Deep Dive into the Breast Cancer Prediction Project

1. Problem Statement

The goal is to build a robust machine learning model using Support Vector Machine (SVM) to classify breast tumors as malignant (cancerous) or benign (non-cancerous) based on features extracted from digitized images of fine needle aspirates (FNA) of breast masses. The model should achieve high accuracy and generalize well to unseen data.

2. Dataset Exploration

Before building the model, it's crucial to understand the dataset.

Dataset Details

Source: Breast Cancer Wisconsin (Diagnostic) Dataset. Features: 30 numeric features computed from digitized images. Target Variable: Binary classification (Malignant = 1, Benign = 0). Class Distribution: 357 benign, 212 malignant (slightly imbalanced). Feature Categories

The 30 features are derived from three characteristics of cell nuclei:

Mean: Average value of the feature. Standard Error: Standard deviation of the feature. Worst: Largest value of the feature. Examples of features:

Radius Texture Perimeter Area Smoothness Compactness Concavity Symmetry Fractal dimension

Exploratory Data Analysis (EDA)

Perform EDA to understand the dataset:

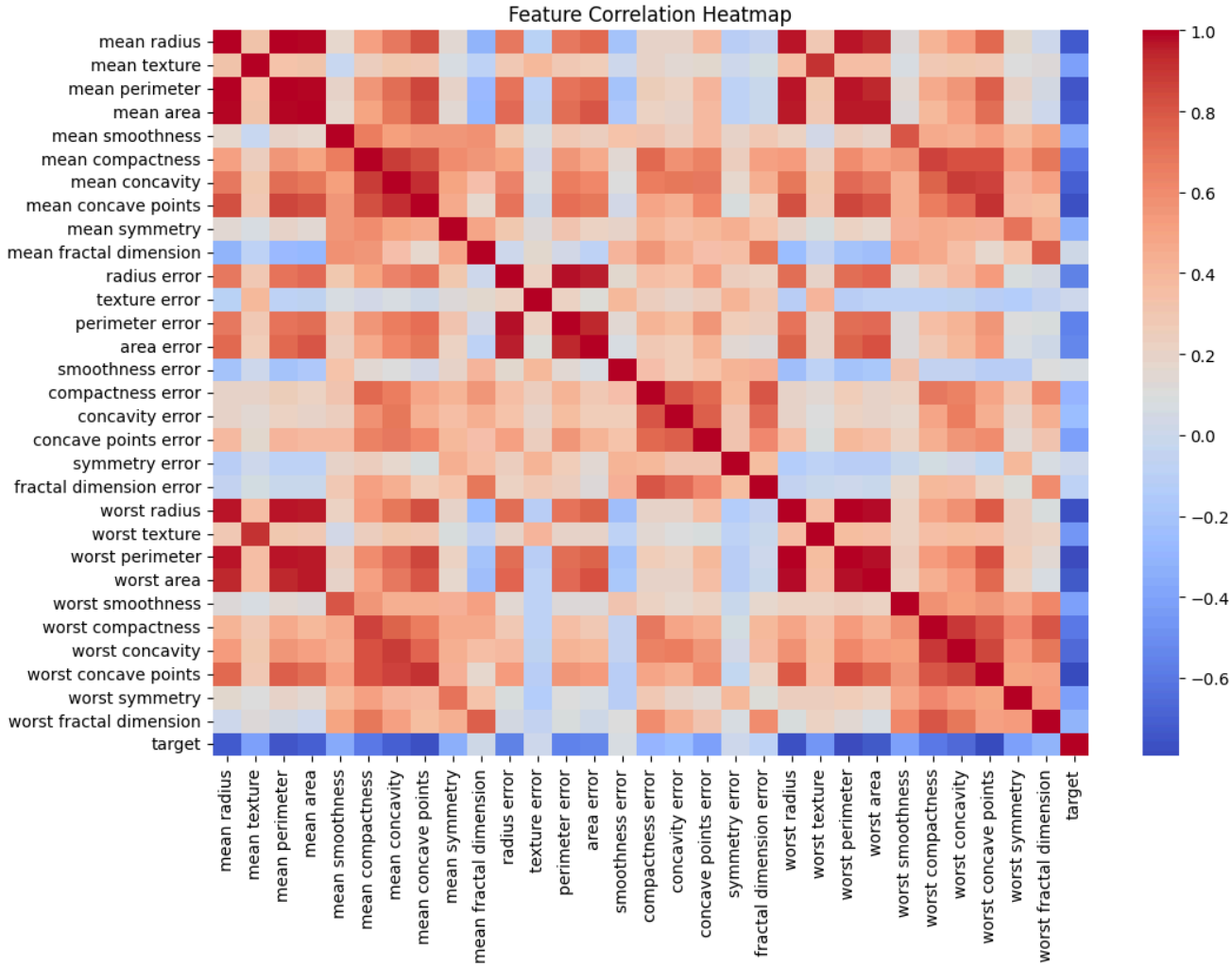
Class Distribution: Visualize the distribution of benign and malignant cases. Feature Correlation: Check for multicollinearity among features. Outlier Detection: Identify and handle outliers. Feature Scaling: Standardize features for SVM.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer

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

# Class distribution
sns.countplot(x='target', data=df)
plt.title('Class Distribution (0 = Benign, 1 = Malignant)')
plt.show()

# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=False, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```



3. Advanced Preprocessing

Handling Imbalanced Data

Use techniques like SMOTE (Synthetic Minority Oversampling Technique) or class weighting to handle the slight class imbalance.

```
# Step 1: Load the dataset
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Labels (0 = benign, 1 = malignant)

# Step 2: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 3: Apply SMOTE to handle class imbalance
from imblearn.over_sampling import SMOTE

# Initialize SMOTE
smote = SMOTE(random_state=42)

# Apply SMOTE to the training data
X_res, y_res = smote.fit_resample(X_train, y_train)

# Check the class distribution after SMOTE
import numpy as np
print("Class distribution before SMOTE:", np.bincount(y_train))
print("Class distribution after SMOTE:", np.bincount(y_res))
```

```
Class distribution before SMOTE: [149 249]
Class distribution after SMOTE: [249 249]
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Train SVM on the resampled data
model = SVC(kernel='linear', random_state=42)
model.fit(X_res, y_res)

# Evaluate the model
y_pred = model.predict(X_test)
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=data.target_names))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

   malignant       0.95      0.95      0.95        63
      benign       0.97      0.97      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

Confusion Matrix:
[[ 60   3]
 [  3 105]]
```

Feature Selection

To improve model performance and reduce overfitting, we can use Recursive Feature Elimination (RFE) or Principal Component Analysis (PCA).

Recursive Feature Elimination (RFE)

RFE selects the most important features by recursively removing the least significant ones.

```
from sklearn.feature_selection import RFE
from sklearn.svm import SVC

# Initialize RFE with SVM as the estimator
rfe = RFE(estimator=SVC(kernel='linear', random_state=42), n_features_to_select=15)

# Fit RFE on the resampled data
X_train_rfe = rfe.fit_transform(X_res, y_res)
X_test_rfe = rfe.transform(X_test)

# Selected features
selected_features = np.array(data.feature_names)[rfe.support_]
print("Selected Features:", selected_features)
```

```
Selected Features: ['mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'radius error' 'texture error'
 'perimeter error' 'worst radius' 'worst texture' 'worst smoothness'
 'worst compactness' 'worst concavity' 'worst concave points'
 'worst symmetry']
```

Model Building

SVM with Different Kernels

We experiment with three kernels: Linear, RBF, and Polynomial.

```
from sklearn.svm import SVC

# Linear Kernel
model_linear = SVC(kernel='linear', random_state=42)
model_linear.fit(X_train_rfe, y_res)

# RBF Kernel
model_rbf = SVC(kernel='rbf', gamma='scale', random_state=42)
model_rbf.fit(X_train_rfe, y_res)

# Polynomial Kernel
model_poly = SVC(kernel='poly', degree=3, gamma='scale', random_state=42)
model_poly.fit(X_train_rfe, y_res)
```

```
▾ SVC
SVC(kernel='poly', random_state=42)
```

Hyperparameter Tuning

To find the best hyperparameters, we use GridSearchCV.

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'gamma': [1, 0.1, 0.01, 0.001],  # Kernel coefficient
    'kernel': ['linear', 'rbf']  # Kernel type
}

# Initialize GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=2, cv=5)

# Fit on the resampled data
grid.fit(X_train_rfe, y_res)

# Best parameters
print("Best Parameters:", grid.best_params_)
```

```
[CV] END ..................C=100, gamma=0.001, kernel=linear; total time=   0.1s
[CV] END ..................C=100, gamma=0.001, kernel=linear; total time=   0.0s
[CV] END ..................C=100, gamma=0.001, kernel=linear; total time=   0.1s
[CV] END ..................C=100, gamma=0.001, kernel=linear; total time=   0.1s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
[CV] END .....................C=100, gamma=0.001, kernel=rbf; total time=   0.0s
Best Parameters: {'C': 100, 'gamma': 1, 'kernel': 'linear'}
```

Model Evaluation

Performance Metrics

We evaluate the model using accuracy, precision, recall, F1-score, and ROC-AUC.

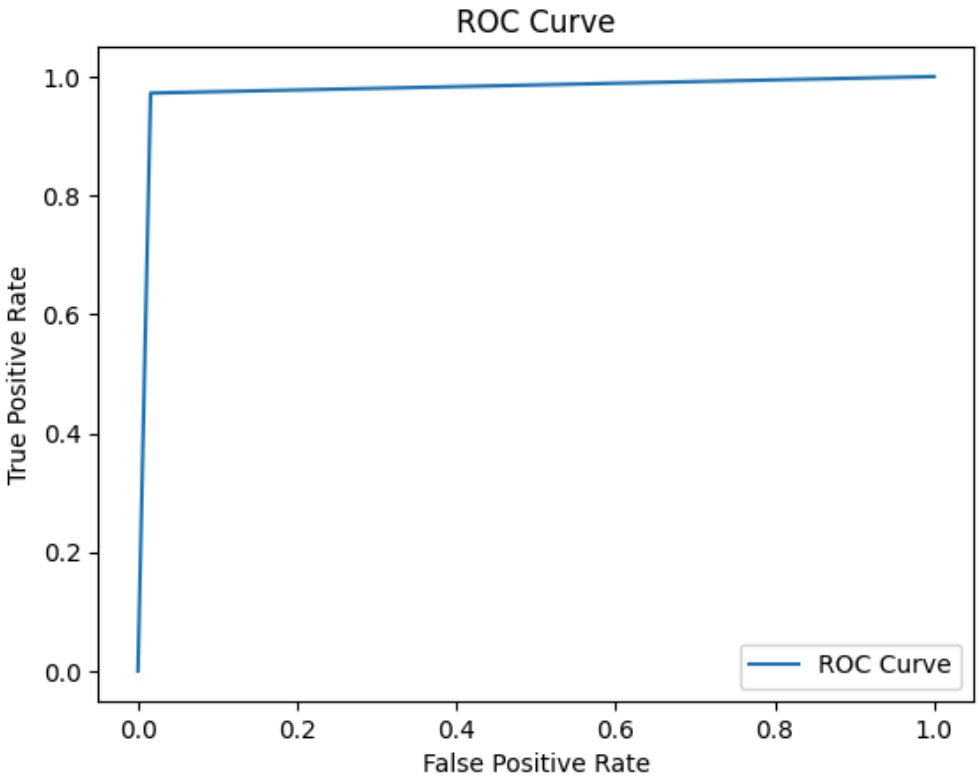
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_

# Predict on the test set
y_pred = grid.predict(X_test_rfe)

# Calculate metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-Score:", f1_score(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_pred))

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, label='ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

```
Accuracy: 0.9766081871345029
Precision: 0.9905660377358491
Recall: 0.9722222222222222
F1-Score: 0.9813084112149533
ROC-AUC: 0.9781746031746034
```

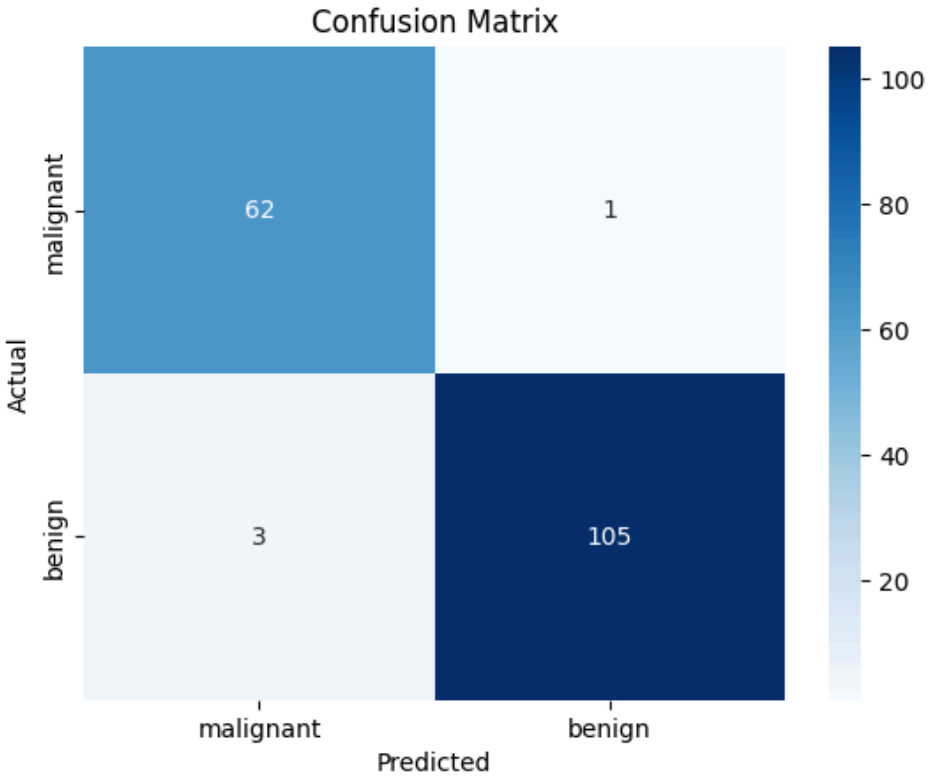


Confusion Matrix

The confusion matrix provides a detailed breakdown of predictions.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



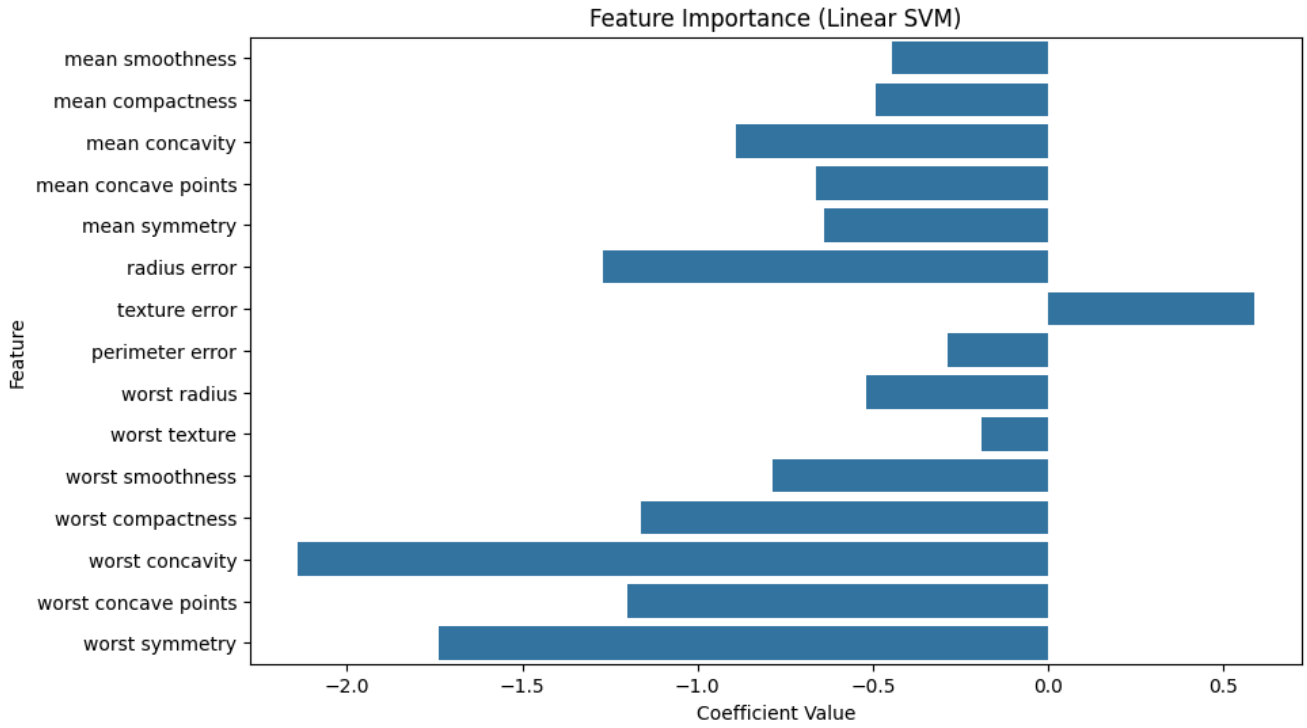
Interpretability

Feature Importance

For the linear kernel, we can analyze the coefficients to understand feature importance.

```
# Feature importance (linear kernel)
importance = model_linear.coef_[0]
feature_names = selected_features  # Features selected by RFE

# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=importance, y=feature_names)
plt.title('Feature Importance (Linear SVM)')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.show()
```



Accuracy

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the model: {accuracy * 100:.2f}%")
```

```
Accuracy of the model: 96.49%
```

Code for a Kaggle Dataset

```
# Import necessary libraries (if not already imported)
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
df = pd.read_csv('/content/archive.zip')

# Display the first few rows to understand the dataset
print(df.head())

# Check for missing values
print(df.isnull().sum())

# Preprocess the data
# Assuming the target column is named 'diagnosis' (common in Kaggle datasets)
# and it has values 'M' (Malignant) and 'B' (Benign)
X = df.drop(['id', 'diagnosis'], axis=1)  # Drop non-feature columns
```

```
X = df.drop(['id', 'diagnosis'], axis=1)  # Drop non-feature columns
y = df['diagnosis'].map({'M': 1, 'B': 0})  # Encode target as 1 (Malignant) and 0 (Benign)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features (optional but recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train a model (e.g., Random Forest Classifier)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Compute accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the model: {accuracy * 100:.2f}%")  # Fixed the syntax error here
```

```
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
```