**ML Wisconsin breast cancer**

**Import**

|  |
| --- |
| import numpy as np  import pandas as pd import warnings warnings.filterwarnings('ignore') import matplotlib.pyplot as plt import seaborn as sns from sklearn.preprocessing import StandardScaler,MaxAbsScaler,MinMaxScaler,RobustScaler from sklearn.model\_selection import GridSearchCV from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score,confusion\_matrix,roc\_auc\_score,ConfusionMatrixDisplay,precision\_score,recall\_score,f1\_score,classification\_report,roc\_curve,plot\_roc\_curve,auc,precision\_recall\_curve,plot\_precision\_recall\_curve,average\_precision\_score from sklearn.tree import DecisionTreeClassifier from sklearn.linear\_model import LogisticRegression from sklearn.svm import SVC |

**Reads the dataset and generates a data frame.**

|  |
| --- |
| names = ['id', 'clump\_thickness', 'uniform\_cell\_size', 'uniform\_cell\_shape','marginal\_adhesion', 'single\_epithelial\_size', 'bare\_nuclei',  'bland\_chromatin', 'normal\_nucleoli', 'mitoses', 'class'] df = pd.read\_table("https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data",sep=',',names=names).replace('?', np.nan).dropna() print(df.describe()) print(df.isna().sum()) |

|  |
| --- |
|  |

**A function that will show the correlation of each feature.**

|  |
| --- |
| def visualizationCorrelation(df):  # compute the corr matrix  corr = df.corr()   # generate a mask for the upper triangle  mask = np.triu(np.ones\_like(corr, dtype=bool))   # set up the matplotlib figure  f, ax = plt.subplots(figsize=(8, 6))   # generate a custom diverging colormap  cmap = sns.diverging\_palette(230, 20, as\_cmap=True)   # draw the heatpmap with the mask and correct aspect ratio  sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0, square=True, linewidths=.5, cbar\_kws={'shrink': .5})  plt.subplots\_adjust(left=0, bottom=0.24, right=1, top=1)  plt.show() |

|  |
| --- |
|  |

**Since we saw that id values are irrelevant, drop id.**

|  |
| --- |
| df.drop(['id'], 1, inplace=True) |

**Divide the features and target and split the dataset.**

|  |
| --- |
| X = np.array(df.drop(['class'], 1)) y = np.array(df['class']) X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25,random\_state=0) |

**Models & Scalers that have undergone hyperparameter tuning to be used in grid search.**

|  |
| --- |
| g\_models = [  (DecisionTreeClassifier(), [{'criterion':['entropy'],'splitter':['best','random'],  'max\_depth':[None,2,3],'max\_features':[None, 'sqrt','log2']}]),  (DecisionTreeClassifier(), [{'criterion':['gini'],'splitter':['best','random'],  'max\_depth':[None,2,3],'max\_features':[None, 'sqrt','log2']}]),  (LogisticRegression(), [{'C' : [0.001, 0.01, 0.1, 1, 10, 100],'max\_iter': [100,1000]}]),  (SVC(), [{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'kernel': ['rbf', 'linear', 'poly'],'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}])]  # Scaler list scaler = [StandardScaler(), MinMaxScaler(),MaxAbsScaler(),RobustScaler()]  # k-fold k value list k\_fold\_k = [5, 6, 7, 8, 9, 10] |

**A function that will find the best combination.**

|  |
| --- |
| def findbest(scaler,g\_models):  # Proceed with hyperparameter tuning through grid search.  for scaler in scaler:  X\_train\_res = scaler.fit\_transform(X\_train)  X\_test\_res = scaler.fit\_transform(X\_test)  result\_list = []  for model, param in g\_models:  for k in k\_fold\_k:  result = []  grid = GridSearchCV(estimator=model, param\_grid=param, scoring='accuracy', cv=k, n\_jobs=-1)  grid.fit(X\_train\_res, y\_train)  #print(' {}: \n Best Accuracy: {:.2f} %'.format(model, grid.best\_score\_ \* 100))  #print('\n Best Parameter : {}', grid.best\_params\_)   # predict with best model and calculate MSE  best\_model = grid.best\_estimator\_  y\_pred = best\_model.predict(X\_test\_res)   #print(confusion\_matrix(y\_test, y\_pred))  #print(classification\_report(y\_test, y\_pred))   result.append(model)  result.append(grid.best\_params\_)  result.append(scaler)  result.append(accuracy\_score(y\_test, y\_pred) \* 100)  result.append(k)  result\_list.append(result)  result\_df = pd.DataFrame(result\_list, columns=['Model', 'Best parameters', 'Scaler', 'Accuracy', 'k-fold k value'])  print(result\_df)  print() |

|  |
| --- |
|  |