

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report)

# ML Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# Feature Selection
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.ensemble import RandomForestClassifier

import warnings
warnings.filterwarnings('ignore')

# Styling
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (14, 6)

print("=" * 80)
print("PROJECT 2: MACHINE LEARNING WITH GRID SEARCH & FEATURE SELECTION")
print("=" * 80)

# =====
# STEP 1: LOAD AND PREPARE DATA
# =====
print("\n[STEP 1] Loading Cleaned Dataset...")
df = pd.read_csv('cleaned_kidney_disease.csv')

print(f"✓ Dataset loaded successfully!")
print(f" - Shape: {df.shape[0]} rows, {df.shape[1]} columns")
print(f"\nFirst few rows:")
print(df.head())

print(f"\n✓ Missing values check: {df.isnull().sum().sum()} (should be 0)")

# =====
# STEP 2: DATA PREPROCESSING
# =====
print("\n" + "=" * 80)
print("STEP 2: DATA PREPROCESSING")
print("=" * 80)

print("\n[2.1] Encoding categorical variables...")

# Separate features and target
X = df.drop('classification', axis=1)
y = df['classification']

print(f"\nTarget variable distribution:")
print(y.value_counts())

# Encode target variable
le_target = LabelEncoder()
y_encoded = le_target.fit_transform(y)
print(f"\n✓ Target encoded: {le_target.classes_} → {np.unique(y_encoded)}")

# Encode categorical features

```

```

categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = X.select_dtypes(include=[np.number]).columns.tolist()

print(f"\nCategorical columns ({len(categorical_cols)}): {categorical_cols}")
print(f"Numerical columns ({len(numerical_cols)}): {numerical_cols}")

X_encoded = X.copy()
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    X_encoded[col] = le.fit_transform(X_encoded[col].astype(str))
    label_encoders[col] = le
    print(f"    ✓ Encoded {col}")

print(f"\n✓ All categorical variables encoded!")

# =====
# STEP 3: SPLIT DATA
# =====
print("\n[2.2] Splitting data into training and testing sets...")

X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

print(f"✓ Training set: {X_train.shape[0]} samples")
print(f"✓ Testing set: {X_test.shape[0]} samples")

# Scale features
print("\n[2.3] Scaling features...")
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print("✓ Features scaled using StandardScaler")

# =====
# STEP 3: MACHINE LEARNING WITH GRID SEARCH
# =====
print("\n" + "=" * 80)
print("STEP 3: APPLYING ML MODELS WITH GRID SEARCH OPTIMIZER")
print("=" * 80)

print("\nWhat is Grid Search?")
print(" - Automatically tests different hyperparameter combinations")
print(" - Finds the BEST parameters for each model")
print(" - Uses cross-validation to avoid overfitting")
print(" - Saves time compared to manual tuning")

# Define models and their parameter grids
models_params = {
    'Logistic Regression': {
        'model': LogisticRegression(max_iter=1000, random_state=42),
        'params': {
            'C': [0.1, 1, 10],
            'solver': ['liblinear', 'lbfgs']
        }
    },
    'Decision Tree': {
        'model': DecisionTreeClassifier(random_state=42),
        'params': {
            'max_depth': [3, 5, 10, None],
            'min_samples_split': [2, 5, 10],
            'criterion': ['gini', 'entropy']
        }
    },
    'Random Forest': {
        'model': RandomForestClassifier(random_state=42),
        'params': {

```

```

        'n_estimators': [50, 100, 200],
        'max_depth': [5, 10, None],
        'min_samples_split': [2, 5]
    }
},
'SVM': {
    'model': SVC(random_state=42),
    'params': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    }
},
'KNN': {
    'model': KNeighborsClassifier(),
    'params': {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan']
    }
}
}

# Store results
results = []
best_models = {}

print("\n[3.1] Training models with Grid Search...\n")

for model_name, mp in models_params.items():
    print(f"\n{'='*60}")
    print(f"Training: {model_name}")
    print(f"{'='*60}")

    # Grid Search
    grid_search = GridSearchCV(
        mp['model'],
        mp['params'],
        cv=5,
        scoring='accuracy',
        n_jobs=-1,
        verbose=0
    )

    grid_search.fit(X_train_scaled, y_train)

    # Best model
    best_model = grid_search.best_estimator_
    best_models[model_name] = best_model

    # Predictions
    y_pred = best_model.predict(X_test_scaled)

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    results.append({
        'Model': model_name,
        'Best Parameters': grid_search.best_params_,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    })

print(f"\n✓ Best Parameters: {grid_search.best_params_}")

```

```

print(f" Accuracy:  {accuracy:.4f}")
print(f" Precision: {precision:.4f}")
print(f" Recall:    {recall:.4f}")
print(f" F1-Score:   {f1:.4f}")

print("\n" + "=" * 80)
print("MODEL COMPARISON - ALL FEATURES")
print("=" * 80)

results_df = pd.DataFrame(results)
print("\n" + results_df.to_string(index=False))

# Find best model
best_idx = results_df['Accuracy'].idxmax()
best_model_name = results_df.loc[best_idx, 'Model']
best_accuracy = results_df.loc[best_idx, 'Accuracy']

print(f"\n🏆 BEST MODEL: {best_model_name} with Accuracy = {best_accuracy:.4f}")

# =====
# STEP 4: VISUALIZATION - MODEL COMPARISON
# =====
print("\n[3.2] Creating visualizations...")

# Model comparison plot
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Accuracy comparison
axes[0, 0].barh(results_df['Model'], results_df['Accuracy'], color='skyblue')
axes[0, 0].set_xlabel('Accuracy', fontweight='bold')
axes[0, 0].set_title('Model Accuracy Comparison (All Features)', fontweight='bold', fontsize=12)
axes[0, 0].set_xlim([0, 1])
for i, v in enumerate(results_df['Accuracy']):
    axes[0, 0].text(v + 0.01, i, f'{v:.4f}', va='center')

# Precision comparison
axes[0, 1].barh(results_df['Model'], results_df['Precision'], color='lightcoral')
axes[0, 1].set_xlabel('Precision', fontweight='bold')
axes[0, 1].set_title('Model Precision Comparison (All Features)', fontweight='bold', fontsize=12)
axes[0, 1].set_xlim([0, 1])
for i, v in enumerate(results_df['Precision']):
    axes[0, 1].text(v + 0.01, i, f'{v:.4f}', va='center')

# Recall comparison
axes[1, 0].barh(results_df['Model'], results_df['Recall'], color='lightgreen')
axes[1, 0].set_xlabel('Recall', fontweight='bold')
axes[1, 0].set_title('Model Recall Comparison (All Features)', fontweight='bold', fontsize=12)
axes[1, 0].set_xlim([0, 1])
for i, v in enumerate(results_df['Recall']):
    axes[1, 0].text(v + 0.01, i, f'{v:.4f}', va='center')

# F1-Score comparison
axes[1, 1].barh(results_df['Model'], results_df['F1-Score'], color='plum')
axes[1, 1].set_xlabel('F1-Score', fontweight='bold')
axes[1, 1].set_title('Model F1-Score Comparison (All Features)', fontweight='bold', fontsize=12)
axes[1, 1].set_xlim([0, 1])
for i, v in enumerate(results_df['F1-Score']):
    axes[1, 1].text(v + 0.01, i, f'{v:.4f}', va='center')

plt.tight_layout()
plt.savefig('06_model_comparison_all_features.png', dpi=300, bbox_inches='tight')
print("✓ Saved: 06_model_comparison_all_features.png")
plt.close()

# Confusion Matrix for best model
y_pred_best = best_models[best_model_name].predict(X_test_scaled)
cm = confusion_matrix(y_test, y_pred_best)

```

```

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
             xticklabels=le_target.classes_, yticklabels=le_target.classes_)
plt.title(f'Confusion Matrix - {best_model_name} (All Features)', fontweight='bold', fontsize=14)
plt.ylabel('Actual', fontweight='bold')
plt.xlabel('Predicted', fontweight='bold')
plt.tight_layout()
plt.savefig('07_confusion_matrix_best_model.png', dpi=300, bbox_inches='tight')
print("✓ Saved: 07_confusion_matrix_best_model.png")
plt.close()

# =====
# STEP 4: FEATURE SELECTION - METHOD 1 (SelectKBest)
# =====
print("\n" + "=" * 80)
print("STEP 4: FEATURE SELECTION")
print("=" * 80)

print("\nWhy Feature Selection?")
print(" - Reduces model complexity")
print(" - Improves model performance")
print(" - Reduces training time")
print(" - Helps understand which features are most important")

print("\n" + "-" * 80)
print("METHOD 1: SelectKBest (Statistical Method)")
print("-" * 80)

print("\nHow it works:")
print(" - Uses statistical tests (ANOVA F-test) to score features")
print(" - Selects K features with highest scores")
print(" - Independent of any ML model")

# SelectKBest with f_classif (ANOVA F-test)
k_best = 10 # Select top 10 features
selector_kbest = SelectKBest(score_func=f_classif, k=k_best)
X_train_kbest = selector_kbest.fit_transform(X_train_scaled, y_train)
X_test_kbest = selector_kbest.transform(X_test_scaled)

# Get feature scores
feature_scores_kbest = pd.DataFrame({
    'Feature': X_encoded.columns,
    'Score': selector_kbest.scores_
}).sort_values('Score', ascending=False)

print(f"\n✓ Selected top {k_best} features using SelectKBest:")
print(feature_scores_kbest.head(k_best).to_string(index=False))

selected_features_kbest = feature_scores_kbest.head(k_best)['Feature'].tolist()

# =====
# METHOD 2: Random Forest Feature Importance
# =====
print("\n" + "-" * 80)
print("METHOD 2: Random Forest Feature Importance (Model-based)")
print("-" * 80)

print("\nHow it works:")
print(" - Trains a Random Forest model")
print(" - Calculates importance based on how much each feature")
print("   decreases impurity in decision trees")
print(" - Model-dependent method")

# Train Random Forest for feature importance
rf_importance = RandomForestClassifier(n_estimators=100, random_state=42)
rf_importance.fit(X_train_scaled, y_train)

# Get feature importances

```

```

feature_importance_rf = pd.DataFrame({
    'Feature': X_encoded.columns,
    'Importance': rf_importance.feature_importances_
}).sort_values('Importance', ascending=False)

print(f"\n✓ Top {k_best} features by Random Forest Importance:")
print(feature_importance_rf.head(k_best).to_string(index=False))

selected_features_rf = feature_importance_rf.head(k_best)['Feature'].tolist()

# =====
# VISUALIZATION: Feature Selection
# =====
print("\n[4.1] Creating feature selection visualizations...")

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# SelectKBest scores
top_features_kb = feature_scores_kbest.head(15)
axes[0].barh(range(len(top_features_kb)), top_features_kb['Score'], color='steelblue')
axes[0].set_yticks(range(len(top_features_kb)))
axes[0].set_yticklabels(top_features_kb['Feature'])
axes[0].set_xlabel('F-Score', fontweight='bold')
axes[0].set_title('Top 15 Features - SelectKBest (ANOVA F-test)', fontweight='bold', fontsize=12)
axes[0].invert_yaxis()

# Random Forest importance
top_features_rf = feature_importance_rf.head(15)
axes[1].barh(range(len(top_features_rf)), top_features_rf['Importance'], color='forestgreen')
axes[1].set_yticks(range(len(top_features_rf)))
axes[1].set_yticklabels(top_features_rf['Feature'])
axes[1].set_xlabel('Importance Score', fontweight='bold')
axes[1].set_title('Top 15 Features - Random Forest Importance', fontweight='bold', fontsize=12)
axes[1].invert_yaxis()

plt.tight_layout()
plt.savefig('08_feature_selection_comparison.png', dpi=300, bbox_inches='tight')
print("✓ Saved: 08_feature_selection_comparison.png")
plt.close()

# =====
# STEP 5: APPLY ML MODELS TO SELECTED FEATURES
# =====
print("\n" + "=" * 80)
print("STEP 5: APPLYING ML MODELS TO SELECTED FEATURES")
print("=" * 80)

print(f"\nUsing top {k_best} features from SelectKBest method")
print(f"Selected features: {selected_features_kbest}")

# Get indices of selected features
selected_indices = [X_encoded.columns.get_loc(feats) for feats in selected_features_kbest]
X_train_selected = X_train_scaled[:, selected_indices]
X_test_selected = X_test_scaled[:, selected_indices]

print(f"\n✓ Training set shape: {X_train_selected.shape}")
print(f"✓ Testing set shape: {X_test_selected.shape}")

# Train models on selected features
results_selected = []
best_models_selected = {}

print("\n[5.1] Training models with selected features...\n")

for model_name, mp in models_params.items():
    print(f"\nTraining: {model_name} (Selected Features)")

    # Grid Search
    grid_search = GridSearchCV(

```

```

        mp['model'],
        mp['params'],
        cv=5,
        scoring='accuracy',
        n_jobs=-1,
        verbose=0
    )

    grid_search.fit(X_train_selected, y_train)

    # Best model
    best_model = grid_search.best_estimator_
    best_models_selected[model_name] = best_model

    # Predictions
    y_pred = best_model.predict(X_test_selected)

    # Metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    results_selected.append({
        'Model': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    })

    print(f"    ✓ Accuracy: {accuracy:.4f}, F1-Score: {f1:.4f}")

print("\n" + "=" * 80)
print("MODEL COMPARISON - SELECTED FEATURES")
print("=" * 80)

results_selected_df = pd.DataFrame(results_selected)
print("\n" + results_selected_df.to_string(index=False))

# Find best model with selected features
best_idx_selected = results_selected_df['Accuracy'].idxmax()
best_model_name_selected = results_selected_df.loc[best_idx_selected, 'Model']
best_accuracy_selected = results_selected_df.loc[best_idx_selected, 'Accuracy']

print(f"\n🏆 BEST MODEL (Selected Features): {best_model_name_selected} with Accuracy = {best_accuracy_selected:.4f}")

# =====
# VISUALIZATION: Selected Features Model Comparison
# =====
print("\n[5.2] Creating visualizations for selected features...")

fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Accuracy comparison
axes[0, 0].barh(results_selected_df['Model'], results_selected_df['Accuracy'], color='skyblue')
axes[0, 0].set_xlabel('Accuracy', fontweight='bold')
axes[0, 0].set_title('Model Accuracy Comparison (Selected Features)', fontweight='bold',
                    fontsize=12)
axes[0, 0].set_xlim([0, 1])
for i, v in enumerate(results_selected_df['Accuracy']):
    axes[0, 0].text(v + 0.01, i, f'{v:.4f}', va='center')

# Precision comparison
axes[0, 1].barh(results_selected_df['Model'], results_selected_df['Precision'],
                color='lightcoral')
axes[0, 1].set_xlabel('Precision', fontweight='bold')
axes[0, 1].set_title('Model Precision Comparison (Selected Features)', fontweight='bold',
                    fontsize=12)

```

```

    fontsize=12)
    axes[0, 1].set_xlim([0, 1])
    for i, v in enumerate(results_selected_df['Precision']):
        axes[0, 1].text(v + 0.01, i, f'{v:.4f}', va='center')

# Recall comparison
axes[1, 0].barh(results_selected_df['Model'], results_selected_df['Recall'], color='lightgreen')
axes[1, 0].set_xlabel('Recall', fontweight='bold')
axes[1, 0].set_title('Model Recall Comparison (Selected Features)', fontweight='bold',
    fontsize=12)
axes[1, 0].set_xlim([0, 1])
for i, v in enumerate(results_selected_df['Recall']):
    axes[1, 0].text(v + 0.01, i, f'{v:.4f}', va='center')

# F1-Score comparison
axes[1, 1].barh(results_selected_df['Model'], results_selected_df['F1-Score'], color='plum')
axes[1, 1].set_xlabel('F1-Score', fontweight='bold')
axes[1, 1].set_title('Model F1-Score Comparison (Selected Features)', fontweight='bold',
    fontsize=12)
axes[1, 1].set_xlim([0, 1])
for i, v in enumerate(results_selected_df['F1-Score']):
    axes[1, 1].text(v + 0.01, i, f'{v:.4f}', va='center')

plt.tight_layout()
plt.savefig('09_model_comparison_selected_features.png', dpi=300, bbox_inches='tight')
print("✓ Saved: 09_model_comparison_selected_features.png")
plt.close()

# =====
# COMPARISON: All Features vs Selected Features
# =====
print("\n" + "=" * 80)
print("FINAL COMPARISON: ALL FEATURES vs SELECTED FEATURES")
print("=" * 80)

comparison_df = pd.DataFrame({
    'Model': results_df['Model'],
    'Accuracy (All)': results_df['Accuracy'],
    'Accuracy (Selected)': results_selected_df['Accuracy'],
    'Difference': results_selected_df['Accuracy'] - results_df['Accuracy']
})

print("\n" + comparison_df.to_string(index=False))

# Visualization: Side-by-side comparison
fig, ax = plt.subplots(figsize=(14, 6))

x = np.arange(len(results_df))
width = 0.35

bars1 = ax.bar(x - width/2, results_df['Accuracy'], width, label='All Features',
    color='steelblue')
bars2 = ax.bar(x + width/2, results_selected_df['Accuracy'], width, label='Selected Features',
    color='coral')

ax.set_xlabel('Models', fontweight='bold', fontsize=12)
ax.set_ylabel('Accuracy', fontweight='bold', fontsize=12)
ax.set_title('Model Performance: All Features vs Selected Features', fontweight='bold',
    fontsize=14)
ax.set_xticks(x)
ax.set_xticklabels(results_df['Model'], rotation=15, ha='right')
ax.legend()
ax.set_ylim([0, 1.1])
ax.grid(axis='y', alpha=0.3)

# Add value labels on bars
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()

```

```
ax.text(bar.get_x() + bar.get_width()/2., height,
        f'{height:.3f}', ha='center', va='bottom', fontsize=9)
```

```
plt.tight_layout()
plt.savefig('10_all_vs_selected_features.png', dpi=300, bbox_inches='tight')
print("✓ Saved: 10_all_vs_selected_features.png")
plt.close()
```

```
# =====
# PROJECT SUMMARY
# =====
print("\n" + "=" * 80)
print("PROJECT 2 COMPLETION SUMMARY")
print("=" * 80)
```

```
print(f"""
```

```
    DATASET PREPARATION ✓
```

Dataset: Kidney Disease (Cleaned from Project 1)

- Total samples: {df.shape[0]}
- Total features: {df.shape[1]}
- Missing values: {df.isnull().sum().sum()} ✓
- Training samples: {X_train.shape[0]}
- Testing samples: {X_test.shape[0]}

```
    MACHINE LEARNING WITH GRID SEARCH ✓
```

Models Trained: {len(models_params)}

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)

Grid Search Optimization:

- Cross-validation folds: 5
- Hyperparameters tested: Multiple combinations per model
- Best parameters automatically selected

Best Model (All Features):

- {best_model_name}
- Accuracy: {best_accuracy:.4f}
- Best Parameters: {results[best_idx]['Best Parameters']}

```
    FEATURE SELECTION ✓
```

Method 1: SelectKBest (Statistical - ANOVA F-test)

- Top {k_best} features selected
- Features: {'', '.join(selected_features_kbest[:5]))}...

Method 2: Random Forest Feature Importance (Model-based)

- Top {k_best} features selected
- Features: {'', '.join(selected_features_rf[:5]))}...

Why Two Methods?

- SelectKBest: Statistical approach, model-independent
- RF Importance: Model-based, captures feature interactions
- Comparing both gives robust feature selection

```
    ML MODELS ON SELECTED FEATURES ✓
```

Best Model (Selected Features):

- {best_model_name_selected}
- Accuracy: {best_accuracy_selected:.4f}
- Features used: {k_best} (reduced from {X_encoded.shape[1]})

Performance Impact:

- Feature reduction: $\{((X_encoded.shape[1] - k_best) / X_encoded.shape[1] * 100):.1f\}$ % fewer features
- Accuracy change: $\{(best_accuracy_selected - best_accuracy):.4f\}$

KEY INSIGHTS

1. Grid Search Benefits:

- Automatically found optimal hyperparameters
- Improved model performance through systematic tuning
- Saved time compared to manual parameter testing

2. Feature Selection Benefits:

- Identified most important predictors of kidney disease
- Reduced model complexity
- Maintained (or improved) accuracy with fewer features

3. Model Performance:

- Best overall model: {best_model_name}
- Most important features identified
- Ready for deployment or further optimization

COMPLETED SUCCESSFULLY!

""")

print("\n" + "=" * 80)

print("FILES GENERATED:")

print("=" * 80)

print("""

Visualizations:

- ✓ 06_model_comparison_all_features.png
- ✓ 07_confusion_matrix_best_model.png
- ✓ 08_feature_selection_comparison.png
- ✓ 09_model_comparison_selected_features.png
- ✓ 10_all_vs_selected_features.png

""")

print("\n" + "=" * 80)

print("END")

print("=" * 80)