Machine Learning Interpretability - Black Box Models

- **White-box models** (regression, decision trees): transparent, interpretable, suitable for smaller datasets
- **Black-box models** (neural networks, gradient-boosted trees): handle large datasets, higher accuracy, but harder to interpret
- Goal: Learn methods to interpret black-box models

White-Box vs Black-Box

- **White-box**: Easy to interpret, transparent decision-making
- **Black-box**: Complex, less interpretable but more powerful with large datasets
- Advances in computing → more widespread use of black-box models

Common Black-Box Models

Gradient Boosted Decision Trees (GBDT)

- Ensemble of weak learners combined via **gradient boosting**
- Applications:
  - Fraud detection
  - Medical outcome prediction
  - Recommender systems
  - Computer vision (e.g., self-driving cars)
  - Customer churn prediction

Neural Networks

- Inspired by biological neurons.
- Structure: Input → Hidden layers → Output.
- Applications in medicine:
  - Medical imaging (X-ray, MRI analysis)
  - Drug discovery (predict effectiveness & side effects)
  - Patient outcome prediction (e.g., disease progression)
- **White-box** = simple, interpretable.
- **Black-box** = complex, less transparent, but more accurate.

- Interpretation methods exist to explain black-box models.

Explaining Black-Boxes: Neural Networks

- Neural networks are powerful but **hard to interpret** due to their non-linear nature.
- Interpretation tools (e.g., weight/activation visualization, gradients, LIME) help explain how inputs affect outputs.
- This lesson builds and explains a neural network with **scikit-learn** using the Wisconsin Breast Cancer Dataset.

GOAL

Describe methods for interpreting/explaining neural networks

Build and interpret a neural network with scikit-learn

Dataset

- **Wisconsin Breast Cancer Dataset**: 569 samples, 30 features, 2 classes (benign/malignant)
- No missing values, but **class imbalance** (357 benign vs. 212 malignant)
- Features: mean, standard error (SE), and "worst" values of cell nuclei characteristics (radius, perimeter, texture)
- Larger nuclei ("worst") often relate to malignancy

Import Data

In [1]:
```
from sklearn.datasets import load_breast_cancer

# load the dataset
data = load_breast_cancer()
```

Basic EDA

In [2]:
```
# print the dataset description
print(data.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
```

```
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ==================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

|details-start|
**References**
|details-split|

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
  for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
```

```
  Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
  San Jose, CA, 1993.
- O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
  prognosis via linear programming. Operations Research, 43(4), pages 570-577,
  July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
  to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
  163-171.

|details-end|
```

In [3]:
```
feature_names = ["id", "radius_mean", "texture_mean", "perimeter_mean",
                 "area_mean", "smoothness_mean", "compactness_mean", "concavity_mea
                 "concave points_mean", "symmetry_mean", "fractal_dimension_mean",
                 "radius_se", "texture_se", "perimeter_se", "area_se", "smoothness_
                 "compactness_se", "concavity_se", "concave points_se", "symmetry_s
                 "fractal_dimension_se", "radius_worst", "texture_worst", "perimete
                 "area_worst", "smoothness_worst", "compactness_worst", "concavity_
                 "concave points_worst", "symmetry_worst", "fractal_dimension_worst
```

Detecting Class Imbalance

In [4]:
```
# assign the feature matrix and target vector
X = data.data
y = data.target
```

In [5]:
```
import pandas as pd

y_data = pd.Series(y)
print(y_data.value_counts())
```

```
1    357
0    212
Name: count, dtype: int64
```

Modeling

Neural Network

- Model: **MLPClassifier** (feedforward NN, trained via backpropagation)
- Parameters tuned: hidden layers, iterations, solver, learning rate
- Train/test split (80/20)

We will use the **scikit-learn** neural network library and import the `MLPClassifier`

The `MLPClassifier` is a **multi-layer feedforward neural network** trained using the **backpropagation algorithm**

What does that mean?

- **Feedforward Neural Network**
  Each neuron in a layer receives input from the previous layer, computes a value, and passes it to the next layer
- **Training Objective**
  Adjust the **weights** of the connections so the network can map inputs to the correct outputs
- **Backpropagation**
  A supervised learning algorithm that iteratively updates weights to minimize the error between the predicted and true output

Tuning Parameters

The `MLPClassifier` provides several parameters to control the model. Here are a few we'll tune in this project:

- **`hidden_layer_sizes`**
  Defines the number of hidden layers and the number of neurons in each
  Example: `(100, 100)` two hidden layers, each with 100 neurons
- **`max_iter`**
  Sets the maximum number of iterations (epochs) for the solver to run before stopping
- **`solver`**
  Defines the optimization algorithm used to train the network

For more details on all parameters, check the scikit-learn documentation.

In [6]:
```
from sklearn.neural_network import MLPClassifier

# create an instance of the classifier
clf = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000,
                    solver='sgd', random_state=1, learning_rate_init=.1)
```

Train, Test, Split

In [7]:
```
from sklearn.model_selection import train_test_split

# split your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
# train the classifier on the train dataset
clf.fit(X_train, y_train)

# predict labels for test dataset
y_pred = clf.predict(X_test)
```

Interpreting & Explaining Neural Networks

There are several ways to **interpret and explain** the results of a neural network
These methods provide insight into:

- **How predictions are made**
- **Which features influence outcomes**
- **Where improvements can be made**

We'll demonstrate how to apply these techniques with **Python** and **scikit-learn**

Global vs. Local Interpretation

- **Global Explanations** Explain the overall behavior of the model and how it makes decisions across the dataset
- **Local Explanations** Focus on understanding the prediction for a single observation (why was *this* patient classified as high-risk?)

We'll first explore **global interpretation methods**, then move on to **local explanations**

Global Explanations

Model Evaluation Metrics

To evaluate the performance of a neural network, we can use standard classification metrics such as:

- **Accuracy** – Proportion of correct predictions
- **Precision** – How many predicted positives are truly positive
- **Recall (Sensitivity)** – How many actual positives were correctly identified
- **F1-Score** – Harmonic mean of precision and recall (balances the two)

In [8]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Compute accuracy
acc = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(acc * 100))

# Compute precision
prec = precision_score(y_test, y_pred)
print("Precision: {:.2f}%".format(prec * 100))
```

```
# Compute recall
rec = recall_score(y_test, y_pred)
print("Recall: {:.2f}%".format(rec * 100))

# Compute F1 score
f1 = f1_score(y_test, y_pred)
print("F1 Score: {:.2f}%".format(f1 * 100))
```

```
Accuracy: 62.28%
Precision: 62.28%
Recall: 100.00%
F1 Score: 76.76%
```

- A recall of 100% means that every negative prediction is correct

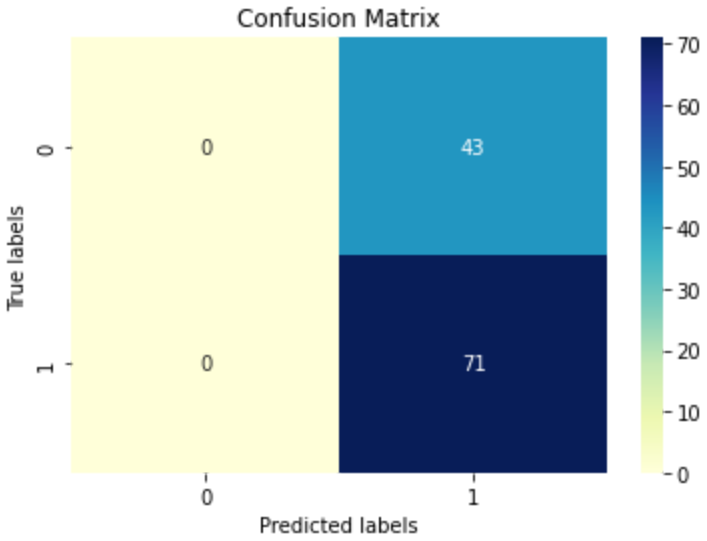
Confusion Matrix

In [9]:
```
# pip install matplotlib seaborn --upgrade
```

In [10]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Visualize the confusion matrix
sns.heatmap(cm, annot=True, cmap="YlGnBu", fmt='g')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



So what can we infer?

Key Insight

The model is **cautious** when predicting benign cases and tends to misclassify negatives as positive

Local Explanations with LIME

LIME

LIME (Local Interpretable Model-Agnostic Explanations):

- Provides explanations for **individual predictions**.
- Approximates the model **locally** around one instance (row)
- **Model-agnostic** works with any ML model
- With neural networks, using `predict_proba` helps:
  - Shows the **probability distribution** for each class.
  - Reveals the **certainty** of predictions

LIME highlights which **features** most influenced a single prediction and how confident the model was

In [11]:
```
# !pip install lime
```

In [12]:
```
from lime import lime_tabular

explainer = lime_tabular.LimeTabularExplainer(X_train, feature_names=feature_names)

# Choose a sample to explain
sample = X_test[0]

# Get the explanation for the sample
exp = explainer.explain_instance(sample, clf.predict_proba, num_features=5)

# # Print the explanation
exp_data = exp.as_list()

exp_df = pd.DataFrame(exp_data)
```

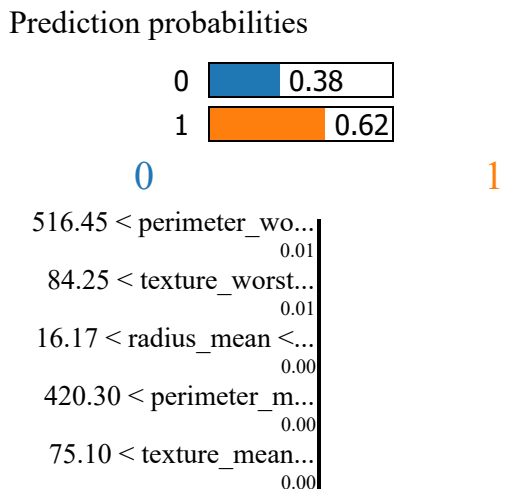
LIME Code Block Explanation

- we use the `lime_tabular.LimeTabularExplainer()` class to create a LIME explainer for the training data. You can pass the number of features you want to see the explanation for using the `num_features` parameter.

- we choose a sample from the test set (`X_test[0]`) and use the `explain_instance()` method to obtain an explanation for the prediction of the sample.
- we use the `as_list()` method to print the explanation as a list of feature-value pairs.

Visualize the explanation:

In [13]:
```
exp.show_in_notebook(show_table=True, show_all=False)
```



Feature	Value
perimeter_worst	677.90
texture_worst	96.05
radius_mean	18.60
perimeter_mean	481.90
texture_mean	81.09

Interpreting the LIME Graph

So what does this graph suggest? Let's examine the pieces from left to right.

Top Left

The prediction probabilities for the observation (first row in the data frame, i.e., index 0) are displayed:

- Probability of **malignant** = **0.62**
- Probability of **benign** = **0.38**

Center

- The bar divides features into two groups:
  - **0** → features typically associated with **benign**
  - **1** → features typically associated with **malignant**
- In this case, the only feature in group **1** is `fractal_dimension_mean`.
  - Range: greater than `0.32` but less than `0.47`
  - True value for this row: `0.40`

Right Side Table

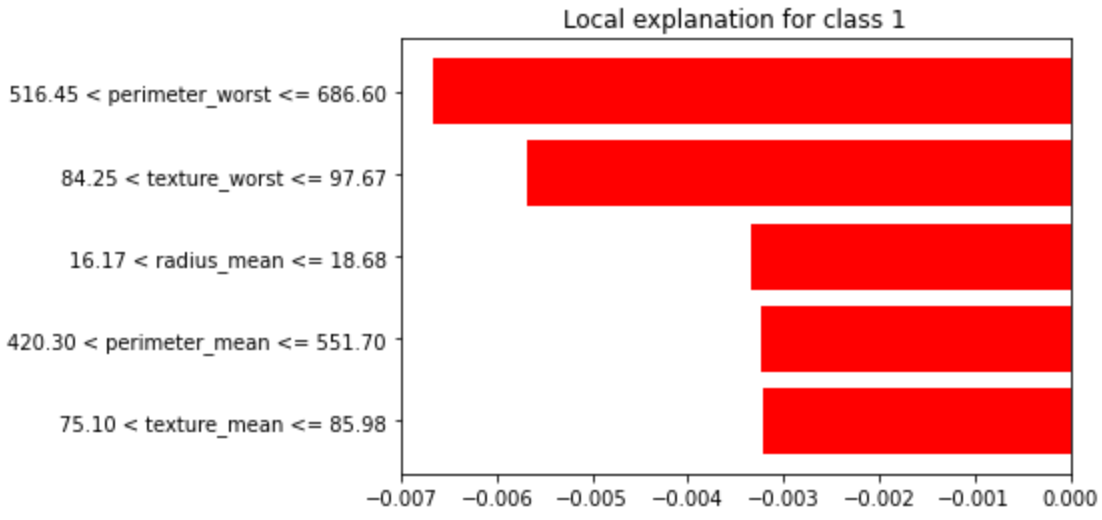
- Displays **Feature** and **Value** columns.
- `fractal_dimension_mean` is highlighted in **orange**.
- True value = `0.40`.

Inference

The model is **mildly confident** that this observation is **malignant**, primarily due to the `fractal_dimension_mean` feature.

- since other feature values are typically associated with **benign**, the model's confidence is **diminished**.
- we will look at another visualization that further explains these results.

In [14]:
```
import matplotlib as pyplot
fig = exp.as_pyplot_figure(label=1)
```



So what can we infer?

- In the chart above, we see that fractal_dimension_mean was the main contributor to the classification of malignant. and that perimeter_worst was the main contributor to uncertainty about the classification.
- Below, we express the same values as a table.

In [15]:
```
print(pd.DataFrame(exp.as_list(),columns=['Feature','Contribution']))
```

```
                            Feature  Contribution
0  516.45 < perimeter_worst <= 686.60     -0.006674
1      84.25 < texture_worst <= 97.67     -0.005673
2        16.17 < radius_mean <= 18.68     -0.003333
3   420.30 < perimeter_mean <= 551.70     -0.003227
4       75.10 < texture_mean <= 85.98     -0.003209
```

Results & Interpretability

Global Explanations

- **Metrics**: Accuracy (62%), Precision (62%), Recall (100%), F1 (76%).
- High recall → Model catches all malignant cases but cautious with benign.
- **Confusion Matrix**: Misclassifies benign as malignant more often.

Local Explanations (LIME)

- LIME explains predictions for single observations.
- Example:
  - `fractal_dimension_mean` strongly pushed prediction toward **malignant**
  - `perimeter_worst` contributed to uncertainty
- Shows **which features drive local predictions** and model confidence

Key Insights

- Features like “worst” radius/perimeter are linked to malignancy.
- Recall = 100% .... no false negatives, but potential false positives (linked to imbalance).
- Neural network explanations improve trust in predictions.

Summary

We built a neural network on medical data, evaluated it with global metrics, and explained predictions using **LIME**. While accuracy was modest, interpretation revealed **why** the model made certain decisions, helping improve reliability in sensitive applications like healthcare.

In [ ]:
```
```