

# Breast Cancer Detection (Wisconsin Dataset) — End-to-End Project

## Overview

This repository contains an end-to-end breast cancer detection project using the Wisconsin Breast Cancer dataset (available in `sklearn.datasets`). It includes data preprocessing, model training (SVM with hyperparameter tuning), model saving/loading, evaluation, a simple Flask inference API, a Jupyter notebook for exploration, and instructions to containerize with Docker.

---

## File structure

```
breast-cancer-wisconsin/  
├── README.md  
├── requirements.txt  
├── .gitignore  
├── data_pipeline/  
│   ├── __init__.py  
│   ├── load_data.py  
│   ├── preprocess.py  
│   └── split_save.py  
├── models/  
│   ├── __init__.py  
│   ├── train.py  
│   ├── evaluate.py  
│   └── persistence.py  
├── api/  
│   ├── app.py  
│   └── predict_client.py  
├── notebooks/  
│   └── exploration.ipynb  
├── docker/  
│   └── Dockerfile  
└── scripts/  
    └── run_all.sh
```

---

# README (this file also contains usage, placed here for convenience)

## Prerequisites

- Python 3.8+
- pip
- Optional: Docker

## Setup

1. Create a virtual environment and activate it (recommended):

```
python -m venv venv
source venv/bin/activate    # macOS / Linux
venv\Scripts\activate      # Windows
```

1. Install dependencies:

```
pip install -r requirements.txt
```

## Quick run (train + evaluate)

```
python models/train.py --out models/svm_joblib.pkl
python models/evaluate.py --model models/svm_joblib.pkl
```

## Run API (after training)

```
# make sure model path in api/app.py or provide via env var
python api/app.py
# then call the client
python api/predict_client.py
```

## Docker (optional)

From project root:

```
docker build -t bc-wisconsin:latest -f docker/Dockerfile .
docker run -p 5000:5000 bc-wisconsin:latest
```

## requirements.txt

```
flask
scikit-learn
pandas
numpy
joblib
matplotlib
seaborn
jupyter
gunicorn
```

## .gitignore

```
venv/
__pycache__/
*.pyc
models/*.pkl
.ipynb_checkpoints/
```

## data\_pipeline/load\_data.py

```
# data_pipeline/load_data.py
from sklearn.datasets import load_breast_cancer
import pandas as pd

def load_wisconsin_as_df():
    data = load_breast_cancer()
    df = pd.DataFrame(data.data, columns=data.feature_names)
    df['target'] = data.target
    # target: 0 = malignant, 1 = benign (sklearn encoding)
    return df

if __name__ == '__main__':
    df = load_wisconsin_as_df()
    print(df.shape)
    print(df.head())
```

---

## data\_pipeline/preprocess.py

```
# data_pipeline/preprocess.py
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold

def basic_preprocess(df, drop_low_variance=True, scaler=None):
    X = df.drop(columns=['target']).copy()
    y = df['target'].copy()

    # Optional: drop constant/near-constant features
    if drop_low_variance:
        sel = VarianceThreshold(threshold=1e-5)
        X = pd.DataFrame(sel.fit_transform(X), columns=[f for f in
X.columns[sel.get_support(indices=True)]])

    # Scaling
    if scaler is None:
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X)
    else:
        X_scaled = scaler.transform(X)

    return X_scaled, y.values, scaler
```

---

## data\_pipeline/split\_save.py

```
# data_pipeline/split_save.py
import joblib
from sklearn.model_selection import train_test_split
from .load_data import load_wisconsin_as_df
from .preprocess import basic_preprocess

def prepare_and_save(test_size=0.2, random_state=42, out_dir='models'):
    df = load_wisconsin_as_df()
    X, y, scaler = basic_preprocess(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, random_state=random_state, stratify=y)
```

```

        joblib.dump({'scaler': scaler}, f'{out_dir}/scaler.joblib')
        joblib.dump({'X_test': X_test, 'y_test': y_test}, f'{out_dir}/
test_data.joblib')
        print('Saved scaler and test split to', out_dir)

if __name__ == '__main__':
    prepare_and_save()

```

## models/train.py

```

# models/train.py
import argparse
import os
import joblib
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from data_pipeline.load_data import load_wisconsin_as_df
from data_pipeline.preprocess import basic_preprocess

def train_and_save(out_path='models/svm.joblib.pkl'):
    os.makedirs(os.path.dirname(out_path) or '.', exist_ok=True)
    df = load_wisconsin_as_df()
    X, y, scaler = basic_preprocess(df)

    # Simple SVM with grid search
    param_grid = {
        'C': [0.1, 1, 10],
        'kernel': ['rbf', 'linear'],
        'gamma': ['scale', 'auto']
    }

    svc = SVC(probability=True)
    grid = GridSearchCV(svc, param_grid, cv=5, scoring='f1', n_jobs=-1)
    grid.fit(X, y)

    print('Best params:', grid.best_params_)
    print('Best CV score:', grid.best_score_)

    # Save model + scaler together
    joblib.dump({'model': grid.best_estimator_, 'scaler': scaler, 'cv_results':
grid.cv_results_}, out_path)

```

```

print('Saved model to', out_path)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--out', '--out_path', dest='out_path', default='models/
svm_joblib.pkl')
    args = parser.parse_args()
    train_and_save(args.out_path)

```

## models/evaluate.py

```

# models/evaluate.py
import argparse
import joblib
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix
import matplotlib.pyplot as plt

def evaluate(model_path='models/svm_joblib.pkl'):
    obj = joblib.load(model_path)
    model = obj['model']
    scaler = obj['scaler']

    # load test data from split_save or recreate split
    try:
        td = joblib.load('models/test_data.joblib')
        X_test = td['X_test']
        y_test = td['y_test']
    except Exception:
        # fallback: recreate split (deterministic)
        from data_pipeline.load_data import load_wisconsin_as_df
        from data_pipeline.preprocess import basic_preprocess
        from sklearn.model_selection import train_test_split
        df = load_wisconsin_as_df()
        X, y, _ = basic_preprocess(df, scaler=scaler)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

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

```

```

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:', f1_score(y_test, y_pred))
try:
    print('ROC AUC:', roc_auc_score(y_test, y_proba))
except Exception:
    pass

cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cm)

# Plot ROC curve quickly
try:
    from sklearn.metrics import roc_curve
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    plt.figure()
    plt.plot(fpr, tpr)
    plt.xlabel('FPR')
    plt.ylabel('TPR')
    plt.title('ROC curve')
    plt.grid(True)
    plt.savefig('models/roc_curve.png')
    print('Saved ROC curve to models/roc_curve.png')
except Exception:
    pass

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', dest='model', default='models/
svm_joblib.pkl')
    args = parser.parse_args()
    evaluate(args.model)

```

---

## models/persistence.py

```

# models/persistence.py
import joblib

def save_model(path, model, scaler=None, extras=None):
    payload = {'model': model}
    if scaler is not None:

```

```

        payload['scaler'] = scaler
    if extras is not None:
        payload.update(extras)
    joblib.dump(payload, path)

def load_model(path):
    return joblib.load(path)

```

## api/app.py

```

# api/app.py
import os
from flask import Flask, request, jsonify
import joblib
import numpy as np

app = Flask(__name__)

MODEL_PATH = os.environ.get('MODEL_PATH', 'models/svm_joblib.pkl')
obj = joblib.load(MODEL_PATH)
model = obj['model']
scaler = obj['scaler']

@app.route('/')
def hello():
    return 'Breast Cancer Detection API - Wisconsin Dataset'

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    # Expect JSON: { "features": [ ... ] } where features length = n_features
    # after preprocessing
    if data is None or 'features' not in data:
        return jsonify({'error': 'provide features array'}), 400

    features = np.array(data['features']).reshape(1, -1)
    # If the saved scaler expects a different original shape, the client should
    # pass scaled features, or
    # alternatively, adjust this API to accept named features. Here we assume
    # preprocessed features.

    proba = model.predict_proba(features)[0, 1]
    pred = int(model.predict(features)[0])

```



```

        return jsonify({'prediction': int(pred), 'probability_benign':
float(proba)})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

## api/predict\_client.py

```

# api/predict_client.py
import requests
import json
from data_pipeline.load_data import load_wisconsin_as_df
from data_pipeline.preprocess import basic_preprocess

# Quick example: take first sample from dataset and send to API

def send_sample(url='http://127.0.0.1:5000/predict'):
    df = load_wisconsin_as_df()
    X, y, scaler = basic_preprocess(df)
    sample = X[0].tolist()
    resp = requests.post(url, json={'features': sample})
    print(resp.status_code, resp.text)

if __name__ == '__main__':
    send_sample()

```

## notebooks/exploration.ipynb

A Jupyter notebook is included in `notebooks/` (generate one locally or run the provided scripts). The notebook should show EDA (distributions, correlation heatmap), PCA visualization, model training snippets, and evaluation visualizations.

## docker/Dockerfile

```

FROM python:3.10-slim
WORKDIR /app
COPY . /app

```

```
RUN pip install --upgrade pip && pip install -r requirements.txt
EXPOSE 5000
CMD ["gunicorn", "-b", "0.0.0.0:5000", "api.app:app"]
```

---

## scripts/run\_all.sh

```
#!/usr/bin/env bash
set -e
python data_pipeline/split_save.py
python models/train.py --out models/svm_joblib.pkl
python models/evaluate.py --model models/svm_joblib.pkl
```

---

## Notes & extensions

- The API currently expects preprocessed/scaled features. For production, change the API to accept raw named features and do the same preprocessing pipeline server-side (use `data_pipeline.preprocess` and `data_pipeline.load_data` logic).
- Consider using `Pipeline` from `sklearn.pipeline` to bundle scaler + model before saving, making inference simpler:

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([('scaler', scaler), ('svc', best_estimator)])
joblib.dump(pipe, 'models/pipe.pkl')
```

- Add unit tests, CI, and a small frontend if you want to demo predictions in browser.

---

## License

MIT

---

*End of project scaffold.*