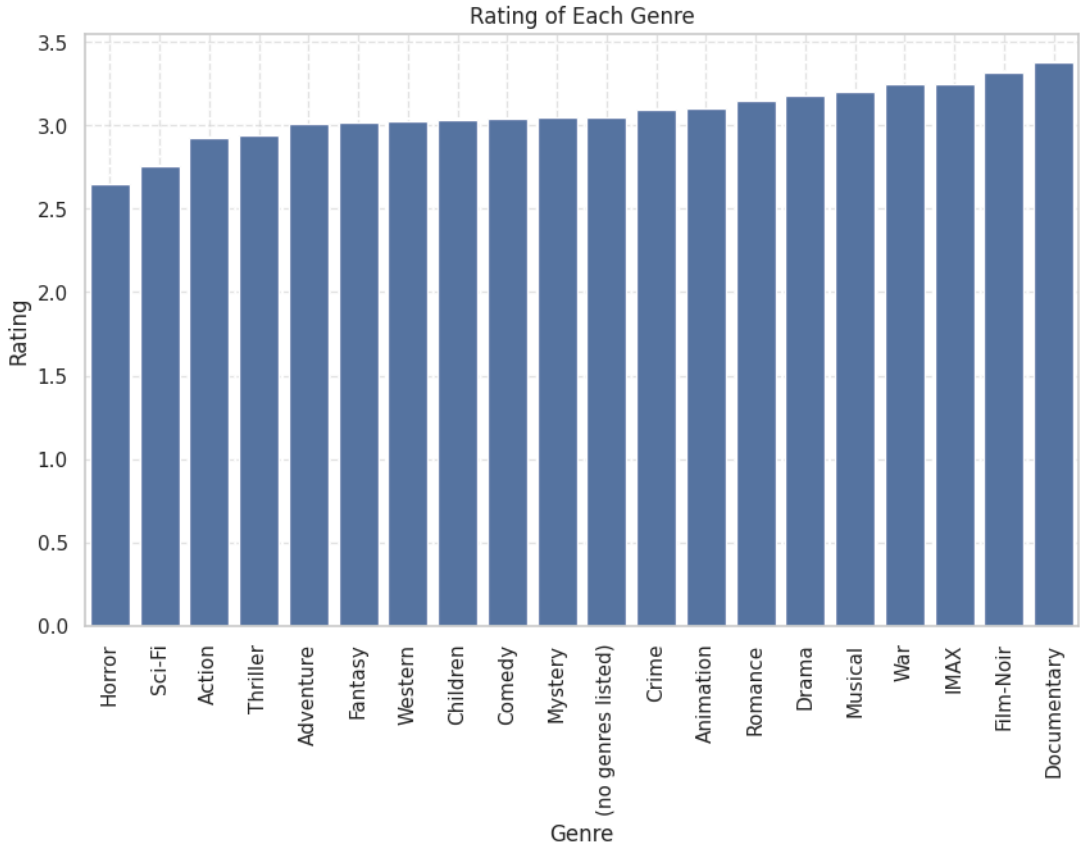

Figure 4: Average Ratings by Genre

It presents the average ratings for each movie genre.

Finally, a chart utilizing a dimensionality reduction algorithm known as t-Distributed Stochastic Neighbor Embedding (t-SNE) was created, as seen in Figure 5, to represent movies in a two-dimensional space based on ratings.

The goal of this algorithm is to find a low-dimensional representation of the data where similar points in the original dataset are also close in the low-dimensional space. This allows for the visualization of data in two or three dimensions while preserving the relationships between the original data points.

The t-SNE process mainly consists of two steps: calculating a similarity matrix between points in the original dataset using a distance function, and then converting this similarity matrix into a probability distribution.

The algorithm then attempts to represent the data points in a low-dimensional space while maintaining the probability distribution of similarities between points. This process involves iteratively minimizing the Kullback-Leibler divergence between the probability distribution of the original dataset and that of the new low-dimensional space.

The final outcome of the t-SNE algorithm is a two-dimensional map of the data. In this specific case, the algorithm is applied to movie ratings to represent them in a two-dimensional space, allowing for the visualization of the distribution of ratings.

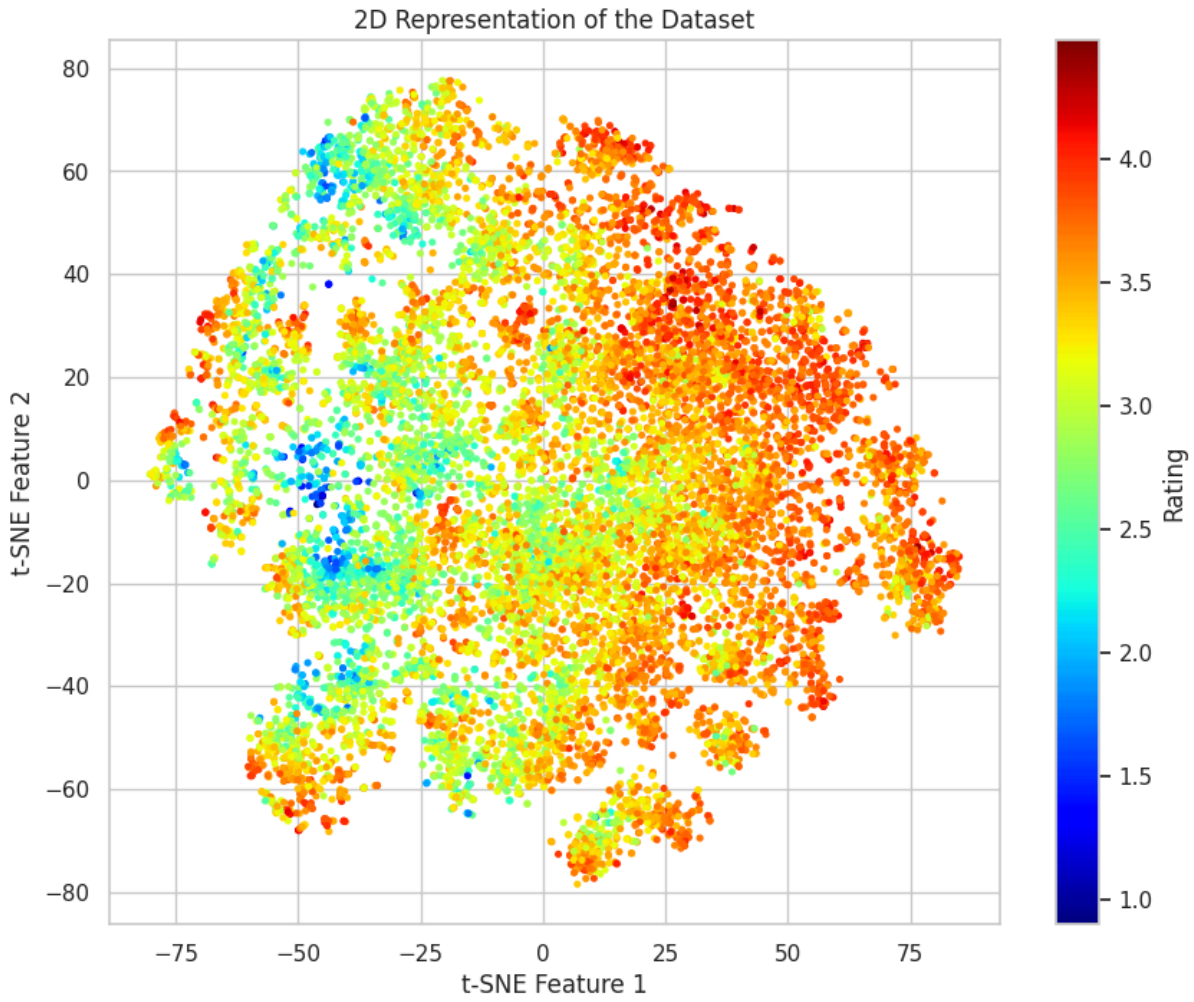

Figure 5: Two-Dimensional Scatter Plot Representation of Data

The observed chart reveals a gradual variation in movie ratings distribution. Movies with the lowest ratings are represented in blue, while those with the highest ratings are shown in red. Additionally, most movie ratings fall between 3 to 3.5, indicating denser areas due to a larger number of examples. This aligns with the earlier observations regarding the distribution of ratings.

4. Data Preprocessing

Preprocessing refers to all operations on raw data to prepare it for further processing.

This stage is crucial as collected data often contains noise, missing values, outliers, and imperfections that can negatively impact analysis results. Through this phase, data is cleaned and transformed to better fit analysis needs. Initially, the dataset is split into training, validation, and testing sets (70-10-20).

The dataset contains neither null nor duplicate values, and scaling is not necessary as values are already between 0 and 1. Principal Component Analysis (PCA), a dimensionality reduction technique aimed at representing dataset variability with fewer principal components, was applied.

This technique transforms a set of correlated variables into a set of linearly uncorrelated new variables, called principal components. By projecting original data onto a new coordinate system with the first axis being the calculated principal component, data can be represented in a lower dimension without significant information loss.

PCA was applied to the training set with dependent variables represented by each movie's average rating, retaining components that accounted for 95% of the data's variance. Performance will be tested with and without PCA to observe how performance varies with changes in dataset dimensions.

5. Modeling

In the modeling phase, various supervised machine learning techniques were employed to predict the average ratings of movies, including traditional methods, neural network-based deep learning techniques, and TabNet (a deep learning model designed for tabular data).
This analysis could be viewed either as a regression or a classification problem. For this study, it was treated as a regression problem, given that the target variable is a continuous value.

5.1 Machine Learning Techniques

Supervised machine learning techniques involve training models on a labeled dataset, where both the input data and the corresponding output results are known.

5.1.1 Linear Regression

Linear regression is a statistical method used to model and analyze the linear relationships between variables. It predicts a dependent variable based on one or more independent variables, aiming to find the best fitting line (in simple linear regression) or hyperplane (in multiple linear regression) that minimizes the sum of squared errors between the actual observations and predictions. In this specific case, the dependent variable is represented by the movie's average rating, with independent variables being the movie features that could influence predictions, such as ratings for different tags.

Core code implementation:

```
lin_regr = linear_model.LinearRegression()
lin_regr.fit(X_train, y_train)
y_pred = lin_regr.predict(X_test)
```

The data for X_train, y_train comes from the dataset.csv file that we preprocessed and stored, which has 13816 rows x 1129 columns of data and combines basic information such as movies, scores, tags, ratings, etc. The data for X_train and y_train are stored in the dataset.csv file, which has 13,816 rows x 1129 columns of data.

5.1.2 Random Forest Regressor

The Random Forest Regressor is an ensemble learning algorithm based on decision trees, designed for solving regression problems. It improves prediction accuracy and robustness by building multiple decision trees and averaging or voting on their predictions. Each tree is trained on a random subset of the dataset, reducing overfitting to specific data and enhancing generalizability. The Random Forest Regressor is particularly effective for large datasets with high-dimensional features and complex data structures.

Core code implementation:

```
param_grid = {
    "n_estimators": [10, 15, 20, 25, 30],
    "criterion": ["squared_error", "friedman_mse"]
}
rf = RandomForestRegressor()

try:
    print(X_train.shape)
    print(y_train.shape)
            grid_search    =    GridSearchCV(rf,    param_grid,    cv=5,
scoring='neg_mean_squared_error', return_train_score=True)
    grid_search.fit(X_train, y_train)
    best_hyper = grid_search.best_params_
    print("Best hyperpara meters: ", best_hyper)
except Exception as e:
    print("An error occurred:", e)
        ……
rf           =           RandomForestRegressor(criterion='friedman_mse',
n_estimators=30).fit(X_train, y_train)
```

The use of Scikit-learn's GridSearchCV aimed to find the best hyperparameter combination for RandomForestRegressor involves defining a parameter grid, `param_grid`, with different values for `n_estimators` (the number of trees) and `criterion` (evaluation criteria). A RandomForestRegressor instance is created, followed by using GridSearchCV with cross-validation (`cv=5`) to identify the best parameters under the `neg_mean_squared_error` scoring criterion. Parameters explored include

- n_estimators = [10, 15, 20, 25, 30]: Specifies the number of trees in the forest.
- criterion = ['squared_error', 'friedman_mse']: Metrics to measure the quality of splits.
- "squared_error": Corresponds to mean squared error, focusing on variance reduction and minimizing L2 loss through the average values at terminal nodes.

- "friedman_mse": Employs Friedman's mean squared error improvement score to assess the effectiveness of potential splits.

5.1.3 Ridge Regression

Ridge Regression is a linear model for regression problems, incorporating L2 regularization to the least squares method, reducing model sensitivity to data noise and overfitting risk. The regularization strength, controlled by alpha or lambda, significantly impacts model performance. Ridge Regression is particularly effective when the number of features exceeds the number of training samples or when features are highly correlated.

Core code implementation:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

param_grid = {"alpha": [0.0001, 0.001, 0.1, 0.5, 1, 5, 10, 20]}
ridge = Ridge()

try:
          grid_search    =    GridSearchCV(ridge,    param_grid,    cv=5,
scoring='neg_mean_squared_error', return_train_score=True)
   grid_search.fit(X_train, y_train)
   best_hyper = grid_search.best_params_
   print("Best hyperparameters: ", best_hyper)
except Exception as e:
   print("An error occurred during GridSearchCV fitting:", e)
      ……
   optimized_ridge = Ridge(best_hyper)
   optimized_ridge.fit(X_train, y_train)
   y_pred = optimized_ridge.predict(X_test)
```

Using Scikit-learn's GridSearchCV, the optimal regularization strength (`alpha`) for the Ridge Regression model is identified. `param_grid` outlines a range of possible alpha values to control regularization strength. GridSearchCV runs Ridge Regression for each alpha value, assessing model performance with 5-fold cross-validation and mean squared error as the scoring criterion. After grid search completion, the code outputs the best-performing hyperparameter set.

5.1.4 K-Nearest Neighbors Regression

K-Nearest Neighbors Regression (KNN Regression) is a non-parametric regression method based on neighbor voting. It predicts the output for a point by finding the K nearest neighbors and using their target values. The output is the average or weighted average of the target values of these neighbors.

Core code implementation:

```
from sklearn.neighbors import KNeighborsRegressor

param_grid = {
   'n_neighbors': [7, 8, 9, 10, 15, 20],
   'weights': ['uniform', 'distance']
}
knn = KNeighborsRegressor()

try:
            grid_search    =    GridSearchCV(knn,    param_grid,    cv=5,
scoring='neg_mean_squared_error', return_train_score=True)
   grid_search.fit(X_train, y_train)
   best_hyper = grid_search.best_params_
   print("Best hyperparameters: ", best_hyper)
except Exception as e:
   print("An error occurred during GridSearchCV fitting:", e)
      ……
knn = KNeighborsRegressor(best_hyper)
knn.fit(X_train, y_train)
y_pred= knn.predict(X_test)
```

Scikit-learn's GridSearchCV was utilized to find the best hyperparameters for the K-Nearest Neighbors Regression (KNN Regression) model. This involved setting up a `param_grid` with different values for `n_neighbors` (the number of neighbors) and `weights` (the weight function). A `KNeighborsRegressor` instance was created, followed by cross-validation (cv=5) to determine the best parameters under the neg_mean_squared_error scoring criterion. After completing the grid search, the code outputs the optimal hyperparameter set. Using these parameters, a new KNN regression model was created and trained on the training set before making predictions on the test set.

In the optimization for K-Nearest Neighbors Regression (KNN Regression) using Scikit-learn's GridSearchCV, the parameters considered were:

- n_neighbors = [7, 8, 9, 10, 15, 20]: This parameter specifies the number of neighbors to consider.

- weights = ['uniform', 'distance']: This parameter specifies the type of weighting to apply to predictions, where:

- uniform: All points in each neighborhood are weighted equally.

- distance: Points are weighted by the inverse of their distance, meaning closer neighbors have a greater influence on the prediction.

5.1.5 Lasso Regression

Lasso Regression is an extension of linear regression that incorporates an L1 regularization term into the loss function. This regularization helps with feature selection by shrinking some coefficients to zero, thereby reducing the number of variables in the model, simplifying it, and helping to prevent overfitting. Lasso Regression is particularly suited for scenarios with multicollinearity or when the number of dimensions exceeds the number of samples. The strength of the regularization is controlled by a hyperparameter, which significantly impacts the model's performance and complexity.

Core code implementation:

```
param_grid = {
                'alpha':[1e-1, 1e-2, 1e-3, 1e-4, 1e-5]
             }
lasso = Lasso()
grid_search = GridSearchCV(lasso, param_grid, cv=5,
scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(X_train,y_train)
best_hyper= grid_search.best_params_
print("Best hyper: ", best_hyper)
      ……
lasso = Lasso(best_hyper)
lasso.fit(X_train, y_train)
y_pred= lasso.predict(X_test)
```

Scikit-learn's GridSearchCV was used to find the optimal regularization parameter alpha for Lasso Regression. A parameter grid, param_grid, contained a range of candidate alpha values. GridSearchCV evaluated each alpha's mean squared error across 5-fold cross-validation to determine the best-performing alpha. Upon identifying the optimal parameter, a new Lasso instance was created with these parameters and fitted to the training

data. Finally, predictions were made on the test data using the trained model, and the best hyperparameter was reported.

- alpha: A regularization parameter that controls the strength of L1 regularization within the model. In Lasso Regression, higher alpha values enhance coefficient constraints, leading to more coefficients becoming zero and thus producing a more simplified (sparse) model. Conversely, lower alpha values reduce the penalty on coefficients, making the model closer to traditional linear regression.

5.1.6 Support Vector Regression SVR

Support Vector Regression (SVR) is a regression method based on Support Vector Machine (SVM) principles, aiming to find a hyperplane in high-dimensional space for best fitting data points. It introduces a margin of tolerance for errors while accurately predicting continuous target values, particularly excelling in datasets with high-dimensional features. Its strengths include handling non-linear relationships and being less sensitive to outliers, with the capability to effectively process various types of datasets through appropriate kernel function selection.

核心代码实现:

```
# define params of grid
param_grid = {
   'kernel': ['linear', 'poly', 'rbf'],
   'epsilon': [0.001, 0.01, 0.1, 1]
}
# SVR
svr = SVR()

# execute the GridsearchCV
grid_search = GridSearchCV(svr, param_grid, cv=5,
scoring='neg_mean_squared_error', return_train_score=True)
grid_search.fit(X_train, y_train)

#obtain the best params and print it
best_hyper = grid_search.best_params_
print("Best hyperparameters:", best_hyper)
      ……
svr = pd.DataFrame([grid_search.best_params_])
      ……
```

```
svr = SVR(best_hyper)
svr.fit(X_train, y_train)
y_pred= svr.predict(X_test)
```

The script utilizes Scikit-learn's GridSearchCV to identify optimal hyperparameters for Support Vector Regression (SVR), examining various kernel types and epsilon values within a defined parameter grid. Performance is gauged through 5-fold cross-validation, employing negative mean squared error as the metric—where lower scores indicate better model performance. The best hyperparameter combination and the corresponding score, converted to Root Mean Squared Error (RMSE), are outputted. A new SVR model with these optimal parameters is then fitted to the training data and predictions are made on the test dataset.

- kernel = ['linear', 'poly', 'rbf']: Specifies the type of kernel to be used.
- epsilon = [0.001, 0.01, 0.1, 1]: Defines the epsilon-tube, which is the width of the error margin allowed around the regression line.

5.2 Neural Network

Neural networks are algorithms that mimic the human brain to identify complex patterns and relationships, widely used in machine learning and AI. They consist of multiple layers, each with neurons connected by weights that transmit information. During training, neural networks adjust these weights to learn data features, enabling tasks like classification, regression, or feature detection.

The network in this project includes variable layers, starting with a fully connected layer that maps input data to a new feature space, followed by several hidden layers, and ends with an output layer.

Core code implementation:

```
import torch.nn as nn

def get_model(input_size, hidden_size=32, dropout_prob=0.2):
   model = [
        nn.Linear(input_size, hidden_size),
        nn.ReLU(),
        nn.Dropout(dropout_prob),
        nn.Linear(hidden_size, hidden_size // 2),
        nn.ReLU(),
        nn.Dropout(dropout_prob),
        nn.Linear(hidden_size // 2, 1)
   ]
   return nn.Sequential(*model)
      ……
from itertools import product

# hyper params
hidden_size = [128, 256]
dropout_prob = [0.3]
dept = [3, 4]
epochs = 200
batch_size = [16, 32]
learning_rate = [0.001]

params = product(hidden_size, dropout_prob, dept, batch_size,
learning_rate)
combinations =
len(hidden_size)*len(dropout_prob)*len(dept)*len(batch_size)*len(learni
ng_rate)
```

```
print("Number of combinations: ", combinations)
```

The code encompasses two main segments:

1. Neural Network Model Definition:

- The `get_model` function constructs a simple neural network with three linear layers interspersed with ReLU activation layers and a Dropout layer to mitigate overfitting.
- Variables include `input_size` for the input layer node count, `hidden_size` for the first hidden layer's nodes, with the second hidden layer having half as many.
- `dropout_prob` controls the Dropout layer's neuron "shut-off" rate during training, with the output layer consisting of a single node for regression tasks.

2. Hyperparameter Combination Iteration with itertools:

- Hyperparameters such as layer size, dropout probability, depth, batch size, and learning rate are predefined.
- `itertools.product` is utilized to iterate through all parameter combinations during training.
- The `combinations` variable totals the number of parameter combinations.

Parameters_name and means:

- hidden_size: Size of the hidden layers.
- dropout_prob: Probability of randomly disabling some neurons during network training.
- depth: Number of hidden layers in the network.
- batch_size = [16, 32]: Size of the input batches.
- learning_rate = [0.001]: Determines the step size at each iteration towards minimizing the loss function.

5.3 TabNet

TabNet is a deep learning model designed for tabular data, blending neural network and decision tree features for efficient feature selection and processing, enhancing interpretability and performance. It employs an attention mechanism to focus on predictive features while ignoring irrelevant data, making it ideal for complex tasks requiring consideration of many features. TabNet processes raw input through a sequential attention mechanism across decision steps for improved interpretability and learning, focusing on subsets of input features with multiple decision blocks.

Core code implementation:

```
def get_model(n_d, n_a, n_steps, n_independent):
    model = TabNetRegressor(
        n_d=n_d,
        n_a=n_a,
        n_steps=n_steps,
        n_independent=n_independent  # 确保参数名称正确
    )
    return model
       ……
#fit model
    model.fit(
        X_train=X_train,
        y_train=y_train,
        eval_set=[(X_val, y_val)],
        eval_name=['mse'],
        patience=10,
        batch_size=b,
        virtual_batch_size=128
    )

    # evaluate model
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
```

Using the TabNet model for regression tasks involves creating a TabNetRegressor instance with the `get_model` function, where `n_d`, `n_a`, `n_steps`, and `n_independent` are configuration parameters affecting the model's internal structure, such as the number of decision steps and the dimensions of the feature transformers. The model is trained using the `model.fit` method with training and validation sets, employing an early stopping strategy

with the `patience` parameter to prevent overfitting. Performance on the test set is evaluated using `model.predict`, calculating MSE and R2 scores. Parameters include:

- batch_size: The number of samples per batch.
- n_d: Size of the prediction layer.
- n_a: Size of the output space for the attention transformer.
- n_steps: Number of steps in the architecture.
- n_independent: Number of independent GLU layers in each GLU block.

6. Performance Evaluation

6.1 Traditional Supervised Machine Learning Techniques

Two key metrics for assessing model performance are Mean Squared Error (MSE) and the Coefficient of Determination (R^2):

- Mean Squared Error (MSE): MSE is the average of the squares of the differences between actual and predicted values. It's a standard metric for measuring model prediction accuracy. Lower MSE indicates smaller prediction errors and better model performance. MSE is highly sensitive to outliers, meaning outliers in data can significantly impact MSE values.
- Coefficient of Determination (R^2): R^2 measures the degree of correlation between model predictions and actual observations. R^2 values range from 0 to 1, where a value close to 1 indicates the model explains the variation of the target variable well. Higher R^2 suggests stronger explanatory power of the model, though it doesn't necessarily imply prediction accuracy. R^2 is insensitive to model complexity and may be artificially high due to overfitting.

These metrics are often used together to comprehensively assess a model's accuracy and fitting ability.

	MSE（PAC / NO PAC）	R2（PAC / NO PAC）
Liner Regression	0.0064 / 0.0054	0.9709 / 0.9754
Random Forest Regression	0.0373 / 0.0125	0.8315 / 0.9433
Ridge Regression	**0.0063 / 0.0053**	**0.9711 / 0.9759**
KNN Regression	0.0402 / 0.8182	0.0405 / 0.8172
Lasso Regression	0.0064 / 0.0054	0.9710 / 0.9755
Support Vector Regression	0.0064 / 0.0054	0.9708 / 0.9754

Table1. The MSE and R2 of ML methods

6.2 Neural Networks

In the context of neural networks, key performance metrics include Accuracy, Loss, Precision, Recall, and F1 Score. Accuracy measures the proportion of correct predictions, offering a quick performance assessment. Loss indicates the discrepancy between predictions and actual values, representing model error. Precision and Recall focus on the quality and scope of positive predictions, suitable for imbalanced datasets. The F1 Score balances Precision and Recall. Together, these metrics provide a comprehensive view of model performance, aiding optimization.

This case emphasizes Loss and R2 Score, considering different configurations for hidden_size, depth, and batch_size. Such as:

```
hidden_size = [128, 256]
dropout_prob = [0.3]
dept = [3, 4]
epochs = 200
batch_size = [16, 32]
learning_rate = [0.001]
```

Ultimately, the best configuration within the current training model was determined based on the value of Loss. Note: This configuration may not apply to all models and all different hardware setups. It is a conclusion drawn only on the experimental equipment, so it should be considered as a reference only.

Best loss: 0.006481497548520565（0.00648）

R2 score: 0.9707600847749389 （0.97076）

Best config:{ 'hidden_size': 256,
'dropout_prob': 0.3,
'dept': 4,
'batch_size': 16,
'learning_rate': 0.001
}

The results indicate the model has a very low loss (0.00648) and a high R2 score (0.97076), demonstrating highly accurate predictions on the dataset, with loss nearing zero and a high degree of fit. A high R2 score means the model explains nearly 97% of the variance in the

target variable, typically indicating excellent model performance and its ability to capture patterns and relationships within the data effectively.

6.3 TabNet

When evaluating TabNet, important metrics similar to those used in traditional machine learning and deep learning models include Mean Squared Error (MSE), R^2 score, Accuracy, Precision, and Recall. These metrics collectively assess the model's predictive performance, error levels, and balanced performance in classification tasks. In this project, we focused on studying MSE and R^2 score to determine the optimal TabNet model configuration. Based on the latest MSE and R^2 scores, we obtained the best TabNet model setup.

Best MSE: 0.007878176810709618

R2 Score: 0.9644592608757693

Best Model: TabNetRegressor(

n_d=16,

n_a=16,

n_steps=3,

n_independent=2

…..

)

7 Conclusion

This project allowed us to test different types of machine learning methods, where, aside from KNN and Random Forest, almost all traditional models performed well. Thus, we can conclude that the problem can be effectively solved using linear models. Non-deep techniques are both quick and simple to implement, as they are readily available in relevant libraries.

In our case, PCA was also used to reduce noise, thereby losing all readable attribute information. Neural networks and TabNet slightly improved performance over non-deep models but required longer training times. Deep neural networks are more complex due to the many parameters involved and are mostly opaque. The distinction lies in the number of neurons or layers of neurons, rather than "tree depth" or "function coefficients". Nonetheless, the execution speed post-modeling and the flexibility of application (e.g., for images, videos, or audio) make it a very valuable approach.

Lastly, TabNet, leveraging revolutionary transformer technology applied to unprocessed tabular data (one of the most common data types), emerges as one of the most promising methods for solving such problems.