🎯 Bucket 1 Prediction Optimization Tricks

Problem Statement

Bucket 1 brands (high-erosion) are rare in the dataset:

- **Bucket 1**: ~130 brands (6.7%) - **Counts 2× in scoring!**
- **Bucket 2**: ~1,823 brands (93.3%)

Since Bucket 1 predictions are weighted double in the PE score, improving Bucket 1 accuracy has outsized impact on competition ranking.

Strategy 1: Sample Weighting ⭐

Give Bucket 1 samples higher weight during training so the model pays more attention to them.

```
import numpy as np

# Simple approach: 2× weight for Bucket 1
sample_weights = np.where(df['bucket'] == 1, 2.0, 1.0)

# More aggressive: 3-5× weight
sample_weights = np.where(df['bucket'] == 1, 4.0, 1.0)

# Apply during training
model.fit(X_train, y_train, sample_weight=sample_weights)

# For LightGBM
lgbm_model = lgb.LGBMRegressor(...)
lgbm_model.fit(X_train, y_train, sample_weight=sample_weights)

# For XGBoost
xgb_model = xgb.XGBRegressor(...)
xgb_model.fit(X_train, y_train, sample_weight=sample_weights)
```

Combined with Time-Window Weighting

```
def compute_bucket_time_weights(df, bucket_weight=2.0, early_weight=2.5):
    """Combine bucket weighting with time-window weighting."""
    weights = np.ones(len(df))

    # Bucket 1 gets higher weight
    weights[df['bucket'] == 1] *= bucket_weight

    # Early months (0-5) get higher weight for Scenario 1
    weights[df['months_postgx'] <= 5] *= early_weight

    return weights
```

Strategy 2: Oversampling Bucket 1

Simple Duplication

```
import pandas as pd

# Duplicate Bucket 1 samples 2-3 times
bucket1_data = train_df[train_df['bucket'] == 1]
bucket2_data = train_df[train_df['bucket'] == 2]

# Create augmented dataset (3× bucket1)
train_df_augmented = pd.concat([
    bucket2_data,
    bucket1_data,
    bucket1_data,
    bucket1_data
], ignore_index=True)

# Shuffle
train_df_augmented = train_df_augmented.sample(frac=1,
random_state=42).reset_index(drop=True)
```

SMOTE (Synthetic Minority Oversampling)

```
from imblearn.over_sampling import SMOTE

# For regression, we can use SMOTE on features
# Note: This works better for classification, but can help balance feature
space

# Create synthetic Bucket 1 samples
smote = SMOTE(sampling_strategy=0.5, random_state=42)  # Increase bucket1
ratio
X_resampled, bucket_resampled = smote.fit_resample(X, df['bucket'])

# Alternative: Use SMOTE-like approach for continuous targets
from imblearn.over_sampling import ADASYN
adasyn = ADASYN(sampling_strategy=0.5, random_state=42)
X_resampled, bucket_resampled = adasyn.fit_resample(X, df['bucket'])
```

Custom Oversampling with Noise

```python
def oversample_bucket1_with_noise(df, X, y, oversample_ratio=2,
noise_std=0.01):
    """Oversample Bucket 1 with small random noise for diversity."""
    bucket1_mask = df['bucket'] == 1
    X_bucket1 = X[bucket1_mask]
    y_bucket1 = y[bucket1_mask]

    # Create noisy copies
    X_synthetic = []
    y_synthetic = []

    for _ in range(oversample_ratio):
        noise = np.random.normal(0, noise_std, X_bucket1.shape)
        X_synthetic.append(X_bucket1 + noise)
        y_synthetic.append(y_bucket1)

    # Combine
    X_augmented = np.vstack([X] + X_synthetic)
    y_augmented = np.concatenate([y] + y_synthetic)

    return X_augmented, y_augmented
```

Strategy 3: Separate Models per Bucket ★ ★ (Highly Recommended)

Train specialized models for each bucket since their erosion patterns differ significantly.

```python
import lightgbm as lgb
import numpy as np

class BucketSpecificModel:
    """Train separate models for Bucket 1 and Bucket 2."""

    def __init__(self, model_params=None):
        self.model_params = model_params or {
            'objective': 'regression',
            'metric': 'mae',
            'verbosity': -1,
            'n_estimators': 500,
            'learning_rate': 0.05
        }
        self.model_bucket1 = None
        self.model_bucket2 = None

    def fit(self, X, y, bucket, sample_weight=None):
        """Fit separate models for each bucket."""
        bucket1_mask = bucket == 1
        bucket2_mask = bucket == 2

        # Bucket 1 model (potentially with different hyperparameters)
        bucket1_params = self.model_params.copy()
        bucket1_params['n_estimators'] = 300  # Fewer trees for smaller dataset
        bucket1_params['min_child_samples'] = 5  # Allow smaller leaves

        self.model_bucket1 = lgb.LGBMRegressor(**bucket1_params)
        self.model_bucket1.fit(
            X[bucket1_mask],
            y[bucket1_mask],
            sample_weight=sample_weight[bucket1_mask] if sample_weight is not None else None
        )

        # Bucket 2 model
        self.model_bucket2 = lgb.LGBMRegressor(**self.model_params)
        self.model_bucket2.fit(
            X[bucket2_mask],
            y[bucket2_mask],
            sample_weight=sample_weight[bucket2_mask] if sample_weight is not None else None
        )

        return self

    def predict(self, X, bucket):
        """Route predictions by bucket."""
        predictions = np.zeros(len(X))

        bucket1_mask = bucket == 1
        bucket2_mask = bucket == 2

        if bucket1_mask.any():
            predictions[bucket1_mask] = self.model_bucket1.predict(X[bucket1_mask])
        if bucket2_mask.any():
            predictions[bucket2_mask] = self.model_bucket2.predict(X[bucket2_mask])

        return predictions


# Usage
bucket_model = BucketSpecificModel()
bucket_model.fit(X_train, y_train, train_df['bucket'].values)
predictions = bucket_model.predict(X_test, test_df['bucket'].values)
```

Ensemble: Global + Bucket-Specific

```
class HybridBucketModel:
    """Combine global model with bucket-specific models."""

    def __init__(self, global_weight=0.3, bucket_weight=0.7):
        self.global_weight = global_weight
        self.bucket_weight = bucket_weight
        self.global_model = lgb.LGBMRegressor()
        self.bucket_model = BucketSpecificModel()

    def fit(self, X, y, bucket):
        self.global_model.fit(X, y)
        self.bucket_model.fit(X, y, bucket)
        return self

    def predict(self, X, bucket):
        global_pred = self.global_model.predict(X)
        bucket_pred = self.bucket_model.predict(X, bucket)

        # Weighted ensemble
        return self.global_weight * global_pred + self.bucket_weight *
bucket_pred
```

Strategy 4: Bucket-Aware Loss Function

Custom loss functions that penalize Bucket 1 errors more heavily.

For LightGBM

```
import numpy as np

def bucket_weighted_mse_objective(y_true, y_pred, bucket, bucket1_weight=2.0):
    """Custom objective that weights Bucket 1 errors more."""
    weights = np.where(bucket == 1, bucket1_weight, 1.0)

    # Gradient
    grad = weights * (y_pred - y_true)

    # Hessian (second derivative)
    hess = weights * np.ones_like(y_true)

    return grad, hess


def bucket_weighted_mae_objective(y_true, y_pred, bucket, bucket1_weight=2.0):
    """Custom MAE objective with bucket weighting."""
    weights = np.where(bucket == 1, bucket1_weight, 1.0)
    # Gradient of MAE
    grad = weights * np.sign(y_pred - y_true)

    # Hessian (constant for MAE)
    hess = weights * np.ones_like(y_true)

    return grad, hess


# Usage with LightGBM
def create_bucket_objective(bucket_array, weight=2.0):
    """Factory function to create objective with bucket info."""
    def objective(y_true, y_pred):
        return bucket_weighted_mse_objective(y_true, y_pred, bucket_array,
weight)
    return objective

# In training
bucket_objective = create_bucket_objective(train_df['bucket'].values,
weight=3.0)
model = lgb.LGBMRegressor(objective=bucket_objective)
```

Custom Evaluation Metric

```
def bucket_weighted_pe(y_true, y_pred, bucket):
    """Calculate PE score with proper bucket weighting (matches
competition)."""
    bucket1_mask = bucket == 1
    bucket2_mask = bucket == 2

    # Bucket 1 PE (weighted 2×)
    if bucket1_mask.sum() > 0:
        bucket1_pe = np.mean(np.abs(y_true[bucket1_mask] -
y_pred[bucket1_mask]))
    else:
        bucket1_pe = 0

    # Bucket 2 PE
    if bucket2_mask.sum() > 0:
        bucket2_pe = np.mean(np.abs(y_true[bucket2_mask] -
y_pred[bucket2_mask]))
    else:
        bucket2_pe = 0

    # Weighted combination (Bucket 1 counts 2×)
    total_weight = 2 * bucket1_mask.sum() + bucket2_mask.sum()
    weighted_pe = (2 * bucket1_mask.sum() * bucket1_pe + bucket2_mask.sum() *
bucket2_pe) / total_weight

    return weighted_pe
```

Strategy 5: Stratified Cross-Validation ★

Ensure Bucket 1 brands appear proportionally in every CV fold.

```
from sklearn.model_selection import StratifiedGroupKFold, GroupKFold
import numpy as np

def stratified_bucket_cv(df, n_splits=5, random_state=42):
    """
    Create CV folds that:
    1. Keep all months of a brand together (GroupKFold)
    2. Ensure proportional Bucket 1 representation (Stratified)
    """
    # Get unique brands with their bucket
    brand_bucket = df.groupby('brand_id')['bucket'].first().reset_index()

    cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,
random_state=random_state)

    # Generate fold indices at brand level
    brand_folds = list(cv.split(
        brand_bucket['brand_id'],
        brand_bucket['bucket'],
        groups=brand_bucket['brand_id']
    ))

    # Map back to full dataset
    for train_brands_idx, val_brands_idx in brand_folds:
        train_brands = brand_bucket.iloc[train_brands_idx]['brand_id'].values
        val_brands = brand_bucket.iloc[val_brands_idx]['brand_id'].values

        train_idx = df[df['brand_id'].isin(train_brands)].index
        val_idx = df[df['brand_id'].isin(val_brands)].index

        yield train_idx.values, val_idx.values


# Usage
for fold, (train_idx, val_idx) in enumerate(stratified_bucket_cv(df)):
    X_train, X_val = X[train_idx], X[val_idx]
    y_train, y_val = y[train_idx], y[val_idx]

    # Check bucket distribution
    train_bucket1_pct = (df.iloc[train_idx]['bucket'] == 1).mean()
    val_bucket1_pct = (df.iloc[val_idx]['bucket'] == 1).mean()
    print(f"Fold {fold}: Train B1={train_bucket1_pct:.1%}, Val B1=
{val_bucket1_pct:.1%}")
```

Simple Stratified GroupKFold

```python
from sklearn.model_selection import StratifiedGroupKFold

# Stratify by bucket, group by brand
cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=42)

for train_idx, val_idx in cv.split(X, df['bucket'], groups=df['brand_id']):
    # Each fold has proportional Bucket 1 representation
    # All months of each brand stay together
    model.fit(X[train_idx], y[train_idx])
    score = model.score(X[val_idx], y[val_idx])
```

Strategy 6: Feature Engineering for High-Erosion Patterns

Bucket 1 brands erode faster - create features that capture this.

```python
def create_bucket1_focused_features(df):
    """Features specifically designed to capture Bucket 1 (high-erosion)
patterns."""

    # Early erosion features (Bucket 1 drops fast in first months)
    df['erosion_rate_month1_3'] = df.groupby('brand_id').apply(
        lambda x: (x[x['months_postgx'] <= 3]['vol_norm'].iloc[-1] - 1.0) / 3
        if len(x[x['months_postgx'] <= 3]) > 0 else 0
    ).reindex(df.index)

    # Maximum drop in first 6 months
    df['max_drop_first_6_months'] = df.groupby('brand_id').apply(
        lambda x: 1.0 - x[x['months_postgx'] <= 6]['vol_norm'].min()
        if len(x[x['months_postgx'] <= 6]) > 0 else 0
    ).reindex(df.index)

    # Speed to 50% erosion
    df['months_to_50pct'] = df.groupby('brand_id').apply(
        lambda x: x[x['vol_norm'] <= 0.5]['months_postgx'].min()
        if (x['vol_norm'] <= 0.5).any() else 999
    ).reindex(df.index)

    # Volatility in early period (Bucket 1 may be more volatile)
    df['early_volatility'] = df.groupby('brand_id').apply(
        lambda x: x[x['months_postgx'] <= 6]['vol_norm'].std()
        if len(x[x['months_postgx'] <= 6]) > 1 else 0
    ).reindex(df.index)

    # Competition impact (Bucket 1 may respond more to generics)
    df['erosion_per_generic'] = df['vol_norm'] / (df['n_gxs'] + 1)
    return df


def create_bucket_interaction_features(df):
    """Interaction features with bucket."""

    # Bucket × time interactions
    df['bucket_x_months'] = df['bucket'] * df['months_postgx']
    df['bucket_x_n_gxs'] = df['bucket'] * df['n_gxs']
    df['bucket_x_avg_vol'] = df['bucket'] * df['avg_vol']

    # Bucket 1 indicator interactions
    df['is_bucket1'] = (df['bucket'] == 1).astype(int)
    df['bucket1_x_early'] = df['is_bucket1'] * (df['months_postgx'] <=
6).astype(int)
    df['bucket1_x_competition'] = df['is_bucket1'] * df['n_gxs']

    return df
```

🚀 Recommended Implementation Order

Quick Wins (Implement First)

1. **Sample Weighting** - Easy to add, immediate impact
2. **Stratified GroupKFold** - Better CV estimates

Medium Effort

3. **Bucket-Focused Features** - Add erosion speed features
4. **Custom Evaluation Metric** - Track true competition score

Advanced (If Time Permits)

5. **Separate Bucket Models** - Highest potential gain
6. **Custom Loss Function** - Fine-tuned optimization

Example: Complete Pipeline

```python
import numpy as np
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import StratifiedGroupKFold

def train_bucket_optimized_model(df, X, y, n_splits=5):
    """Complete pipeline with Bucket 1 optimization."""
    # 1. Create sample weights (Bucket 1 = 3×)
    sample_weights = np.where(df['bucket'] == 1, 3.0, 1.0)

    # 2. Stratified GroupKFold
    cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,
random_state=42)

    # 3. Model with tuned parameters
    model_params = {
        'objective': 'regression',
        'metric': 'mae',
        'n_estimators': 500,
        'learning_rate': 0.05,
        'num_leaves': 31,
        'min_child_samples': 10,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'verbosity': -1
    }

    oof_predictions = np.zeros(len(df))
    models = []

    for fold, (train_idx, val_idx) in enumerate(cv.split(X, df['bucket'],
groups=df['brand_id'])):
        X_train, X_val = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]
        weights_train = sample_weights[train_idx]

        model = lgb.LGBMRegressor(**model_params)
        model.fit(
            X_train, y_train,
            sample_weight=weights_train,
            eval_set=[(X_val, y_val)],
            callbacks=[lgb.early_stopping(50, verbose=False)]
        )

        oof_predictions[val_idx] = model.predict(X_val)
        models.append(model)

        # Report per-bucket performance
        val_bucket = df.iloc[val_idx]['bucket'].values
        b1_mae = np.mean(np.abs(y_val[val_bucket == 1] -
oof_predictions[val_idx][val_bucket == 1]))
        b2_mae = np.mean(np.abs(y_val[val_bucket == 2] -
oof_predictions[val_idx][val_bucket == 2]))
        print(f"Fold {fold}: Bucket1 MAE = {b1_mae:.4f}, Bucket2 MAE =
{b2_mae:.4f}")

    return models, oof_predictions
```

References

- Competition PE Score weights Bucket 1 at 2×
- Bucket 1 = High erosion brands (drop to <50% within 6 months)
- ~130 Bucket 1 brands vs ~1,823 Bucket 2 brands in training data