Contents

BREAST CANCER: A NEURAL NETWORK BASED CLASSIFIER

Barbara Corradini

February, 2019

Introduction

Breast cancer is cancer that develops from breast tissue. Although it can affect both men and women, it is the most common type of cancer in female population, and it is due to an uncontrolled growth of some cells of the mammary gland, which turn into malignant cells.
In order to detect and diagnose breast cancer, different approaches have been prompted, which allow to study cancer under different points of view:

- Clinical, is the first stage of diagnosis, consisting of a breast exam through palpation;
- Radiologic, uses imaging techniques such as ultrasonography or mammography;
- Cytological, is an approach which uses cell samples, picked up from breast tissue with a fine needle and syringe, to analyse their structure and appearance;
- Hystological, through a needle biopsy - i.e. pieces of breast tissue are picked up from a suspicious area to be analyzed.

Although The Hystological method is considered as more detailed with respect to the Cytological one, lots of medical papers and journals point out that the latter, since it uses different parameters, provides just different (but not incorrect) information, so it is declared still valid.

Chapter 1

Dataset definition

The "Breast Cancer Wisconsin (Diagnostic) Data Set" (from UCI Machine Learning Repository[1]) was used to realize a neural network classifier for breast cancer.
The dataset is a collection of features observed and measured from digitalized images of cells acquired via Cytological procedure (Fine Needle Aspirate or FNA) of a breast mass.
Main characteristics of the dataset are:

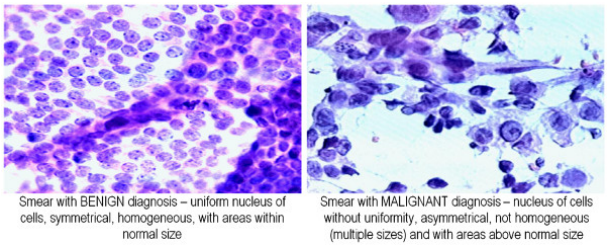

Figure 1.1: Image of cell nuclei after FNA

- It has 569 samples (rows) with 32 attributes (columns) each.
- Attributes:
  1. Id (identification number of the cell sample, not necessary for training);
  2. Diagnosis ($\in \{M, B\}$, contains the classification of malignant or benignant for each instance);

The following attributes are referred to the cell nuclei:

[1]Link: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

3. Radius (mean of distances from center to points on the perimeter);
4. Texture (standard deviation of gray-scale values);
5. Perimeter;
6. Area;
7. Smoothness (local variation in radius lengths);
8. Compactness ($Perimeter^2/Area - 1.0$);
9. Concavity (severity of concave portions of the contour);
10. Concave points (number of concave portions of the contour);
11. Symmetry;
12. Fractal dimension (measures the nuclear border irregularity considering "$CoastlineApproximation$" $- 1$[2]);

All attributes from 3 to 12 are mentioned three times in the dataset, because their mean, standard error and worst value were calculated.

# concave points_me mean for number of concave portions of the contour	# symmetry_mean	# fractal_dimension_r mean for "coastline approximation" - 1	# radius_se standard error for the mean of distances from center to points on the perimeter	# texture_se standard error for standard deviation of gray-scale values
0.1471	0.2419	0.07871	1.095	0.9053
0.07017	0.1812	0.05667	0.5435	0.7339
0.1279	0.2069	0.05999	0.7456	0.7869
0.1052	0.2597	0.09744	0.4956	1.156
0.1043	0.1809	0.05883	0.7572	0.7813
0.08089	0.2087	0.07613	0.3345	0.8902
0.074	0.1794	0.05742	0.4467	0.7732
0.05985	0.2196	0.07451	0.5835	1.377
0.09353	0.235	0.07389	0.3063	1.002
0.08543	0.203	0.08243	0.2976	1.599
0.03323	0.1528	0.05697	0.3795	1.187
0.06606	0.1842	0.06082	0.5058	0.9849
0.1118	0.2397	0.078	0.9555	3.568

Figure 1.2: Dataset appearance

[2]Mandelbrot B. B. The Fractal Geometry of Nature W. H. Freeman and Company New York 1977.

Chapter 2

The code

2.1 Libraries

Keras library was used for the Neural Network implementation. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow or Theano[1].

```
import numpy as np # linear algebra
import pandas as pd # data processing
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
```

2.2 Preprocessing of data

The dataset was preprocessed in order to give to the Neural Network just numerical values: in particular, the column which gives the diagnosis was converted from char to int type and columns with NaN values were canceled (see code 2.1).

```
df = pd.read_csv("Breast_Cancer_Wisconsin_Dataset.csv", sep = ',
    ')
# replace M and B with int values (one-hot encoding)
one_hot_encoding = df.replace(['M','B'], [1,0])
# cancel NaN values
clean_dataset = one_hot_encoding.dropna(axis = 1)
```

Listing 2.1: Dataset cleaning

[1]Link to Keras website: https://keras.io/

Then, the dataset was splitted into training and test portion (70% and 30% of the total, respectively), and the Diagnosis column was separated from the dataset and put into the output variables (y_train, y_test).

```
# split into training and test set
train, test = train_test_split(clean_dataset.values, test_size =
    0.30)
y_train, y_test = train[:, 1], test[:, 1]
x_train, x_test = np.delete(train, [0,1], axis = 1), np.delete(
   test, [0,1], axis = 1)
```

Listing 2.2: Dataset is splitted into training and test set

2.3 Summary plots: accuracy and loss

After the training of the Neural Network, both accuracy and loss for training and test set are plotted.
This is a useful stage, since it allows to have a look at the behaviour of these parameters and have some information about the goodness of the Network.

```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Chapter 3

Neural Network models

3.1 Multi Layer Perceptron with adam optimizer

Neural Network is structured as follows (fig. 3.1):

- **Input layer**: 12 neurons with *tanh* as activation function;
- **Hidden layer**: 20 neurons with *relu* as activation function;
- **Output layer**: 1 neuron with *sigmoid* as activation function.

Moreover, the training set was splitted again, and its 20% was put into the validation set.

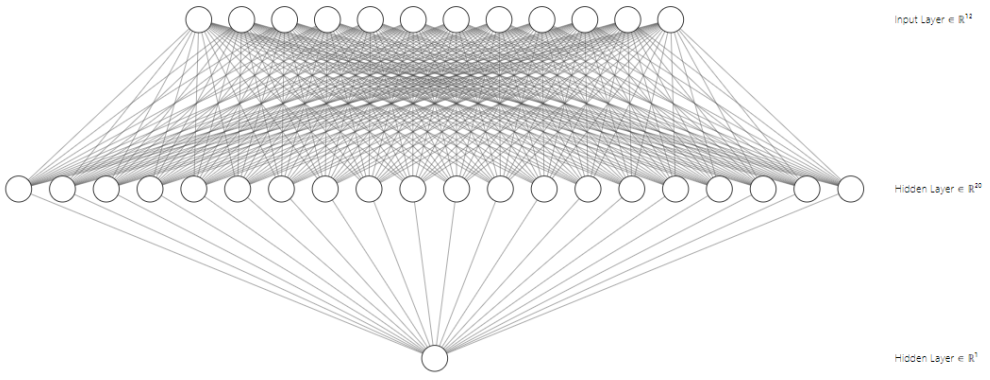

Figure 3.1: Neural Network architecture

```
# Fix the seed
np.random.seed(1)
# Create model
model = Sequential()
model.add(Dense(12, input_dim=x_train.shape[1], activation='tanh'))
model.add(Dense(20, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

3.1.1 Results

Results provided by this model, over 100 epochs, are (fig. 3.3):

- Accuracy: 92.96%
- Loss: 0.0747

```
# Fit the model
history = model.fit(x_train, y_train, validation_split = 0.20, epochs=100, batch_size=9)
# Evaluate the model
scores = model.evaluate(x_train, y_train)
# Print results
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print("\n%s: %.2f" % (model.metrics_names[0], scores[0]))
```

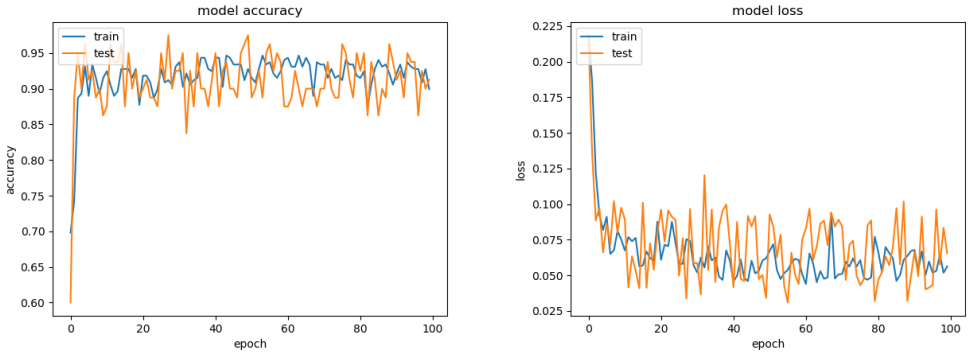

Figure 3.2: Accuracy and loss of the network

3.2 Multi Layer Perceptron with adam optimizer and two hidden layers

Neural Network is structured as follows (fig. 3.3):

- **Input layer**: 10 neurons with *tanh* as activation function;
- **First hidden layer**: 20 neurons with *relu* as activation function;
- **Second hidden layer**: 2 neurons with *relu* as activation function;
- **Output layer**: 1 neuron with *sigmoid* as activation function.

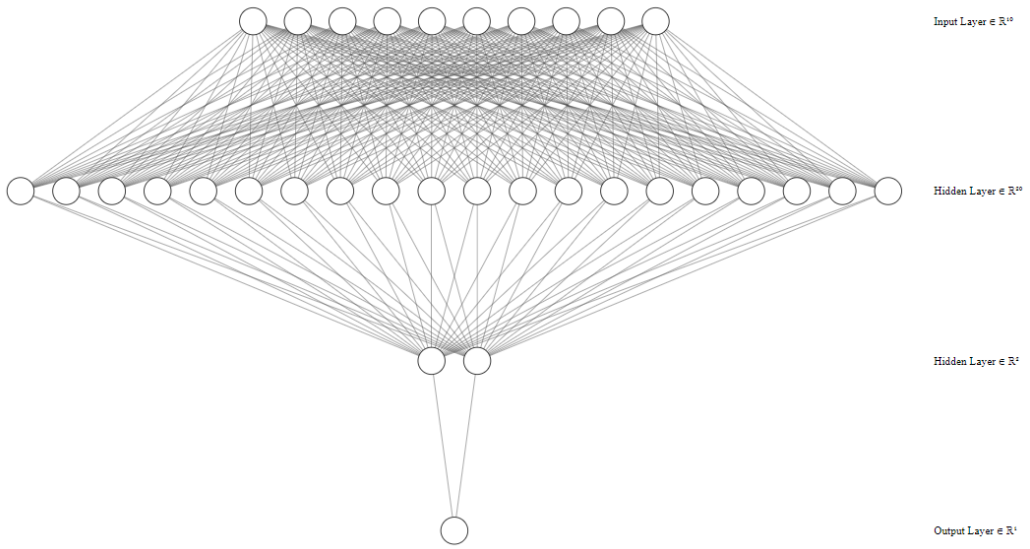

Figure 3.3: Neural Network architecture

```
# Fix the seed
np.random.seed(1)
# Create the model
model = Sequential()
model.add(Dense(10, input_dim=x_train.shape[1], activation='tanh
    '))
model.add(Dense(20, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='mean_squared_error', optimizer='adam',
    metrics=['accuracy'])
```

3.2.1 Results

Results provided by this model, over 100 epochs as before, are (fig. 3.4):

- Accuracy: 90.70%
- Loss: 0.1062

```
# Fit the model
history = model.fit(x_train, y_train, validation_split = 0.20,
    epochs=100, batch_size=9)
# Evaluate the model
scores = model.evaluate(x_train, y_train)
# Print results
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
print("\n%s: %.2f" % (model.metrics_names[0], scores[0]))
```

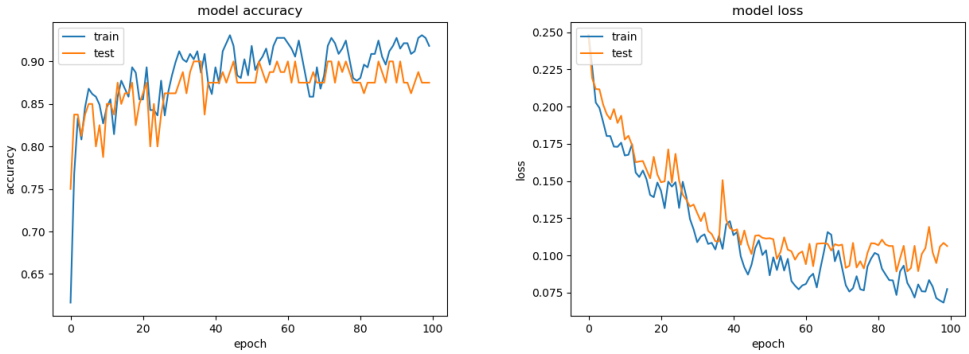

Figure 3.4: Accuracy and loss of the network

Chapter 4

An ensemble approach

4.1 What is the Random Forest algorithm?

Random forests is an ensemble algorithm: this means that it uses multiple learning algorithms to obtain better predictive performance than the one that could be obtained from any method taken alone. It can be used both for classification and regression.
Random Forest algorithm acts as follows:

1. creates decision trees on randomly selected data samples;
2. gets prediction from each tree;
3. selects the best solution by means of voting.

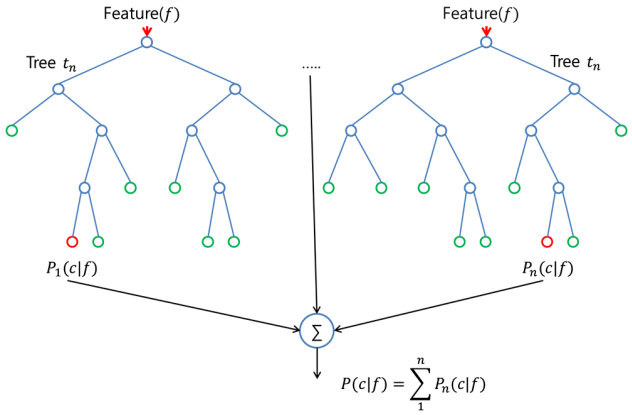

Figure 4.1: Decision trees in Random Forest algorithm

Random Forest algorithm was used for implementing another Breast Cancer classifier.

The output parameters taken under consideration are the accuracy and the bar plot that puts in relationship the attributes (features) and their meaningfulness in the classification process.

4.2 The code

After importing the *sklearn* library, the first part of the code, in which the dataset is preprocessed, is the same as in the previous chapter.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
# Prepare dataset for training
df = pd.read_csv("Breast_Cancer_Wisconsin_Dataset.csv", sep = ',
    ')
a = df.replace(['M','B'], [1,0])
a = a.dropna(axis = 1)
train, test = train_test_split(a.values, test_size = 0.30)
y_train, y_test = train[:, 1], test[:, 1]
x_train, x_test = np.delete(train, [0,1], axis = 1), np.delete(
    test, [0,1], axis = 1)
```

The classifier was initialized with 100 estimators and then trained.

```
# Create the classifier
classifier=RandomForestClassifier(n_estimators=100)
# Train the model
classifier.fit(x_train,y_train)
y_pred=classifier.predict(x_test)
```

4.2.1 Results

The results obtained by the Random Forest algorithm are:

- An higher accuracy, of 96.49%;
- A bar plot (fig. 4.2) that, surprisingly enough, reveals that the most discriminant features in the classification problem are the worst ones.

```
# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
b = list(a.drop(['id','diagnosis'], axis=1))
```

```
feature_importance = pd.Series(classifier.feature_importances_,
    index=b).sort_values(ascending=False)
# Creating a bar plot
sns.barplot(x=feature_importance, y=feature_importance.index)
# Add labels to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Important Features in Breast Cancer Classification")
plt.legend()
plt.show()
```

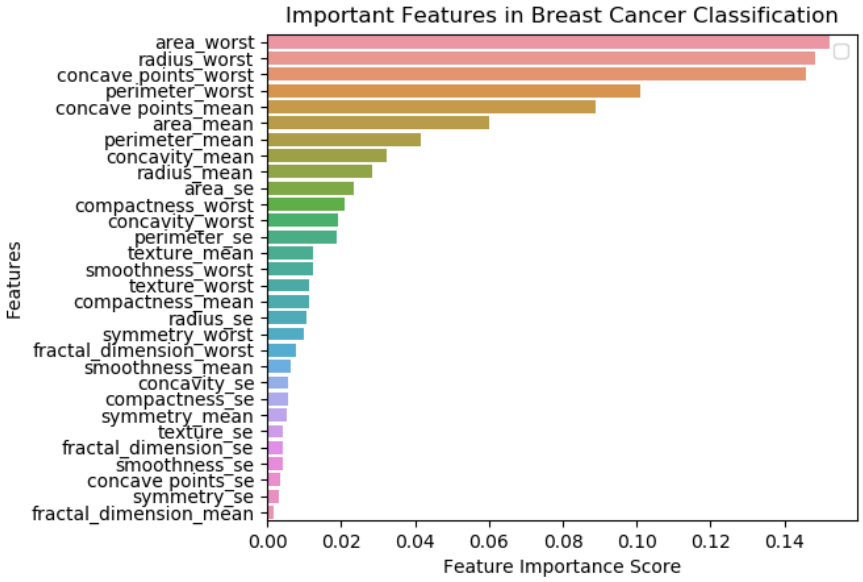

Figure 4.2: Important features detection using random forest