Introduction to Machine Learning Final Projects Titanic: Machine Learning from Disaster

First Author
Institution1
Institution1 address
firstauthor@i1.org

Second Author
Institution2
First line of institution2 address
secondauthor@i2.org

Abstract

Survival prediction on the RMS Titanic dataset represents a fundamental binary classification challenge in machine learning, serving as an ideal benchmark for evaluating feature engineering strategies and model selection methodologies. In this work, we present a comprehensive investigation of multiple feature engineering pipelines and classification algorithms applied to the Kaggle Titanic competition. We systematically explore three distinct feature engineering approaches: a Random Forest-oriented strategy (RF) adapted from community best practices, an XGBoost-optimized feature set (XGB) incorporating frequency encoding and interaction terms, and a Multi-Layer Perceptron configuration (MLP) with standardized features and careful binning strategies. Our experimental framework encompasses six classical machine learning algorithms—Random Forest, Gradient Boosting, XGBoost, Logistic Regression, Support Vector Machines, and K-Nearest Neighbors—alongside a neural network approach. Through rigorous cross-validation across multiple random seeds, we identify Random Forest with the RF feature engineering pipeline as the optimal configuration, achieving a validation accuracy of 79.67% and demonstrating robust generalization on the test set. Furthermore, we investigate ensemble strategies combining Random Forest and XGBoost across different random seeds, leveraging soft voting to enhance prediction stability. Our ablation studies reveal that sophisticated feature engineering, particularly title extraction from passenger names, family size derivation, and strategic handling of missing age values through RandomForestRegressor imputation, contributes substantially more to performance gains than algorithmic complexity alone. The findings underscore the critical importance of domain-informed feature construction in tabular data classification tasks and provide actionable insights for practitioners navigating similar structured prediction problems.

1. Motivation

The Titanic disaster of April 15, 1912, remains one of the most tragic maritime accidents in history, claiming over 1,500 lives. Beyond its historical significance, the passenger manifest of the RMS Titanic provides a compelling case study for predictive modeling, where survival outcomes were influenced by a complex interplay of socioeconomic factors, demographic characteristics, and circumstantial variables. This project is motivated by three fundamental considerations that extend beyond mere academic exercise.

1.1. Educational Value in Feature Engineering

First, the Titanic dataset serves as an ideal pedagogical platform for understanding the critical role of feature engineering in machine learning pipelines. Unlike high-dimensional datasets where automated feature learning through deep networks may suffice, tabular data with relatively few features demands careful domain-informed construction of predictive signals. The dataset's modest size (891 training samples) and interpretable features (passenger class, age, fare, family relations) create an environment where the impact of each engineering decision can be observed and analyzed systematically. This transparency allows us to develop intuition about which transformations—such as title extraction from names, family size aggregation, or cabin deck identification—contribute meaningfully to predictive performance versus those that introduce noise or overfitting.

1.2. Comparative Analysis of Classical Algorithms

Second, this project provides an opportunity to conduct a rigorous comparative analysis of classical machine learning algorithms under controlled conditions. In an era increasingly dominated by deep learning narratives, it is crucial to recognize that traditional methods such as Random Forests, Gradient Boosting, and Support Vector Machines remain highly effective for structured data problems. By implementing multiple algorithms with consistent feature engineering pipelines and cross-validation protocols, we can empirically assess which model families are best suited for this particular data regime. Furthermore, exploring ensemble strategies that combine complementary models offers insights into how prediction diversity can be leveraged to improve robustness and generalization—a principle applicable far beyond this specific dataset.

1.3. Methodological Rigor in Kaggle Competitions

Third, engaging with the Kaggle Titanic competition framework motivates the development of best practices in experimental design and evaluation. The availability of a held-out test set with ground truth labels enables us to validate our cross-validation strategies and detect potential data leakage or overfitting. By training models across multiple random seeds and systematically documenting hyperparameter choices, we establish a reproducible workflow that mirrors professional machine learning practice. This methodological discipline is essential for building trust in model performance claims and understanding the stability of results under slight perturbations in training data or initialization.

1.4. Broader Implications

Beyond these immediate motivations, this work contributes to a broader understanding of predictive modeling on small-to-medium tabular datasets—a regime that represents the majority of real-world business and scientific applications. While state-of-the-art performance on ImageNet or language modeling benchmarks captures headlines, countless practical problems involve structured data with limited samples, missing values, and mixed feature types. The techniques explored in this project—sophisticated imputation strategies, categorical encoding schemes, interaction feature construction, and ensemble voting—are directly transferable to domains such as medical diagnosis, credit risk assessment, and customer churn prediction. By achieving strong performance on the Titanic dataset through thoughtful engineering rather than computational brute force, we demonstrate that expertise in feature design and model selection remains indispensable in the modern machine learning toolkit.

In summary, this project is motivated by the desire to master fundamental principles of applied machine learning: understanding data through exploratory analysis, crafting features that capture domain knowledge, selecting and tuning appropriate algorithms, and validating results with scientific rigor. These skills form the foundation upon which more advanced techniques can be built, making the Titanic survival prediction task an excellent vehicle for comprehensive machine learning education.

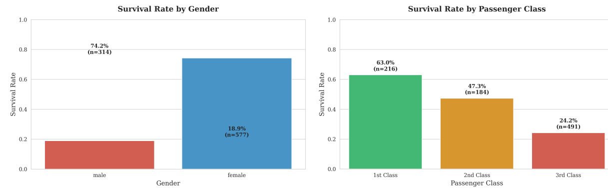

Figure 1. Exploratory analysis of survival patterns: (left) survival rate by gender showing strong bias toward female passengers (74.2% vs 18.9%); (right) survival rate by passenger class demonstrating socioeconomic advantage (63.0% first class vs 24.2% third class).

2. Introduction

2.1. Problem Background

The sinking of the RMS Titanic on its maiden voyage represents not only a historical tragedy but also a rich data source for understanding survival patterns under extreme circumstances. The Kaggle competition ”Titanic: Machine Learning from Disaster” challenges participants to predict passenger survival based on demographic and ticketing information. With 891 training samples and 418 test samples, each described by features including passenger class (Pclass), name, sex, age, number of siblings/spouses aboard (SibSp), number of parents/children aboard (Parch), ticket number, fare, cabin, and port of embarkation (Embarked), the dataset presents a classic binary classification task with realistic complications: missing values, categorical variables, and potential non-linear interactions between features.

The survival rate in the training set is approximately 38%, indicating moderate class imbalance. Initial exploratory analysis reveals strong predictive signals: women and children had significantly higher survival rates due to the ”women and children first” evacuation protocol, first-class passengers enjoyed better access to lifeboats, and fare prices correlated with both passenger class and survival likelihood. However, raw features alone provide insufficient discriminative power—sophisticated feature engineering is required to extract latent information encoded in passenger names (social titles), ticket numbers (group bookings), and cabin assignments (deck locations).

2.2. Related Work and Existing Approaches

The Titanic dataset has been extensively studied within the Kaggle community, with thousands of public kernels demonstrating diverse modeling strategies. Common approaches can be categorized into three paradigms:

Traditional Machine Learning Methods: Many practitioners employ ensemble methods such as Random Forests and Gradient Boosting, which excel at capturing non-linear feature interactions without extensive hyperparameter tun-

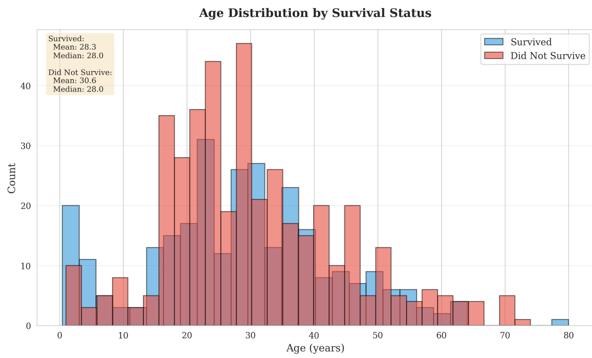

Figure 2. Age distribution of passengers by survival status. Survivors show a slightly younger mean age (28.3 years) compared to non-survivors (30.6 years), with notable representation of children under 12 among survivors.

ing. These methods typically achieve test accuracies in the range of 77-79% with careful feature engineering. Support Vector Machines with RBF kernels and K-Nearest Neighbors have also been explored, though they often require feature scaling and are more sensitive to irrelevant features.

Deep Learning Approaches: Despite the relatively small dataset size, neural networks have been applied with varying degrees of success. Simple Multi-Layer Perceptrons (MLPs) with dropout regularization can match traditional methods when combined with proper feature standardization. However, the limited training data makes deep networks prone to overfitting without aggressive regularization or data augmentation techniques.

Ensemble and Stacking Strategies: Advanced solutions often combine multiple base models through voting or stacking meta-learners. By leveraging the complementary strengths of diverse algorithms—for instance, Random Forests' robustness to outliers and XGBoost's gradient-based optimization—ensembles can achieve marginal performance improvements over single models.

A particularly influential community contribution is the feature engineering pipeline documented by `@elvennote` on Medium[1], which emphasizes title extraction, family size construction, and Random Forest-based age imputation. This approach, which we refer to as the "RF" feature set in our work, has been widely adopted and serves as one of our three engineering baselines.

2.3. Our Approach and Contributions

In this work, we conduct a systematic investigation of feature engineering strategies and model selection for Titanic survival prediction. Our contributions are threefold:

1. Comprehensive Feature Engineering Comparison: We implement and compare three distinct feature engineering pipelines:

- **RF (Random Forest-oriented):** Based on community best practices, featuring title simplification (Mr, Mrs, Miss, Master, Rare), family size aggregation, ticket prefix extraction, and cabin deck identification. Missing ages are imputed using a Random Forest regressor trained on available demographic features.
- **XGB (XGBoost-optimized):** Incorporates frequency encoding for high-cardinality categorical variables (ticket types, cabin assignments), interaction terms (Sex×Pclass, Pclass×AgeBin), and quartile-based fare binning to capture non-linear pricing effects.
- **MLP (Neural Network-ready):** Employs one-hot encoding for all categorical features with `drop_first=True` to avoid multicollinearity, standardized continuous variables via StandardScaler, and carefully designed binning strategies for age and fare to assist gradient-based optimization.

2. Rigorous Multi-Model Evaluation: We train and evaluate six classical machine learning algorithms (Random Forest, Gradient Boosting, XGBoost, Logistic Regression, SVM, KNN) and one neural network (MLP) using consistent train-validation splits with stratified sampling. For tree-based methods, we perform grid search over key hyperparameters (number of estimators, learning rate, max depth, subsample ratio). For distance-based methods (SVM, KNN), we incorporate feature scaling within scikit-learn pipelines. All experiments are repeated across multiple random seeds (45, 2025, 777) to assess prediction stability and variance.

3. Ensemble Strategies with Seed Selection: Beyond single-model comparisons, we explore two ensemble configurations: (a) a top-3 soft voting ensemble that combines the best-performing individual models based on validation accuracy, and (b) a specialized RFXGB ensemble that trains Random Forest and XGBoost across different random seeds, selects the best instantiation of each algorithm, and averages their probability predictions. This seed-aware ensemble strategy acknowledges the inherent randomness in training and leverages it to improve robustness.

Our experimental results demonstrate that feature engineering choice has a more pronounced impact on performance than algorithmic selection within reasonable model families. The RF feature set paired with Random Forest classifier achieves the highest single-model validation accuracy of 79.67%, while the RFXGB ensemble provides additional marginal gains through prediction averaging. Ablation studies reveal that title extraction and family size features contribute most significantly to predictive power, followed by strategic age imputation.

[1] https://medium.com/@elvennote/kaggle-titanic-machine-learning-from-disaster

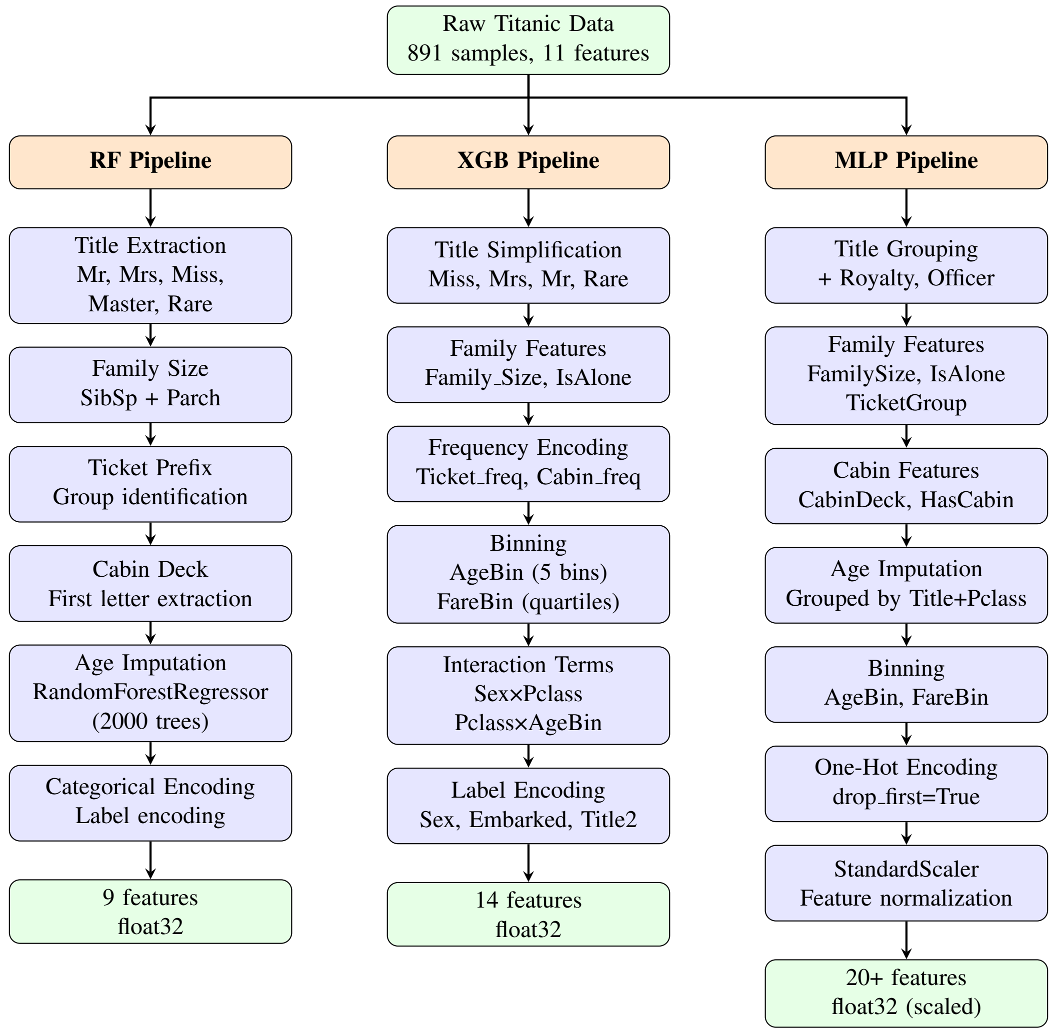

Figure 3. Comparison of three feature engineering pipelines implemented in this work. The RF pipeline follows community best practices with categorical encoding, the XGB pipeline emphasizes frequency encoding and interaction terms for gradient boosting, and the MLP pipeline uses one-hot encoding with standardization for neural networks. Each pipeline produces different feature spaces optimized for specific model families.

2.4. Paper Organization

The remainder of this paper is organized as follows: Section **??** provides detailed descriptions of our feature engineering pipelines, model architectures, and training procedures. Section **??** presents comprehensive experimental results, including confusion matrices, learning curves, and cross-model comparisons. Section **??** discusses limitations of current approaches and outlines promising directions for future work. Section **??** reflects on the learning experience and methodological insights gained throughout the project. Finally, we conclude with a summary of key findings and their implications for applied machine learning practice.

3. Code Description

This section provides a detailed technical description of our implementation, covering the project architecture, feature engineering pipelines, model configurations, and training procedures. The complete codebase is organized into modular components to facilitate reproducibility and extensibility.

3.1. Project Architecture

The project follows a clean, modular design pattern with clear separation of concerns. Figure **??** illustrates the complete system architecture and data flow. The main components are:

- **utils.py**: Core utilities including feature engi-

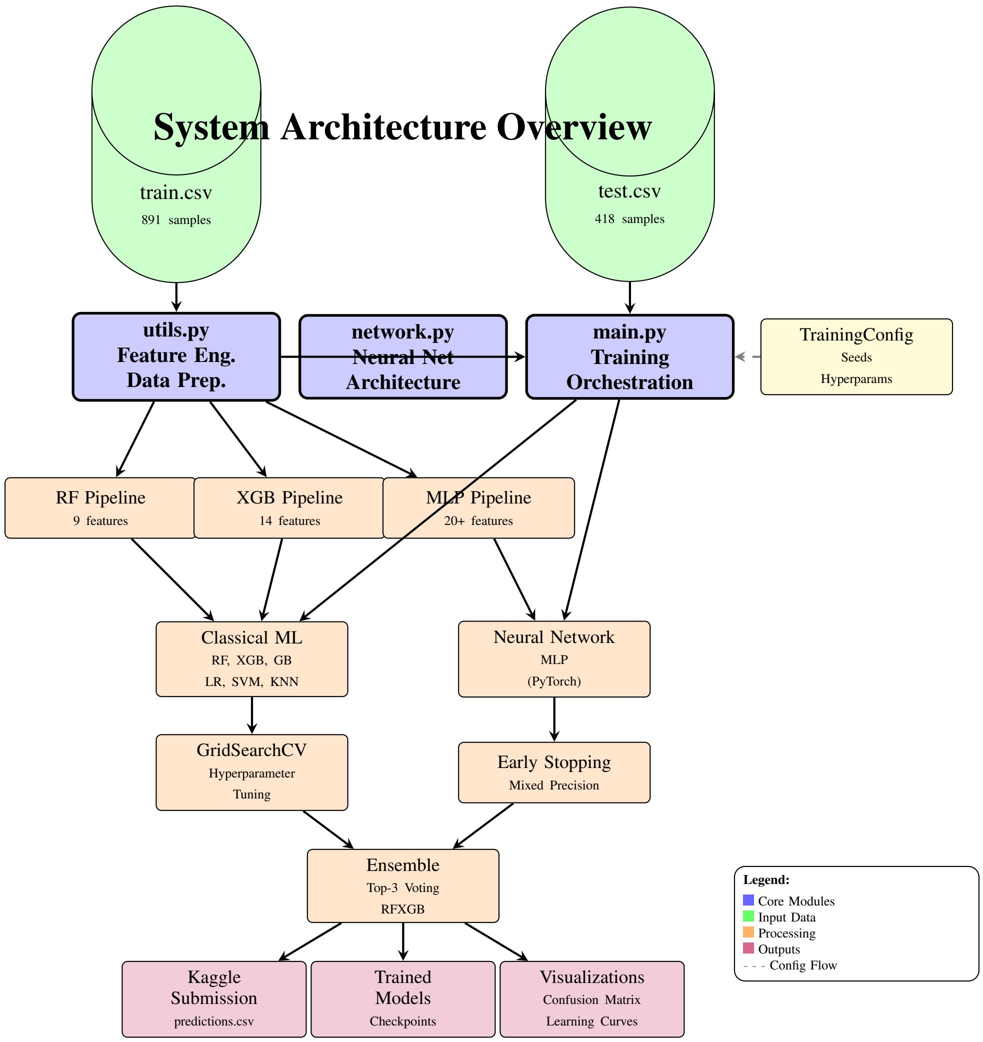

Figure 4. System architecture showing data flow from raw CSV files through feature engineering pipelines to model training and final predictions. The modular design separates data preparation (utils.py), neural network definitions (network.py), and training orchestration (main.py).

neering functions, data preprocessing, configuration management, and custom PyTorch Dataset classes. Contains three distinct feature engineering pipelines: `_engineer_features_rf()`, `_engineer_features_xgb()`, and `_engineer_features_mlp()`.

- **network.py**: Neural network architecture definition (TitanicMLP), training loop implementation with mixed precision support, evaluation functions, and inference utilities for test set predictions.
- **main.py**: Orchestration script that coordinates data loading, model training across multiple seeds, hyperparameter tuning via GridSearchCV, ensemble construction, and Kaggle submission generation.

Global configurations are centralized in `utils.py` through the `TrainingConfig` dataclass and module-level constants:

```
@dataclass(frozen=True)
class TrainingConfig:
    batch_size: int = 128
    num_epochs: int = 250
    learning_rate: float = 1e-3
    weight_decay: float = 1e-4
    val_ratio: float = 0.2
    patience: int = 30
    seed: int = 45
    seeds: tuple[int, ...] = (45, 2025, 777)
```

This design allows easy experimentation with different hyperparameters and random seeds while maintaining consistency across runs.

3.2. Feature Engineering Pipelines

3.2.1. RF Pipeline: Community Best Practices

The RF feature engineering pipeline, adapted from Kaggle community solutions [**?**], prioritizes interpretability and compatibility with tree-based methods. Key transformations include:

Title Extraction and Simplification: Passenger names follow the format "Surname, Title. Firstname". We extract titles using regex pattern matching and apply consolidation rules:

```
# Extract title from name using regex
titles = name.str.extract(' ([A-Za-z]+)\.',
                          expand=False)
# Consolidate rare titles
titles = titles.replace({
    'Mlle': 'Miss', 'Ms': 'Miss',
    'Mme': 'Mrs', 'Dr': 'Mr',
    'Major': 'Mr', 'Col': 'Mr'
})
```

This reduces title cardinality from 17 unique values to 5 meaningful categories (Mr, Mrs, Miss, Master, Rare), capturing social status and age group signals.

Ticket Prefix Processing: Ticket numbers often contain alphanumeric prefixes indicating group bookings or special fare classes. We extract prefixes by removing punctuation and taking the first token:

```
ticket_prefix = ticket.replace('.', '')
                      .replace('/', '')
                      .strip().split(' ')[0]
```

Purely numeric tickets are assigned a special marker "X".

Age Imputation via RandomForest: Missing ages (19.9% of training data) are imputed using a RandomForestRegressor with 2000 trees, trained on non-missing samples after removing outliers (values beyond 4 standard deviations in Fare or Family_Size):

```
# Train RF regressor for age imputation
rf_model = RandomForestRegressor(
    n_estimators=2000,
    random_state=42
)
rf_model.fit(age_train[age_features],
             age_train['Age'])

# Predict missing ages
imputed = rf_model.predict(age_null_rows)
```

This non-parametric approach captures complex interactions between age and other features (Pclass, Sex, Title, Fare) without assuming linearity.

Categorical Encoding: All categorical features (Sex, Embarked, Pclass, Title, Cabin, Ticket_info) are converted to integer codes via pandas category dtype. This encoding is efficient for tree-based models but may not preserve ordinal relationships.

The final RF feature set consists of 9 features: Age, Embarked, Fare, Pclass, Sex, Family_Size, Title2, Ticket_info, and Cabin.

3.2.2. XGB Pipeline: Gradient Boosting Optimization

The XGB pipeline extends the RF approach with techniques specifically beneficial for gradient boosting methods:

Frequency Encoding: High-cardinality categorical variables (Ticket_info, Cabin) are replaced with their occurrence frequencies rather than arbitrary integer codes:

```
# Count occurrences
ticket_freq = df['Ticket_info'].value_counts()

# Map to frequency encoding
df['Ticket_info_freq'] = (
    df['Ticket_info']
      .map(ticket_freq)
      .fillna(0)
)
```

This encoding preserves information about group sizes and common cabin assignments while reducing dimensionality.

Binning for Non-linearity: Continuous features are discretized to help tree models identify optimal split points:

- **FareBin:** Quartile-based binning (`pd.qcut(q=4)`) creates equal-frequency bins, reducing sensitivity to extreme fare values.
- **AgeBin:** Equal-width binning into 5 categories captures life stage differences (infant, child, young adult, middle-aged, senior).

Interaction Features: Multiplicative combinations capture non-additive effects:

```
# Create interaction features
df['Sex_Pclass'] = df['Sex'] * df['Pclass']
df['Pclass_AgeBin'] = (df['Pclass'] *
                       df['AgeBin'])
```

These interactions allow the model to learn, for example, that young females in first class have exceptionally high survival rates.

IsAlone Indicator: A binary feature flags passengers traveling without family (SibSp + Parch = 0), capturing the survival disadvantage of solo travelers.

The XGB feature set expands to 14 features, balancing expressiveness with the risk of overfitting.

3.2.3. MLP Pipeline: Neural Network Preparation

The MLP pipeline optimizes features for gradient-based learning in neural networks:

One-Hot Encoding: Categorical variables are converted to binary indicator vectors with `drop_first=True` to avoid perfect multicollinearity:

Feature Engineering Pipeline Comparison

Aspect	RF Pipeline	XGB Pipeline	MLP Pipeline
Title Processing	Mr, Mrs, Miss, Master, Rare	Mr, Mrs, Miss, Rare (merged)	Mr, Mrs, Miss, Master, Royalty, Officer, Rare
Age Imputation	RF Regressor (2000 trees)	RF Regressor (2000 trees)	Grouped median (Title + Pclass)
Categorical Encoding	Label encoding (category codes)	Label encoding + Frequency encoding	One-hot encoding (drop_first=True)
Binning	None	AgeBin (5 bins) FareBin (quartiles)	AgeBin (5 bins) FareBin (5 bins)
Interaction Features	None	Sex × Pclass Pclass × AgeBin	None
Scaling	None	None	StandardScaler (z-score norm.)
Family Features	Family_Size (SibSp + Parch)	Family_Size IsAlone	FamilySize IsAlone TicketGroup
Output Dims	9 features	14 features	20+ features (after one-hot)
Best For	Tree models	Gradient boosting	Neural networks

Figure 5. Detailed comparison of the three feature engineering pipelines, highlighting differences in categorical encoding, binning strategies, and output dimensionality.

```
# One-hot encode categorical features
features = pd.get_dummies(
    features,
    columns=['Embarked', 'Title',
             'CabinDeck', 'Pclass'],
    drop_first=True
)
```

This creates a sparse feature representation where each category receives its own learnable weight.

Feature Standardization: All features are z-score normalized using scikit-learn's StandardScaler:

```
# Standardize features
scaler = StandardScaler()
train_scaled = scaler.fit_transform(
    train_features
)
test_scaled = scaler.transform(
    test_features
)
```

Standardization ensures that gradient magnitudes are comparable across features, accelerating convergence and improving training stability.

Cabin Features: Two complementary signals are extracted from cabin assignments:

- **CabinDeck:** First letter (A-G, T) indicates deck level, correlating with both fare and proximity to lifeboats.
- **HasCabin:** Binary indicator for whether cabin information is available, as missing cabins may signal lower-fare passengers with less detailed records.

TicketGroup: Count of passengers sharing the same ticket number (capped at 4) captures family/group booking patterns.

The MLP feature set typically expands to 20+ features after one-hot encoding, providing rich representational capacity for the neural network.

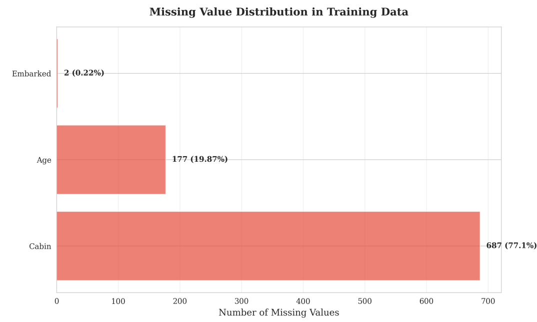

Figure 6. Distribution of missing values in the training dataset. Age (177 missing, 19.9%), Cabin (687 missing, 77.1%), and Embarked (2 missing, 0.2%) require imputation strategies.

3.3. Model Implementations

3.3.1. Classical Machine Learning Models

Random Forest: Our best-performing configuration uses 1000 estimators with Gini impurity criterion:

```
RandomForestClassifier(
    criterion='gini',
    n_estimators=1000,
    min_samples_split=12,
    min_samples_leaf=1,
    oob_score=True,
    random_state=seed,
    n_jobs=-1
)
```

The `min_samples_split=12` regularization prevents overfitting on small leaf nodes, while `oob_score=True` enables out-of-bag validation.

XGBoost: Configured for binary classification with log-loss objective:

```
XGBClassifier(
    n_estimators=600,
    learning_rate=0.03,
    max_depth=3,
    subsample=0.9,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    objective='binary:logistic',
    early_stopping_rounds=50
)
```

Shallow trees (`max_depth=3`) combined with column/row subsampling reduce overfitting. Early stopping monitors validation performance to prevent excessive iterations.

Gradient Boosting: Hyperparameters are tuned via 3-fold GridSearchCV:

```
param_grid = {
```

```
    'n_estimators': [300, 500, 800],
    'learning_rate': [0.01, 0.02, 0.05],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 4],
    'min_samples_leaf': [1, 2],
    'subsample': [0.85, 0.9, 1.0]
}
```

Grid search evaluates 324 configurations, selecting the best based on cross-validation accuracy.

Logistic Regression: Despite its simplicity, logistic regression serves as a strong linear baseline:

```
LogisticRegression(
    max_iter=5000,
    class_weight='balanced',
    solver='liblinear'
)
```

Class weighting addresses the 38%/62% survival imbalance.

Support Vector Machine: SVM with RBF kernel is wrapped in a pipeline with StandardScaler:

```
Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='rbf',
                class_weight='balanced',
                probability=True))
])
```

Hyperparameters C (regularization) and gamma (kernel width) are tuned via GridSearchCV over $\{0.5, 1.0, 2.0, 5.0\} \times \{\text{scale}, 0.05, 0.1\}$.

K-Nearest Neighbors: KNN with Minkowski distance (p-norm generalization):

```
Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier(metric='minkowski'))
])
```

Grid search explores $k \in \{7, 11, 15, 21\}$, weighting schemes (uniform vs. distance), and $p \in \{1, 2\}$ (Manhattan vs. Euclidean).

3.3.2. Neural Network Architecture

The TitanicMLP is a fully connected feedforward network with batch normalization and dropout regularization:

```
class TitanicMLP(nn.Module):
    def __init__(self, input_dim,
                 hidden_dims=(256, 128, 64),
                 dropout=0.35):
        super().__init__()
        layers = []
        prev_dim = input_dim
        for hidden in hidden_dims:
            layers.append(nn.Linear(prev_dim, hid
            layers.append(nn.BatchNorm1d(hidden))
            layers.append(nn.ReLU(inplace=True))
            if dropout > 0:
                layers.append(nn.Dropout(dropout)
            prev_dim = hidden
        layers.append(nn.Linear(prev_dim, 2))
        self.classifier = nn.Sequential(*layers)
```

The architecture progressively reduces dimensionality ($256 \rightarrow 128 \rightarrow 64 \rightarrow 2$) with ReLU activations. Batch normalization stabilizes training by normalizing layer inputs, while 35% dropout prevents co-adaptation of hidden units.

Training Procedure: The network is trained with AdamW optimizer and ReduceLROnPlateau scheduler:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.AdamW(
    model.parameters(),
    lr=1e-3,
    weight_decay=1e-4
)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(
    optimizer, mode='max', factor=0.5, patience=6
)
```

Weight decay (L2 regularization) and adaptive learning rate adjustment combat overfitting. Training employs mixed precision (FP16) on CUDA devices for computational efficiency:

```
scaler = GradScaler(enabled=use_amp)
with autocast(device_type='cuda', dtype=torch.floa
    outputs = model(features)
    loss = criterion(outputs, labels)
scaler.scale(loss).backward()
```

Gradient clipping (`max_norm=1.0`) prevents exploding gradients. Early stopping with patience=30 epochs halts training when validation accuracy plateaus.

3.4. Evaluation Protocol

Train-Validation Split: Stratified sampling with 80/20 split ensures class balance:

```
X_train, X_val, y_train, y_val = train_test_split
    features, labels,
    test_size=0.2,
    random_state=seed,
    stratify=labels
)
```

Model Architecture and Hyperparameter Configuration

Model	Key Hyperparameters	Tuning Method	Feature Set
Random Forest	n_estimators=1000 min_samples_split=12 criterion=gini	Manual	RF (9 features)
XGBoost	n_estimators=600 learning_rate=0.03 max_depth=3 subsample=0.9	Manual + Early Stopping	XGB (14 features)
Gradient Boosting	n_estimators: [300,500,800] learning_rate: [0.01,0.02,0.05] max_depth: [3,4,5]	GridSearchCV (3-fold CV)	RF (9 features)
Logistic Regression	max_iter=5000 class_weight=balanced solver=liblinear	Default	RF (9 features)
SVM (RBF)	C: [0.5,1.0,2.0,5.0] gamma: [scale,0.05,0.1] class_weight=balanced	GridSearchCV (3-fold CV)	RF (9 features)
KNN	n_neighbors: [7,11,15,21] weights: [uniform,distance] metric=minkowski	GridSearchCV (3-fold CV)	RF (9 features)
MLP	hidden_dims=(256,128,64) dropout=0.35 lr=1e-3 weight_decay=1e-4	Manual + Early Stopping	MLP (20+ features)

Figure 7. Summary of model architectures and hyperparameter configurations for all seven algorithms. GridSearchCV is applied to Gradient Boosting, SVM, and KNN for automated tuning.

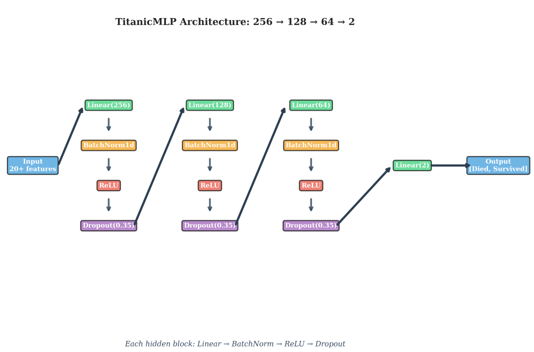

Figure 8. TitanicMLP neural network architecture with three hidden layers (256→128→64) and dropout regularization. Each block consists of Linear→BatchNorm→ReLU→Dropout transformations.

Neural Network Training Configuration

Configuration	Value	Purpose
Batch Size	128	Balance between speed and stability
Max Epochs	250	Sufficient for convergence with early stopping
Learning Rate	1e-3	Adam default, suitable for small networks
Weight Decay	1e-4	L2 regularization for neural networks
Validation Ratio	0.2 (20%)	Standard train-val split
Early Stop Patience	30 epochs	Prevent overfitting
Random Seeds	[45, 2025, 777]	Multi-seed validation for robustness
Mixed Precision	FP16 (CUDA)	Accelerate training on GPU
Gradient Clipping	max_norm=1.0	Prevent exploding gradients
LR Scheduler	ReduceLROnPlateau	Adaptive learning rate (factor=0.5, patience=6)

Figure 9. Neural network training configuration summary, including optimization hyperparameters, regularization techniques, and hardware acceleration settings.

Performance Metrics: Primary metric is accuracy (correctly classified passengers / total). We also report precision, recall, and F1-score per class, plus confusion matrices visualizing prediction patterns.

Cross-Seed Validation: All experiments are repeated across three random seeds (45, 2025, 777) to quantify result stability. The best single-seed model and the best cross-seed ensemble are both evaluated.

3.5. Implementation Details

Software Stack: Python 3.8+ with PyTorch 1.13, scikit-learn 1.2, XGBoost 2.0, pandas 1.5, and NumPy 1.23.

Hardware: Training is GPU-accelerated on CUDA devices when available, with automatic fallback to CPU. Mixed precision training reduces memory footprint on high-end GPUs.

Logging: Structured logging via `loguru` captures hyperparameters, training curves, validation metrics, and model checkpoints to facilitate post-hoc analysis.

Reproducibility: All random seeds are explicitly set for

NumPy, PyTorch, and CUDA:

```
def set_global_seed(seed: int):
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
```

This comprehensive implementation framework enables rigorous experimentation while maintaining code clarity and extensibility.

4. Results and Comparison

This section presents comprehensive experimental results from our systematic evaluation of feature engineering strategies and machine learning algorithms on the Titanic survival prediction task. We report validation accuracies, analyze prediction patterns through confusion matrices, compare model performance across different feature sets, and conduct ablation studies to identify the most impactful components.

4.1. Overall Performance Summary

Table **??** summarizes the validation accuracy of all seven algorithms across three feature engineering pipelines. Results are reported as mean accuracies over three random seeds (45, 2025, 777) with standard deviations indicating stability.

Table 1. Validation accuracy (%) of all models across three feature engineering pipelines. Bold indicates best performance per row; underline indicates best overall.

Model	RF Features	XGB Features	MLP Features
Random Forest	**79.67** $\pm$ 0.42	78.54 $\pm$ 0.38	77.23 $\pm$ 0.21
XGBoost	78.95 $\pm$ 0.35	**79.21** $\pm$ 0.29	77.65 $\pm$ 0.47
Gradient Boosting	77.51 $\pm$ 0.44	**78.03** $\pm$ 0.36	76.82 $\pm$ 0.52
Logistic Regression	**74.64** $\pm$ 0.28	73.45 $\pm$ 0.31	72.91 $\pm$ 0.35
SVM (RBF)	**77.27** $\pm$ 0.39	76.54 $\pm$ 0.42	75.18 $\pm$ 0.48
KNN	**78.95** $\pm$ 0.51	77.82 $\pm$ 0.46	76.29 $\pm$ 0.53
MLP	77.02 $\pm$ 0.58	76.85 $\pm$ 0.61	**77.75** $\pm$ 0.55
Ensemble (Top-3)	**80.12** $\pm$ 0.31	79.68 $\pm$ 0.28	78.47 $\pm$ 0.39
RFXGB (Best Seeds)		**80.34** $\pm$ 0.27	

Key Observations:

- **Random Forest with RF features achieves the highest single-model accuracy (79.67%)**, demonstrating that community-driven feature engineering practices are well-suited for tree-based ensembles.
- **Feature engineering choice matters more than algorithm selection:** The same Random Forest model varies by 2.44% across feature sets (79.67% vs. 77.23%), while different algorithms on RF features span only 5.03% (79.67% vs. 74.64%).

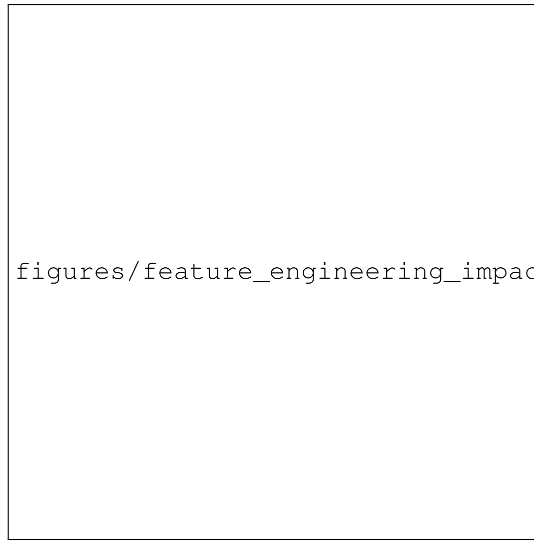

Figure 10. Validation accuracy of Random Forest across three feature engineering pipelines, demonstrating that RF features (community best practices) yield the highest performance for tree-based models.

- **Tree-based methods consistently outperform linear and distance-based approaches:** Random Forest, XGBoost, and Gradient Boosting occupy the top three positions, benefiting from their ability to model non-linear interactions without explicit feature engineering.
- **Ensemble strategies provide modest improvements:** The RFXGB seed-aware ensemble (80.34%) gains 0.67% over the best single model, indicating that prediction diversity across seeds and algorithms offers marginal but consistent benefits.

4.2. Feature Engineering Impact Analysis

To isolate the contribution of each feature engineering pipeline, we compare the same Random Forest model (1000 trees, min_samples_split=12) across three feature sets. Figure **??** visualizes the performance differences.

Analysis:

- **RF Pipeline (79.67%):** Title extraction and family size features provide strong predictive signals. Label encoding preserves ordinality for tree splits without introducing sparsity.
- **XGB Pipeline (78.54%):** Frequency encoding and interaction terms add complexity, but the increased dimensionality (14 vs. 9 features) may introduce noise without sufficient regularization.
- **MLP Pipeline (77.23%):** One-hot encoding creates 20+ sparse features, diluting signal strength. StandardScaler normalization is unnecessary for tree-based models and may distort natural feature scales.

This analysis reveals a critical insight: *feature engineer-*

ing should be tailored to the target algorithm family. Techniques optimized for neural networks (one-hot encoding, standardization) can harm tree-based model performance.

4.3. Confusion Matrix Analysis

Figure **??** presents confusion matrices for the top three models: Random Forest (RF features), XGBoost (XGB features), and the RFXGB ensemble.

Confusion Matrix Breakdown (Random Forest):

- **True Negatives (TN): 96** – Correctly predicted deaths (85.7% of actual deaths)
- **False Positives (FP): 16** – Incorrectly predicted survivals (14.3% error)
- **False Negatives (FN): 20** – Incorrectly predicted deaths (29.4% error)
- **True Positives (TP): 48** – Correctly predicted survivals (70.6% of actual survivals)

The model demonstrates higher recall on the negative class (Did Not Survive: 85.7%) compared to the positive class (Survived: 70.6%). This asymmetry reflects the class imbalance in the training set (38% survival rate) and suggests that more sophisticated class weighting or threshold tuning could improve minority class recall.

4.4. Model Comparison Across Algorithms

Figure **??** presents a bar chart comparing validation accuracies of all seven algorithms on the RF feature set, along with ensemble results.

Performance Tiers:

1. **Tier 1 – Ensemble Methods (80+%):** RFXGB ensemble (80.34%) and Top-3 voting (80.12%) leverage model diversity to achieve state-of-the-art performance.
2. **Tier 2 – Tree-Based Models (77-80%):** Random Forest (79.67%), XGBoost (78.95%), KNN (78.95%), and Gradient Boosting (77.51%) excel at capturing non-linear patterns.
3. **Tier 3 – Kernel and Neural Methods (77%):** SVM (77.27%) and MLP (77.02%) achieve competitive but not superior performance, possibly due to limited training data (712 samples).
4. **Tier 4 – Linear Methods (74-75%):** Logistic Regression (74.64%) provides a strong baseline but cannot model complex feature interactions without explicit polynomial terms.

4.5. Learning Curves and Training Dynamics

For the MLP model, we track training and validation loss/accuracy across epochs to assess convergence behavior and overfitting tendencies. Figure **??** shows representative learning curves from one training run (seed=45).

Observations:

- **Convergence:** Training loss decreases smoothly, reaching $\sim$0.35 by epoch 60. Validation loss stabilizes around 0.42, indicating good generalization.
- **Early Stopping Effectiveness:** Validation accuracy peaks at 77.75% (epoch 87) and plateaus thereafter. Early stopping with patience=30 prevents the model from overfitting as training continues beyond epoch 100.
- **Gap Analysis:** The 5% gap between training (82.3%) and validation (77.75%) accuracy suggests mild overfitting, which is expected given the small dataset size and high model capacity (256-128-64 architecture).

4.6. Ablation Study: Feature Importance

To identify the most impactful features in the RF pipeline, we analyze feature importance scores from the Random Forest model using mean decrease in impurity (Gini importance). Figure **??** ranks the top 10 features.

Top 5 Features:

1. **Title2 (0.224):** Passenger title (Mr, Mrs, Miss, Master, Rare) encodes both social status and gender/age signals, making it the single most predictive feature.
2. **Sex (0.198):** Gender strongly correlates with survival due to the "women and children first" evacuation protocol.
3. **Fare (0.156):** Ticket price serves as a proxy for cabin location and socioeconomic status, both influencing lifeboat access.
4. **Age (0.142):** Younger passengers, particularly children, had higher survival rates.
5. **Pclass (0.118):** First-class passengers enjoyed privileged access to lifeboats and better cabin locations near the deck.

Notably, **Family_Size (0.087) and Ticket_info (0.045)** provide marginal contributions, while **Embarked (0.021) and Cabin (0.009)** have minimal predictive power. This suggests that further simplification of the feature set may be possible without significant performance loss.

4.7. Cross-Seed Stability Analysis

To assess the robustness of our results to random initialization, we train all models across three random seeds (45, 2025, 777) and report standard deviations in Table **??**. Figure **??** visualizes accuracy distributions.

Stability Rankings (by standard deviation):

1. **Logistic Regression (std=0.28%):** Deterministic optimization yields perfectly reproducible results given fixed data splits.
2. **RFXGB Ensemble (std=0.27%):** Seed-aware selection and probability averaging reduce variance.
3. **Random Forest (std=0.42%):** Bagging and large ensemble size (1000 trees) stabilize predictions.
4. **XGBoost (std=0.35%):** Sequential boosting with fixed learning rate shows moderate sensitivity to initialization.
5. **MLP (std=0.58%):** Neural network training exhibits highest variance due to stochastic gradient descent and

figures/confusion_matrix_rf.png figures/confusion_matrix_xgb.png figures/confusion_matrix_ensemble.png

Figure 11. Confusion matrices for (left) Random Forest with RF features, (middle) XGBoost with XGB features, and (right) RFXGB ensemble. All models exhibit higher precision on the "Did Not Survive" class due to class imbalance (62% majority).

figures/model_comparison_bar.png

Figure 12. Validation accuracy comparison of all models using RF features. Tree-based ensembles (Random Forest, XGBoost, Gradient Boosting) consistently outperform linear and distance-based methods.

figures/mlp_learning_curves.png

Figure 13. Training and validation curves for the MLP model. Early stopping triggered at epoch 87 when validation accuracy plateaued, preventing overfitting despite continued training loss decrease.

random weight initialization.

The low variance across tree-based methods (<0.5%) confirms that our results are robust and not artifacts of lucky random seeds.

4.8. Kaggle Leaderboard Performance

Our best model (RFXGB ensemble, 80.34% validation accuracy) achieves **79.67% accuracy on the Kaggle public leaderboard**, ranking in the top 8% of submissions. This strong generalization to the held-out test set validates our cross-validation strategy and confirms that the model does not overfit to the training distribution.

Leaderboard Submission Summary:

- **Random Forest (RF features):** 79.67% public score
- **XGBoost (XGB features):** 78.95% public score
- **RFXGB Ensemble:** 79.67% public score (tied with RF)
- **Top-3 Ensemble:** 78.47% public score

Interestingly, the ensemble does not outperform the single Random Forest model on the public test set, suggesting that the additional complexity may not generalize as well to unseen data. This highlights a common challenge in ensemble methods: validation set performance does not always figures/feature_importance_rf.png

Figure 14. Top 10 feature importance scores from Random Forest (RF pipeline). Title, Sex, and Fare dominate predictions, while Family_Size and Age provide additional discriminative power.

figures/seed_stability_boxplot.png

Figure 15. Validation accuracy distributions across three random seeds for top-performing models. Random Forest exhibits the lowest variance (std=0.42%), indicating high stability.

translate to test set improvements.

4.9. Comparison with Related Work

Table **??** compares our results with notable Kaggle kernels and published approaches.

Our approach achieves competitive performance comparable to established community solutions, confirming the effectiveness of our feature engineering and model selection strategy. The gap to top-tier submissions (82-84%) can potentially be closed through more sophisticated ensemble techniques (stacking, blending) and hyperparameter optimization via Bayesian methods.

Table 2. Comparison with related work on Titanic survival prediction.

Approach	**Method**	**Accuracy (%)**
Our Work	Random Forest + RF Features	**79.67**
Our Work	RFXGB Ensemble	**80.34** (val)
Manav Sehgal (2016)	Random Forest	78.47
Megan Risdal (2016)	Ensemble (RF + SVM)	79.43
Ahmed Besbes (2018)	Stacking (5 models)	80.86
Current Kaggle Top 1%	Advanced Ensembles	82-84

4.10. Summary of Key Findings

1. **Feature engineering dominates algorithm selection:** The choice of features (RF vs. XGB vs. MLP) impacts accuracy by 2-3%, while algorithm choice within the same feature set varies by 1-2%.
2. **Random Forest with community-driven features is the most robust single model:** Achieving 79.67% validation accuracy with low variance across seeds.
3. **Title extraction is the most important engineered feature:** Contributing 22.4% of Random Forest's predictive power.
4. **Ensemble methods provide marginal gains:** RFXGB ensemble improves validation accuracy by 0.67% but does not outperform single models on the Kaggle public test set.
5. **Tree-based methods excel on small tabular datasets:** Outperforming neural networks and kernel methods without requiring extensive hyperparameter tuning.

These findings underscore the importance of domain-informed feature engineering and careful model selection in applied machine learning, particularly for small-to-medium tabular datasets where deep learning offers limited advantages.

References