Using the Wisconsin breast cancer diagnostic data set for predictive analysis

Yagmur Onay (04-09-2022)

Attribute Information:

- **1) ID number**
- **2) Diagnosis (M = malignant, B = benign)**

-3-32.Ten real-valued features are computed for each cell nucleus:

- **a) radius (mean of distances from center to points on the perimeter)**
- **b) texture (standard deviation of gray-scale values)**
- **c) perimeter**
- **d) area**
- **e) smoothness (local variation in radius lengths)**
- **f) compactness (perimeter^2 / area - 1.0)**
- **g). concavity (severity of concave portions of the contour)**
- **h). concave points (number of concave portions of the contour)**
- **i). symmetry**
- **j). fractal dimension ("coastline approximation" - 1)**

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

Load Libraries

In [60]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

%matplotlib inline
import matplotlib.pyplot as plt # side-stepping mpl backend
import matplotlib.gridspec as gridspec # subplots
import mpld3 as mpl

# import models from scikit learn module
from sklearn.model_selection import train_test_split,KFold # For K-fold cross validation
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import metrics

import keras

import tensorflow as tf
```

Load the data

In [61]:

```
df = pd.read_csv("../input/data.csv",header = 0)
df.head()
```

Clean and prepare data

In [62]:

```
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(df)
```

In [63]:

```
# unique values in the column diagnosis are 'M' for malignant and 'B' for benign.
df.diagnosis.unique()
```

In [64]:

```
# convert values in  diagnosis column to numerical: map Ms to ones and Bs to zeros
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
df.head()
```

Explore data

In [65]:

```
df.describe()
```

In [66]:

```
plt.hist(df['diagnosis'])
plt.title('Diagnosis (M=1 , B=0)')
plt.show()
```

There are about 350 malignant and 220 benign tumor cell data.

nucleus features vs diagnosis

In [67]:

```
features_mean=list(df.columns[1:11])

# split dataframe into two based on diagnosis
dfM=df[df['diagnosis'] ==1]
dfB=df[df['diagnosis'] ==0]
```

In [68]:

```
#Stack the data
plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(df[features_mean[idx]]) - min(df[features_mean[idx]]))/50
    ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]], bins=np.arange(min(df[feat
ures_mean[idx]]), max(df[features_mean[idx]]) + binwidth, binwidth) , alpha=0.5,stacked=T
rue, density = True, label=['M','B'],color=['r','g'])
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()
```

Observations

1. **Mean values of cell radius, perimeter, area, compactness, concavity and concave points can be used in classification of the cancer. Larger values of these parameters tends to show a correlation with malignant tumors.**

2. **Mean values of texture, smoothness, symmetry or fractual dimension don't show a particular preference of one diagnosis over the other. In any of the histograms there are no noticeable large outliers that require further cleanup.**

Creating a test set and a training set

Do a simple 80:20 split to create a training data set and a test data set.We won't touch the test set till the final evaluation of the traineed model.

In [69]:

```
traindf, testdf = train_test_split(df, test_size = 0.2)
```

Model Classification

Here we are going to build a classification model and evaluate its performance using the test set.

In [70]:

```
# Generic function for making a classification model and assesing the performance.
def classification_model(model, traindf, predictors, outcome):
    #Perform k-fold cross-validation with 5 folds
    kf = KFold(n_splits=5)
    error = []
    for train, val in kf.split(traindf):

        # Training x
        train_predictors = (traindf[predictors].iloc[train,:])
        # Training y
        train_target = traindf[outcome].iloc[train]
        # Val x
        val_predictors = (traindf[predictors].iloc[val,:])
        # Val y
        val_target = traindf[outcome].iloc[val]

        # Training the algorithm using the predictors and target.
        model.fit(train_predictors, train_target)

        # Print error from each cross-validation run
        score = model.score(val_predictors, val_target)
        print("Cross-Validation Score from current fold: %s" % "{0:.3%}".format(score))

        # Record error from each cross-validation run
        error.append(model.score(val_predictors, val_target))

    #Make predictions on training set:
    predictions = model.predict(traindf[predictors])
    #Print accuracy
    accuracy = metrics.accuracy_score(predictions,traindf[outcome])
    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
    print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))
```

Logistic Regression model

Logistic regression is widely used for classification of discrete data. In this case we will use it for binary (1,0) classification.

Based on the observations in the histogram plots, we can reasonably hypothesize that the cancer diagnosis depends on the mean cell radius, mean perimeter, mean area, mean compactness, mean concavity and mean concave points. We can then perform a logistic regression analysis using those features as follows:

In [71]:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave p
oints_mean']
outcome_var='diagnosis'
model=LogisticRegression()
classification_model(model,traindf,predictor_var,outcome_var)
```

In [72]:

```
# Make predictions on test set:
predictions = model.predict(testdf[predictor_var])
# Print accuracy
accuracy = metrics.accuracy_score(predictions,testdf[outcome_var])
print("Logistic Regression Test Accuracy : %s" % "{0:.3%}".format(accuracy))
```

Can we do better with another model?

Decision Tree Model

In [73]:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave p
oints_mean']
model = DecisionTreeClassifier()
classification_model(model,traindf,predictor_var,outcome_var)
```

Accuracy is improved but the model seems to suffer with validation scores. We are probably over-fitting.

In [74]:

```
# Make predictions on test set:
predictions = model.predict(testdf[predictor_var])
# Print accuracy
accuracy = metrics.accuracy_score(predictions,testdf[outcome_var])
print("Decision Tree Test Accuracy : %s" % "{0:.3%}".format(accuracy))
```

Random Forest

In [75]:

```
predictor_var = ['radius_mean','perimeter_mean','area_mean','compactness_mean','concave p
oints_mean']
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_f
eatures=2)
classification_model(model,traindf,predictor_var,outcome_var)
```

In [76]:

```
# Use all the features of the nucleus
predictor_var = features_mean
model = RandomForestClassifier(n_estimators=100,min_samples_split=25, max_depth=7, max_f
eatures=2)
classification_model(model, traindf,predictor_var,outcome_var)
```

Using all nucleus features improve results. Training set predictions have an accuracy of 95.165% and cross-validation score of 93.626%. Now let's evaluate the model accuracy on the test set.

In [77]:

```
# Make predictions on test set:
predictions = model.predict(testdf[predictor_var])
# Print accuracy
accuracy = metrics.accuracy_score(predictions,testdf[outcome_var])
print("Random Forest Test Accuracy : %s" % "{0:.3%}".format(accuracy))
```

Random forest test accuracy is 92.982%. An advantage with Random Forest is that it returns a feature importance matrix which can be used to select features. So lets select the top 5 features and use them as

predictors for exploration.

In [78]:

```
# Create a series with feature importances:
featimp = pd.Series(model.feature_importances_, index=predictor_var).sort_values(ascendi
ng=False)
print(featimp)
```

In [79]:

```
# Using top 5 features
predictor_var = ['concave points_mean','area_mean','radius_mean','perimeter_mean','concav
ity_mean',]
model = RandomForestClassifier(n_estimators=100, min_samples_split=25, max_depth=7, max_
features=2)
classification_model(model,traindf,predictor_var,outcome_var)
```

Using the top 5 features returns worse accuracy and validation scores (94.286% and 92.088%) than when we used all nucleaus feauters above (95.165% and 93.626%).

ANN

In [81]:

```
# Set the variables for training
max_epochs = 200
batch_size= 8
patience = 10

# Early stopping callback:
# This callback will stop the training when there is no improvement in
# the loss for given (patience) consecutive epochs.
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=patience)

def train_and_cross_validate_model(model, traindf, predictors, outcome):
    #Perform k-fold cross-validation with 5 folds
    kf = KFold(n_splits=5)
    error = []
    for train, val in kf.split(traindf):

        # Training x
        train_predictors = (traindf[predictors].iloc[train,:])
        # Training y
        train_target = traindf[outcome].iloc[train]
        # Val x
        val_predictors = (traindf[predictors].iloc[val,:])
        # Val y
        val_target = traindf[outcome].iloc[val]

        # Training the algorithm using the predictors and target.
        model.fit(
            train_predictors,
            train_target,
            epochs=max_epochs,
            batch_size=batch_size,
            validation_data=(val_predictors, val_target),
            callbacks=[callback]
        )

        # Print error from each cross-validation run
        scores = model.evaluate(val_predictors, val_target, verbose=0)
        # scores[0] is loss whereas scores[1] is accuracy
        print("Cross-Validation Score from current fold: %s" % "{0:.3%}".format(scores[1
]))

        # Record error from each cross-validation run
        error.append(scores[1])
    # Make predictions on training set and calculate accuracy
    loss, accuracy = model.evaluate(traindf[predictors], traindf[outcome])

    print("Accuracy : %s" % "{0:.3%}".format(accuracy))
    print("Score : %s" % "{0:.3%}".format(np.mean(error)))
```

We will create a Neural Network that has one hidden layer:

- **Input layer has 10 nodes for all nucleus features. The activation function for this layer is Rectified Linear Units (ReLU). ReLU is a half rectified function; that is, for all the inputs less than 0 the value is 0 while for anything positive the value is retained.**
- **Hidden layer consists of 8 nodes. The activation function for this layer is ReLU.**
- **Output layer has 1 node. It uses the sigmoid activation function that will squeeze all the values between 0 and 1 into the form of a sigmoid curve.One output unit is used since for each record values in X, a probability will be predicted. If it is high (>0.9) than the tumor is definitely malignant. If it is less (<0.2) then it is definitely benign.**

In [82]:

```
# use all features
predictor_var = features_mean
feature_size = len(predictor_var)

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(feature_size,)),
    keras.layers.Dense(5, activation=tf.nn.relu),
    keras.layers.Dense(1, activation=tf.nn.sigmoid),
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.summary()
```

Model has been trained with 5-fold cross validation. By using corss valdiation, whole training set has been used on top of early stopping based on model prediction accuracy on the validation set. Training set prediction accuracy is 89.231% and mean validation set prediction accuracy is 89.011%. Now, let's evaluate the NN's performance on our test set.

In [84]:

```
# test
test_loss, test_acc = model.evaluate(testdf[predictor_var], testdf[outcome_var])

print("NN Test Accuracy : %s" % "{0:.3%}".format(test_acc))
```

In [86]:

```
model.save("breast_cancer_nn.h5")
```

Conclusion

The best model to be used for diagnosing breast cancer as found in this analysis is the NN based on holdout test results. Using all nucleaus features for training yields better results than as with less features.

The prediction accuracy for the test data set using the:

1. **logistic regression model is 90.351%.**
2. **random forest model is 92.982%.**
3. **decision tree model is 92.982%.**
4. **NN is 93.860%**