Nan_Mudhalvan

Phase 4: *Development Part 2 - Feature Engineering, Model Training, and Evaluation*

Team Details

Team name: Dynamite coders

Problem Statement: Predicting IMDb Scores

Dataset Link:

https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores

Introduction:

In this phase, we continue to build the IMDb score prediction model by refining feature engineering, training the machine learning model, and evaluating its performance. These steps are essential for enhancing the model's predictive accuracy and ensuring that it meets the project's objectives.

Step 1: Feature Engineering (Continued)

- **Feature Transformation:** Apply appropriate transformations to features as needed. This might include logarithmic transformations for skewed data or scaling features to standardized ranges.
- **Feature Interaction:** Explore interactions between features and create new variables that capture these relationships. For example,

you might create interaction terms between movie genre and runtime.

- **Dimensionality Reduction:** Consider dimensionality reduction techniques such as Principal Component Analysis (PCA) to reduce the number of features while retaining important information.

Step 2: Model Training (Continued)

- **Algorithm Selection:** Confirm the choice of the machine learning algorithm based on its performance on the training and validation data.

```
lin_scores = cross_val_score(lin_reg, netflix_prepared, netflix_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

- **Hyperparameter Tuning:** Fine-tune the model's hyperparameters to optimize its performance. Techniques like grid search or random search can be used to find the best hyperparameter combinations.
- **Cross-Validation:** Utilize cross-validation to validate the model's performance and assess its generalizability.

```
lin_scores = cross_val_score(lin_reg, netflix_prepared, netflix_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error', return_train_score=True)
```

```
grid_search.fit(netflix_prepared, netflix_labels)
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
pd.DataFrame(grid_search.cv_results_)
```

Step 3: Evaluation

- **Model Evaluation:** Use appropriate evaluation metrics to assess the model's performance on the validation data. Common metrics for regression tasks include Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared (R^2) score.

```
X_test = strat_test_set.drop("Runtime", axis=1)
y_test = strat_test_set["Runtime"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
final_rmse

from sklearn.metrics import mean_squared_error

netflix_predictions = lin_reg.predict(netflix_prepared)
lin_mse = mean_squared_error(netflix_labels, netflix_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

- **Model Interpretability:** Ensure that the model's predictions are interpretable and explainable. This is especially important if the model is intended for user consumption.
- **Overfitting and Underfitting:** Check for signs of overfitting (model learning noise in the data) or underfitting (model being too simple). Adjust model complexity as needed.

Step 4: Model Fine-Tuning

- **Iterative Development:** Review the results of model evaluation and make iterative adjustments to features, hyperparameters, or even the choice of algorithm if necessary.
- **Regularization:** Consider the use of regularization techniques (e.g., L1 or L2 regularization) to prevent overfitting.
- **Ensemble Methods:** Experiment with ensemble methods like Random Forest, Gradient Boosting, or stacking to combine the strengths of multiple models.

Step 5: Model Training Documentation

- **Documentation:** Create comprehensive documentation that records all aspects of model training and evaluation, including hyperparameter choices and cross-validation results. This documentation is crucial for transparency and replication.

Step 6: Final Evaluation

- **Testing Data:** Evaluate the model's performance on a separate testing dataset that it has not encountered during training or validation. This testing dataset is a proxy for real-world performance.

```
from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
mean = squared_errors.mean()
m = len(squared_errors)

np.sqrt(stats.t.interval(confidence, m - 1,
                         loc=np.mean(squared_errors),
                         scale=stats.sem(squared_errors)))

tscore = stats.t.ppf((1 + confidence) / 2, df=m - 1)
tmargin = tscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - tmargin), np.sqrt(mean + tmargin)
zscore = stats.norm.ppf((1 + confidence) / 2)
zmargin = zscore * squared_errors.std(ddof=1) / np.sqrt(m)
np.sqrt(mean - zmargin), np.sqrt(mean + zmargin)
```

```
array([ 5.0708514 , 10.65998939])

(5.070851401917997, 10.659989389344949)

(5.160475520565385, 10.616892206428679)
```

- **User Testing:** If applicable, involve potential users in testing the model and gather feedback on its predictions and usability.

Link for the notebook:
https://colab.research.google.com/drive/16F_qPTs5wxYh_poFeqyEYgtmHJMy5c-y?usp=sharing

Conclusion:

In this part of Phase 4, we continue to refine the IMDb score prediction model. The process includes advanced feature engineering, fine-tuning the model, and rigorous evaluation. The ultimate goal is to build a robust and accurate model that can effectively predict IMDb scores, helping users discover highly rated films that match their preferences.