# Supervised Learning - Intermediate Handout

Machine Learning for Smarter Innovation

## 1 Supervised Learning - Intermediate Handout

**Target Audience**: Practitioners with Python knowledge **Duration**: 60 minutes reading + coding **Level**: Intermediate (implementation focused)

---

### 1.1 Setup and Environment

Before diving into supervised learning implementations, ensure you have the necessary libraries installed. All examples use scikit-learn, the standard Python library for machine learning.

```
# Core libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Scikit-learn essentials
from sklearn.model_selection import train_test_split, cross_val_score,
    GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             mean_squared_error, r2_score, mean_absolute_error)

# Models
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge,
    Lasso
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier,
    GradientBoostingRegressor
from sklearn.svm import SVC, SVR
from sklearn.neighbors import KNeighborsClassifier

# Set random seed for reproducibility
np.random.seed(42)
```

---

## 1.2 1. Data Preparation Pipeline

### 1.2.1 Concept Overview

Before training any model, data must be properly prepared. This includes handling missing values, encoding categorical variables, scaling numerical features, and splitting into train/test sets. Poor data preparation is the most common cause of model failures.

### 1.2.2 Implementation

```
def prepare_data(df, target_column, test_size=0.2, scale=True):
    """
    Complete data preparation pipeline.

    Parameters:
    -----------
    df : pandas DataFrame
        Raw data with features and target
    target_column : str
        Name of the target column
    test_size : float
        Fraction of data for testing
    scale : bool
        Whether to apply standard scaling

    Returns:
    --------
    X_train, X_test, y_train, y_test, scaler (or None)
    """
    # Separate features and target
    X = df.drop(columns=[target_column])
    y = df[target_column]

    # Handle missing values
    # Numerical: fill with median
    numerical_cols = X.select_dtypes(include=[np.number]).columns
    X[numerical_cols] = X[numerical_cols].fillna(X[numerical_cols].median())

    # Categorical: fill with mode
    categorical_cols = X.select_dtypes(include=['object', 'category']).columns
    for col in categorical_cols:
        X[col] = X[col].fillna(X[col].mode()[0])

    # Encode categorical variables
    label_encoders = {}
    for col in categorical_cols:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))
        label_encoders[col] = le

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42, stratify=y if y.nunique()
    < 20 else None
    )

    # Scale features
    scaler = None
    if scale:
        scaler = StandardScaler()
        X_train = pd.DataFrame(scaler.fit_transform(X_train),
                               columns=X_train.columns, index=X_train.index)
```

```
        X_test = pd.DataFrame(scaler.transform(X_test),
                              columns=X_test.columns, index=X_test.index)

    return X_train, X_test, y_train, y_test, scaler
```

### 1.2.3 Usage Example

```
# Load sample data
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Prepare data
X_train, X_test, y_train, y_test, scaler = prepare_data(df, 'target')

print(f"Training samples: {len(X_train)}")
print(f"Test samples: {len(X_test)}")
print(f"Features: {X_train.shape[1]}")
```

---

## 1.3 2. Classification Models

### 1.3.1 Concept Overview

Classification predicts categorical outcomes. The choice of algorithm depends on data size, feature types, interpretability requirements, and accuracy needs. Start simple with Logistic Regression, then try ensemble methods if accuracy is insufficient.

### 1.3.2 Logistic Regression Implementation

```
def train_logistic_regression(X_train, y_train, X_test, y_test):
    """
    Train and evaluate logistic regression classifier.
    """
    model = LogisticRegression(max_iter=1000, random_state=42)
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Evaluation
    print("Logistic Regression Results:")
    print(f"  Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"  Recall:    {recall_score(y_test, y_pred):.4f}")
    print(f"  F1 Score:  {f1_score(y_test, y_pred):.4f}")

    # Feature importance (coefficients)
    if hasattr(model, 'coef_'):
        importance = pd.DataFrame({
            'feature': X_train.columns,
            'coefficient': model.coef_[0]
        }).sort_values('coefficient', key=abs, ascending=False)
```

```
        print("\nTop 5 features by coefficient magnitude:")
        print(importance.head())

    return model, y_pred, y_prob
```

### 1.3.3 Random Forest Implementation

```
def train_random_forest(X_train, y_train, X_test, y_test, n_estimators=100):
    """
    Train and evaluate random forest classifier.
    """
    model = RandomForestClassifier(
        n_estimators=n_estimators,
        max_depth=10,
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42,
        n_jobs=-1
    )
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Evaluation
    print("Random Forest Results:")
    print(f"  Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"  Recall:    {recall_score(y_test, y_pred):.4f}")
    print(f"  F1 Score:  {f1_score(y_test, y_pred):.4f}")

    # Feature importance
    importance = pd.DataFrame({
        'feature': X_train.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)
    print("\nTop 5 features by importance:")
    print(importance.head())

    return model, y_pred, y_prob
```

### 1.3.4 Gradient Boosting Implementation

```
def train_gradient_boosting(X_train, y_train, X_test, y_test):
    """
    Train and evaluate gradient boosting classifier.
    """
    model = GradientBoostingClassifier(
        n_estimators=100,
        learning_rate=0.1,
        max_depth=5,
        min_samples_split=5,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Predictions
```

```
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Evaluation
    print("Gradient Boosting Results:")
    print(f"  Accuracy:  {accuracy_score(y_test, y_pred):.4f}")
    print(f"  Precision: {precision_score(y_test, y_pred):.4f}")
    print(f"  Recall:    {recall_score(y_test, y_pred):.4f}")
    print(f"  F1 Score:  {f1_score(y_test, y_pred):.4f}")

    return model, y_pred, y_prob
```

### 1.3.5 Model Comparison

```
def compare_classifiers(X_train, y_train, X_test, y_test):
    """
    Compare multiple classifiers on the same data.
    """
    models = {
        'Logistic Regression': LogisticRegression(max_iter=1000, random_state
    =42),
        'Decision Tree': DecisionTreeClassifier(max_depth=10, random_state=42)
    ,
        'Random Forest': RandomForestClassifier(n_estimators=100, random_state
    =42),
        'Gradient Boosting': GradientBoostingClassifier(n_estimators=100,
    random_state=42),
        'SVM': SVC(kernel='rbf', probability=True, random_state=42),
        'KNN': KNeighborsClassifier(n_neighbors=5)
    }

    results = []
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results.append({
            'Model': name,
            'Accuracy': accuracy_score(y_test, y_pred),
            'Precision': precision_score(y_test, y_pred),
            'Recall': recall_score(y_test, y_pred),
            'F1': f1_score(y_test, y_pred)
        })

    results_df = pd.DataFrame(results).sort_values('F1', ascending=False)
    print("\nClassifier Comparison:")
    print(results_df.to_string(index=False))

    return results_df
```

---

## 1.4 3. Regression Models

### 1.4.1 Concept Overview

Regression predicts continuous numerical values. Unlike classification, regression outputs can be any number within a range. Evaluation uses metrics like RMSE (Root Mean Squared Error) and R-squared

that measure prediction accuracy differently than classification metrics.

### 1.4.2 Linear Regression Implementation

```
def train_linear_regression(X_train, y_train, X_test, y_test):
    """
    Train and evaluate linear regression model.
    """
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Evaluation
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print("Linear Regression Results:")
    print(f"  RMSE: {rmse:.4f}")
    print(f"  MAE:  {mae:.4f}")
    print(f"  R2:   {r2:.4f}")

    # Coefficients
    coef_df = pd.DataFrame({
        'feature': X_train.columns,
        'coefficient': model.coef_
    }).sort_values('coefficient', key=abs, ascending=False)
    print("\nTop 5 features by coefficient magnitude:")
    print(coef_df.head())

    return model, y_pred
```

### 1.4.3 Regularized Regression (Ridge and Lasso)

```
def train_regularized_regression(X_train, y_train, X_test, y_test):
    """
    Compare Ridge and Lasso regression.
    """
    results = []

    # Ridge Regression
    ridge = Ridge(alpha=1.0)
    ridge.fit(X_train, y_train)
    y_pred_ridge = ridge.predict(X_test)

    results.append({
        'Model': 'Ridge',
        'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_ridge)),
        'R2': r2_score(y_test, y_pred_ridge),
        'Non-zero coefficients': np.sum(ridge.coef_ != 0)
    })

    # Lasso Regression
    lasso = Lasso(alpha=0.1)
    lasso.fit(X_train, y_train)
    y_pred_lasso = lasso.predict(X_test)

    results.append({
```

```
        'Model': 'Lasso',
        'RMSE': np.sqrt(mean_squared_error(y_test, y_pred_lasso)),
        'R2': r2_score(y_test, y_pred_lasso),
        'Non-zero coefficients': np.sum(lasso.coef_ != 0)
    })

    print("\nRegularized Regression Comparison:")
    print(pd.DataFrame(results).to_string(index=False))

    return ridge, lasso
```

### 1.4.4 Ensemble Regression

```
def train_ensemble_regression(X_train, y_train, X_test, y_test):
    """
    Train Random Forest and Gradient Boosting regressors.
    """
    models = {
        'Random Forest': RandomForestRegressor(n_estimators=100, random_state
    =42),
        'Gradient Boosting': GradientBoostingRegressor(n_estimators=100,
    random_state=42)
    }

    results = []
    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results.append({
            'Model': name,
            'RMSE': np.sqrt(mean_squared_error(y_test, y_pred)),
            'MAE': mean_absolute_error(y_test, y_pred),
            'R2': r2_score(y_test, y_pred)
        })

    print("\nEnsemble Regression Comparison:")
    print(pd.DataFrame(results).to_string(index=False))

    return models
```

---

## 1.5 4. Cross-Validation

### 1.5.1 Concept Overview

Cross-validation provides more reliable performance estimates than a single train-test split. It repeatedly trains and tests the model on different subsets of data, then averages the results. This reveals how stable model performance is across different data samples.

### 1.5.2 Implementation

```
def cross_validate_model(model, X, y, cv=5, scoring='accuracy'):
    """
    Perform k-fold cross-validation.
```

```
    Parameters:
    -----------
    model : sklearn estimator
        Model to evaluate
    X : array-like
        Features
    y : array-like
        Target
    cv : int
        Number of folds
    scoring : str
        Scoring metric ('accuracy', 'f1', 'roc_auc', 'neg_mean_squared_error')
    """
    scores = cross_val_score(model, X, y, cv=cv, scoring=scoring)

    print(f"Cross-Validation Results ({cv} folds):")
    print(f"  Scores: {scores}")
    print(f"  Mean:   {scores.mean():.4f}")
    print(f"  Std:    {scores.std():.4f}")
    print(f"  95% CI: [{scores.mean() - 1.96*scores.std():.4f}, "
          f"{scores.mean() + 1.96*scores.std():.4f}]")

    return scores


# Example usage
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
X, y = data.data, data.target

model = RandomForestClassifier(n_estimators=100, random_state=42)
scores = cross_validate_model(model, X, y, cv=5, scoring='accuracy')
```

---

## 1.6 5. Hyperparameter Tuning

### 1.6.1 Concept Overview

Hyperparameters are model settings you choose before training (like number of trees in a forest). Tuning finds the best settings for your specific data. Grid search tries all combinations; random search samples combinations and is faster for many hyperparameters.

### 1.6.2 Grid Search Implementation

```
def tune_random_forest(X_train, y_train, X_test, y_test):
    """
    Tune Random Forest hyperparameters using grid search.
    """
    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [5, 10, 20, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }

    rf = RandomForestClassifier(random_state=42)
```

```
    grid_search = GridSearchCV(
        rf, param_grid, cv=5, scoring='f1',
        n_jobs=-1, verbose=1
    )
    grid_search.fit(X_train, y_train)

    print(f"\nBest parameters: {grid_search.best_params_}")
    print(f"Best CV F1 score: {grid_search.best_score_:.4f}")

    # Evaluate on test set
    y_pred = grid_search.predict(X_test)
    print(f"Test F1 score: {f1_score(y_test, y_pred):.4f}")

    return grid_search.best_estimator_
```

### 1.6.3 Random Search Implementation

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

def tune_gradient_boosting(X_train, y_train, X_test, y_test, n_iter=50):
    """
    Tune Gradient Boosting using randomized search.
    """
    param_distributions = {
        'n_estimators': randint(50, 300),
        'learning_rate': uniform(0.01, 0.3),
        'max_depth': randint(3, 15),
        'min_samples_split': randint(2, 20),
        'min_samples_leaf': randint(1, 10)
    }

    gb = GradientBoostingClassifier(random_state=42)

    random_search = RandomizedSearchCV(
        gb, param_distributions, n_iter=n_iter, cv=5,
        scoring='f1', n_jobs=-1, random_state=42, verbose=1
    )
    random_search.fit(X_train, y_train)

    print(f"\nBest parameters: {random_search.best_params_}")
    print(f"Best CV F1 score: {random_search.best_score_:.4f}")

    # Evaluate on test set
    y_pred = random_search.predict(X_test)
    print(f"Test F1 score: {f1_score(y_test, y_pred):.4f}")

    return random_search.best_estimator_
```

---

## 1.7 6. Model Evaluation and Visualization

### 1.7.1 Confusion Matrix

```
def plot_confusion_matrix(y_true, y_pred, labels=None):
    """
    Plot confusion matrix as heatmap.
```

```
    """
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels or ['Negative', 'Positive'],
                yticklabels=labels or ['Negative', 'Positive'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.tight_layout()
    plt.show()

    # Print classification report
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=labels))
```

### 1.7.2 ROC Curve

```
from sklearn.metrics import roc_curve, auc

def plot_roc_curve(y_true, y_prob):
    """
    Plot ROC curve with AUC score.
    """
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2,
             label=f'ROC curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--',
             label='Random classifier')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    return roc_auc
```

### 1.7.3 Feature Importance Plot

```
def plot_feature_importance(model, feature_names, top_n=15):
    """
    Plot feature importance for tree-based models.
    """
    if hasattr(model, 'feature_importances_'):
        importance = model.feature_importances_
    elif hasattr(model, 'coef_'):
        importance = np.abs(model.coef_[0]) if len(model.coef_.shape) > 1 else
     np.abs(model.coef_)
    else:
        print("Model does not have feature importance attribute")
```

```
        return

    # Create DataFrame and sort
    importance_df = pd.DataFrame({
        'feature': feature_names,
        'importance': importance
    }).sort_values('importance', ascending=True).tail(top_n)

    plt.figure(figsize=(10, 8))
    plt.barh(importance_df['feature'], importance_df['importance'], color='
    steelblue')
    plt.xlabel('Importance')
    plt.title(f'Top {top_n} Feature Importances')
    plt.tight_layout()
    plt.show()
```

---

## 1.8 7. Complete Pipeline Example

### 1.8.1 End-to-End Classification

```
def complete_classification_pipeline(df, target_column):
    """
    Complete supervised learning classification pipeline.
    """
    print("=" * 60)
    print("SUPERVISED LEARNING CLASSIFICATION PIPELINE")
    print("=" * 60)

    # Step 1: Prepare data
    print("\n[1] Preparing data...")
    X_train, X_test, y_train, y_test, scaler = prepare_data(df, target_column)
    print(f"    Training: {len(X_train)} samples")
    print(f"    Testing:  {len(X_test)} samples")
    print(f"    Features: {X_train.shape[1]}")

    # Step 2: Compare baseline models
    print("\n[2] Comparing baseline models...")
    results = compare_classifiers(X_train, y_train, X_test, y_test)

    # Step 3: Select best model for tuning
    best_model_name = results.iloc[0]['Model']
    print(f"\n[3] Best baseline model: {best_model_name}")

    # Step 4: Hyperparameter tuning
    print("\n[4] Tuning hyperparameters...")
    if best_model_name == 'Random Forest':
        best_model = tune_random_forest(X_train, y_train, X_test, y_test)
    elif best_model_name == 'Gradient Boosting':
        best_model = tune_gradient_boosting(X_train, y_train, X_test, y_test)
    else:
        # Default to Random Forest for tuning
        best_model = tune_random_forest(X_train, y_train, X_test, y_test)

    # Step 5: Final evaluation
    print("\n[5] Final evaluation...")
    y_pred = best_model.predict(X_test)
    y_prob = best_model.predict_proba(X_test)[:, 1]
```

```
    print("\n" + "=" * 60)
    print("FINAL MODEL PERFORMANCE")
    print("=" * 60)
    print(classification_report(y_test, y_pred))

    # Step 6: Visualizations
    print("\n[6] Generating visualizations...")
    plot_confusion_matrix(y_test, y_pred)
    plot_roc_curve(y_test, y_prob)
    plot_feature_importance(best_model, X_train.columns)

    return best_model, scaler


# Example usage
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

model, scaler = complete_classification_pipeline(df, 'target')
```

---

## 1.9 Common Hyperparameters

| | | | | |
|---|---|---|---|---|
| () * | () * | () * | () * | |
| 0.1 | 7 | 3 | 9 | Typical |
| | | | | 0.3659Range |
| () * | () * | () * | () * | |
| 0.1 | 7 | 3 | 9 | Mor |
| dom For- est | 500 | trees = bet- ter ac- cu- racy, slower train- ing | | |
| () * | () * | () * | () * | |
| 0.1 | 7 | 3 | 9 | _dep |
| dom For- est | 30 or more None | = more com- plex, risk over- fit- ting | | |

| | | | |
|---|---|---|---|
| () * | () * | () * | () * |
| 0.1 | 7 | 2 | Typical 0.3659 Range |
| () * | () * | () * | () * |
| 0.1 | 7 | 2 | 5 Higher |
| dom For- est | 20 | = | more reg- u- lar- iza- tion |
| () * | () * | () * | () * |
| 0.1 | 7 | 2 | 5 Lower |
| di- ing Boost- ing | 0.3 | late | slower learn- ing, bet- ter ac- cu- racy |
| () * | () * | () * | () * |
| 0.1 | 7 | 2 | 5 More |
| di- ent Boost- ing | 500 | = | bet- ter, with di- min- ish- ing re- turns |
| () * | () * | () * | () * |
| 0.1 | 7 | 2 | 5 Lower |
| gis- tic Re- gres- sion | 100 | = | more reg- u- lar- iza- tion |

| | | | |
|---|---|---|---|
| () * | () * | () * | () * |
| 0.1 | 7 | 7 | Typical 0.3659 Range |
| 0.1 | 7 | 7 | Higher 10 = less regularization |
| 0.1 | 7 | 7 | Higher 1 = more complex boundaries |
| 0.1 | 7 | 7 | Odd numbers, lower = more complex |

---

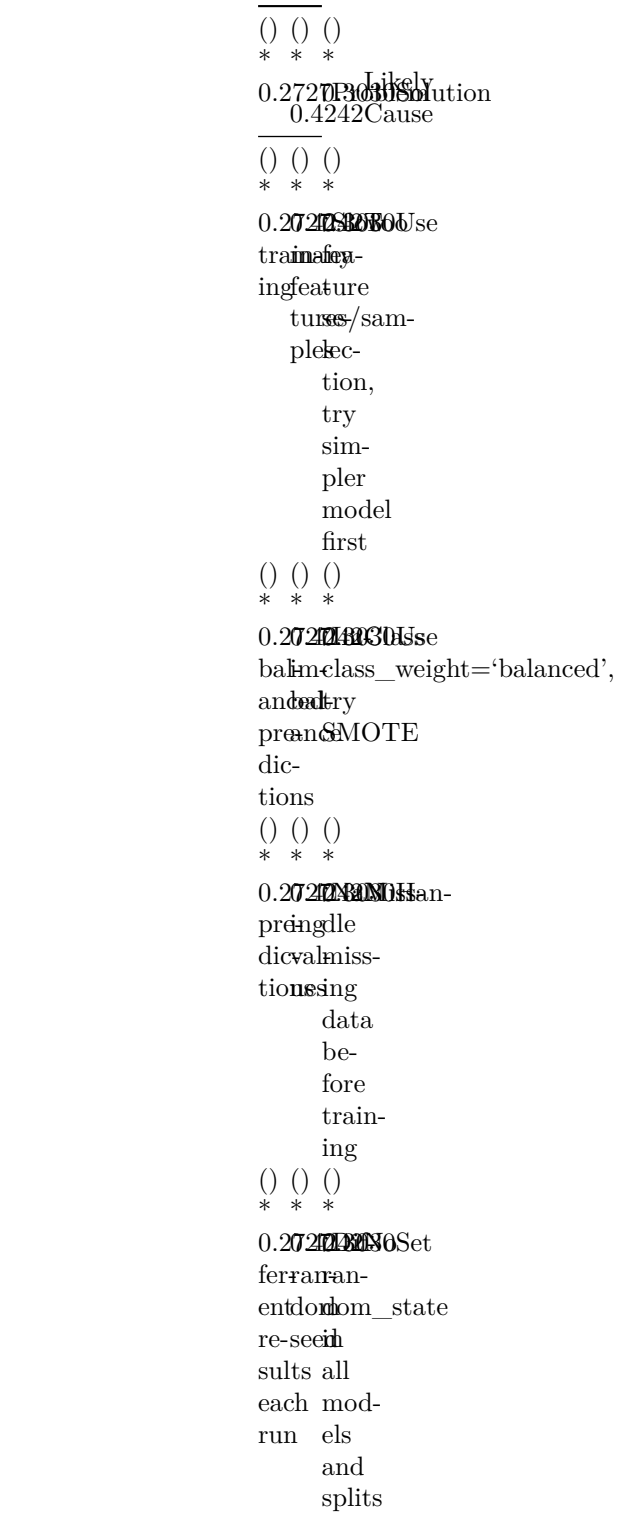
## 1.10 Practice Projects

1. **Customer Churn Prediction**: Load a telecom customer dataset, predict which customers will cancel service. Focus on recall to catch at-risk customers.
2. **House Price Prediction**: Use the California housing dataset to predict median house values. Experiment with feature engineering (polynomial features, interaction terms).
3. **Credit Card Fraud Detection**: Work with imbalanced data. Compare different sampling strategies (SMOTE, undersampling) and their effect on precision/recall tradeoffs.
4. **Multi-class Classification**: Use the iris or digits dataset to practice multi-class classification. Compare one-vs-rest versus one-vs-one approaches.
5. **Model Stacking**: Build an ensemble that combines predictions from multiple models using a meta-learner. Compare to individual models.

---

## 1.11 Troubleshooting

| | | |
|---|---|---|
| () * | () * | () * |
| 0.2727 | 0.3030 | Likely Cause |
| | 0.4242 | |
| () * | () * | () * |
| 0.2727 | 0.4848 | Try more complex model, add features |
| () * | () * | () * |
| 0.2727 | 0.4848 | Regularize, reduce complexity, get more data |
| () * | () * | () * |
| 0.2727 | 0.4848 | Increase max_iter, scale features |
| () * | () * | () * |
| 0.2727 | 0.4848 | Use smaller batch size, reduce features, use incremental learning |




## 1.12 Next Steps

Ready for the mathematical foundations? The advanced handout covers: - Derivation of loss functions and optimization algorithms - Bias-variance tradeoff mathematics - Regularization theory (L1 vs L2) - Ensemble theory and boosting algorithms - Learning theory and generalization bounds - Production deployment considerations

For practical next steps: - Try XGBoost and LightGBM for improved performance - Explore automated

machine learning (AutoML) tools - Learn about feature selection techniques - Study model interpretability with SHAP values

---

*Implementation is where theory meets reality. Start with simple models, establish baselines, then incrementally add complexity. The best practitioners spend more time understanding their data than tuning hyperparameters. Remember: a well-prepared dataset with a simple model often beats a sophisticated model on messy data.*