Titanic Survival Prediction with Ensemble Learning

1. Introduction

This document details the implementation of a machine learning pipeline designed to predict the survival of passengers aboard the Titanic using ensemble learning methods. The primary objective was to develop a model that could effectively classify passengers as survivors or non-survivors based on their characteristics. The solution employs feature engineering, model selection, hyperparameter tuning, and ensemble techniques to achieve a robust predictive model.

2. Data Exploration and Preprocessing

The Titanic dataset contains several features, including passenger demographics, ticket information, and cabin details. The dataset was split into training and test sets, with the training set used to build the model and the test set for evaluating the final predictions.

2.1. Data Loading

The data was loaded into pandas DataFrames from CSV files:

```
train_df = pd.read_csv('/your-drive/titanic/train.csv')
test_df = pd.read_csv('/your-drive/titanic/test.csv')
```

2.2. Feature Engineering

Feature engineering involved creating new features and modifying existing ones to improve the model's predictive power.

- **Title Extraction**: Titles were extracted from the `Name` field, grouped into categories, and rare titles were combined into a single 'Rare' category.
- **Family Size and IsAlone**: Two new features were created: `FamilySize`, which sums the number of siblings/spouses and parents/children, and `IsAlone`, which indicates if a passenger is travelling alone.
- **Deck Extraction**: The first character of the `Cabin` field was used to extract the deck information, with missing values labelled as 'M' for missing.
- **Fare Per Person**: A new feature was created by dividing the fare by the family size.
- **Log Transformation of Fare**: The `Fare` feature was log-transformed to reduce skewness.
- **Age Group Binning**: The `Age` feature was binned into categories (Child, Teen, YoungAdult, Adult, Senior).

- **Interaction Features**: Interaction features like `Pclass*Sex` and `Pclass*FamilySize` were created to capture relationships between different features.

2.3. Handling Missing Values

- **Age**: Missing values were filled using the median age of passengers grouped by `Pclass` and `Sex`.
- **Embarked**: Missing values were filled with the mode of the `Embarked` feature.
- **Fare**: Missing values in the `Fare` feature were filled with the median fare.

2.4. Preprocessing Pipeline

A preprocessing pipeline was created to handle numerical and categorical features separately:

- **Numerical Features**: Imputation of missing values with the median followed by standard scaling.
- **Categorical Features**: Imputation of missing values with the most frequent value followed by one-hot encoding.

The pipeline was then applied to both the training and test sets.

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ]), numerical_features),
        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))
        ]), categorical_features)
    ])
```

3. Model Training and Evaluation

The core of this project involved training various machine learning models and using an ensemble method (stacking) to combine their strengths.

3.1. Model Selection

Five models were selected based on their ability to handle tabular data and their popularity in classification tasks:

- Logistic Regression
- Random Forest

- Gradient Boosting
- XGBoost
- CatBoost

Each model was evaluated using cross-validation to establish a baseline accuracy.

```
models = {
   'LogisticRegression': log_reg,
   'RandomForest': rf,
   'GradientBoosting': gb,
   'XGBoost': xgb_model,
   'CatBoost': catboost_model
}
```

3.2. Hyperparameter Tuning

To improve model performance, hyperparameters were tuned using RandomizedSearchCV. This method allows efficient exploration of the hyperparameter space.

```
param_distributions_rf = {
   'n_estimators': [100, 200, 300, 400, 500],
   'max_depth': [4, 8, 12, 16, None]
}
```

Similar hyperparameter grids were defined for XGBoost and CatBoost. The best parameters were selected based on cross-validation accuracy.

3.3. Ensemble Methods

Two stacking models were created using different meta-learners: CatBoost and MLPClassifier. These meta-learners were chosen to see if a neural network-based approach could outperform a gradient boosting model.

- **CatBoost Meta-Learner**: The first stacking model used CatBoost as the meta-learner.
- **MLPClassifier Meta-Learner**: The second stacking model used a Multi-Layer Perceptron (MLP) as the meta-learner.

The two models were compared using cross-validation, and the better-performing model was selected for final predictions.

```
stacking_clf_catboost = StackingClassifier(
   estimators=[('rf', rf_search.best_estimator_),
               ('xgb', xgboost_search.best_estimator_),
               ('cat', catboost_search.best_estimator_)],
```

```
    final_estimator=CatBoostClassifier(
        n_estimators=300,
        learning_rate=0.021544346900318832,
        depth=8,
        verbose=0,
        random_state=42
    )
)
```

4. Results and Discussion

The final models were evaluated on the test set after selecting the best-performing stacking model. The results were documented and analysed.

- **Model Performance**: The CatBoost-based stacking model achieved slightly better cross-validation accuracy than the MLP-based model.
- **Final Submission**: The predictions from the selected model were used to create the final submission.

```
submission = pd.DataFrame({
    "PassengerId": test_df['PassengerId'],
    "Survived": predictions
})
submission.to_csv('/your-drive/titanic/final_submission.csv', index=False)
```

5. Critique and Potential Improvements

While the approach taken was effective, there are several areas where the methodology could be improved or expanded:

- **Feature Selection**: The current model includes all engineered features. A more thorough feature selection process, possibly using L1 regularisation or recursive feature elimination, could reduce overfitting.
- **Ensemble Methods**: While stacking provided a performance boost, other ensemble methods such as boosting or bagging could have been explored further.
- **Deep Learning Models**: Instead of using MLP as a meta-learner, a more complex deep learning model could have been tested, although this would require more computational resources.
- **Automated Machine Learning (AutoML)**: AutoML frameworks could have been employed to automate hyperparameter tuning and model selection, potentially finding better configurations.

- **Computational Constraints**: The RandomizedSearchCV approach, while thorough, is computationally expensive. GridSearchCV, while more exhaustive, would have taken considerably longer. Bayesian optimization could be a more efficient alternative.

6. Conclusion

This document outlined a comprehensive approach to building a predictive model for Titanic survival using advanced machine learning techniques. By focusing on careful feature engineering, methodical model tuning, and ensemble learning, the project successfully improved predictive accuracy. The inclusion of thorough cross-validation, logging, and evaluation ensures the model's robustness and generalizability. Future work could involve exploring alternative meta-learners, feature selection methods, and deep learning approaches to further enhance performance.