Faculty of Engineering and Natural Sciences
Department of Information Technologies
Data science
Introduction to Data Science

Comparing performances of logistic regression, k-nearest neighbor and random forest for predicting breast cancer

Student: Biševac Nežla
Professor: Assist. Prof. Dr. Zerina Mašetić

Sarajevo, 11 January, 2021

ABSTRACT

Two most common methods in supervised learning are classification and regression. We use regression when we want to predict the value of a variable based on the value of another variable where the value we want to predict is called a dependent variable.
Unlike regression, where you predict a continuous number, you use classification to predict a category. Classification models include linear models like Logistic Regression, SVM (Support Vector Machines) and nonlinear models like K-NN, Kernel SVM, Naive Bayes, Decision Tree Classification and Random Forest Classification. In this paper, we are comparing results of Logistic Regression, Random Forest and KNN on the breast cancer dataset. We will predict what are the chances that a breast cancer is malignant or benign. We also used k-Fold Cross Validation to estimate the skill of a machine learning model on unseen data.

1.INTRODUCTION

Breast cancer is a group of diseases in which cells in breast tissue change and divide uncontrolled, typically resulting in a lump or mass. According to statistics, the incidence of breast cancer has risen dramatically during the last four decades. There are two important things for the outcome of a cancer: whether it's benign or malignant. If it's benign then it means that is not cancerous, it does not spread to other parts of the body.
If it's malignant, then it refers to cancer cells that can invade and kill nearby tissue and spread to other parts of the person's body.

As machine learning and data science are starting to be adopted as a tool in healthcare applications, the industry is slowly pushing the boundaries on what it can do. Its primary function will most likely involve data analysis based on the fact that each patient generates large volumes of health data such as X-ray results, vaccinations, blood samples, vital signs, DNA sequences, current medications, other past medical history, and much more.[1]

One application of machine learning in a healthcare context is digital diagnosis. ML can detect patterns of certain diseases within patient electronic healthcare records and inform clinicians of any anomalies. In this sense, the artificial intelligence technique can be compared to a second pair of eyes that can evaluate patient health based on the knowledge extracted from big data sets by summarizing millions of observations of diseases that a patient could possibly have.

To illustrate just how useful machine learning as a medical diagnosis tool can be, we examined its use in breast cancer detection using a publicly available UCI's machine learning repository.

Solving a problem with machine learning often involves many iterative experiments meant to find the best model for solving the problem by further tuning the model. Given that there are many machine learning algorithms and different neural network architectures, a researcher (based on his/her experience, knowledge and trusting his/her intuition) will select the most promising model to set up the first experiment.

In our example, we decided to choose linear classification model Logistic Regression, and two nonlinear models: K-NN (K-Nearest Neighbor) and Random Forest Classification. Each model consists of the following steps:

1. Importing the libraries
2. Importing the dataset
3. Splitting the dataset into the Training and Test set
4. Training the Classification model on the training set
5. Making a Confusion matrix
6. Computing the accuracy using k-Fold Cross Validation

2.METHODOLOGY

As it was mentioned above, we will compare four classification models and their results on our dataset. First three steps are the same for all models:

- Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

- Importing the dataset

```
dataset = pd.read_csv('breast_cancer.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

  X is our matrix of features and y is our dependent variable vector. Since the first feature is Sample code number and it doesn't affect the outcome, we excluded it as a feature in X, which now contains columns 2-9 from our dataset and. Dependent variable vector contains the last column.

- Splitting the dataset into the Training and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

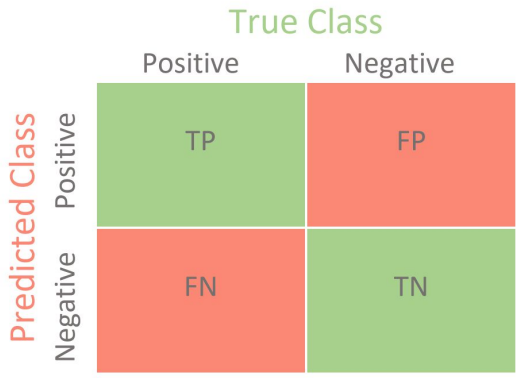
We used *train_test_split* procedure to split the dataset into the training and test set. We will use a training set to fit and train the model, to predict the dependent variable and compute the accuracy with k-Fold Cross Validation. Test sets (combined with training sets) will be used for creating the confusion matrix and for accuracy score. Also both sets can be used for visualising the results. For splitting the dataset we used the most common distribution, 20:80: test set will contain 20% of the dataset, while the remaining 80% of the data will be stored in the training set.

- Training the Classification model on the training set
  This step will be described in the next part because it's different for every model.
- Making a Confusion matrix

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
ac=accuracy_score(y_test, y_pred)
print(ac)
```

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Here, we used it to see how accurate are predictions from the X_test, because y_pred = classifier.predict(X_test) compared to real results from the y_test set.
Confusion matrix is 2x2 matrix and has four values:

- TP - values predicted as positive and they are positive
- TN - values predicted as negative and they are negative
- FP - values predicted as positive, but actually they are negative
- FN - values predicted as negative, but actually they are positive



- Computing the accuracy using k-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

Here, we used cross_val_score procedure to compute the accuracy of the model and standard deviation.
cv represents parameter that determines number of sections/folds where each fold is used as a testing set at some point. Here, the dataset is is split into 10 folds. In the first iteration, the first fold is used to test the model and the rest are used to train the model. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 10 folds have been used as the testing set.
Standard deviation describes how spread out the values are. A low **standard deviation** means that most of the numbers are close to the mean (average) value. A high **standard deviation** means that the values are spread out over a wider range.

2.1 LOGISTIC REGRESSION

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model. Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

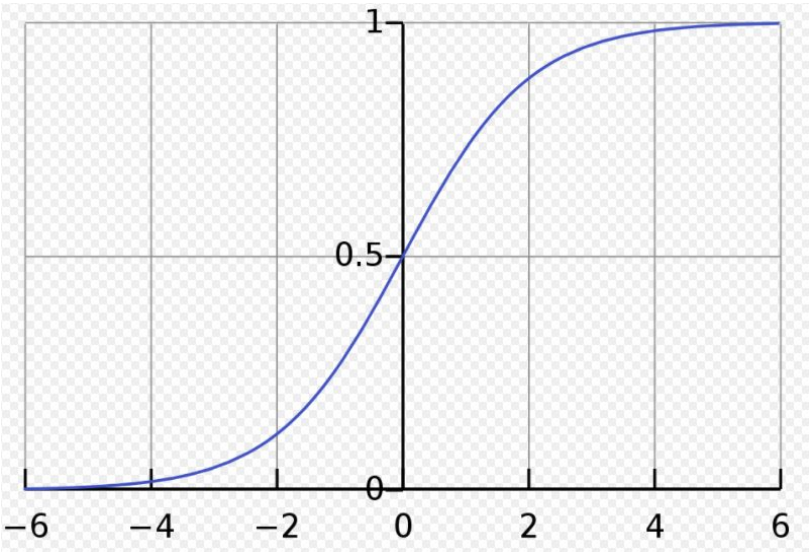
Function that represents logistic regression is an example of sigmoid function, logistic function that is described by formula (red rectangle):

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1+e^{-z}}$$

and shown in the figure:



The function g(x), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

$h_\theta(x)$ will give us the probability that our output is 1. For example, $h_\theta(x) = 0.7$ gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%). [2]

$$h_\theta(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$
$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$

- Training the Logistic Regression model on our dataset

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

We applied Logistic regression on parameter classifier, and then we fitted classifier on our dataset (on training set).

- Confusion matrix results

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
ac=accuracy_score(y_test, y_pred)
```

```
[[84  3]
 [ 3 47]]
```

In red rectangle we have the confusion matrix.
Number of predicted and true classes are: TP = 84, FP = 3, FN = 3, TN = 47.

- Computing the accuracy with k-Fold Cross validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 96.70 %
Standard Deviation: 1.97 %
```

When training logistic regression, we get the accuracy of 96.70% and standard deviation of 1.97%. Accuracy is greater than 95% which means that we didn't make a bad choice for choosing Logistic regression method and our standard deviation is not high, but not low either: 1.97% and it tells us that maybe another method could be more suitable for our dataset.

2.2 K-NN (K-NEAREST NEIGHBOR)

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

Just like almost everything else, KNN works because of the deeply rooted mathematical theories it uses. When implementing KNN, the first step is to transform data points into feature vectors, or their mathematical value. The algorithm then works by finding the distance between the mathematical values of these points. The most common way to find this distance is the Euclidean distance. KNN runs this formula to compute the distance

between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities.[3]

Steps of the algorithm:

1. Choose the number of K neighbor
2. Take the K nearest neighbor of the new data point, according to the Euclidean distance
3. Among these K neighbors, count the number of data points in each category
4. Assign the new data point to the category where you counted the most neighbors
5. Your model is ready

- Training K-NN model on our dataset

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

Here, we used the KNeighborsClassifier class to create the instance of KNN model. KNeighborsClassifier has three parameters: n_neighbors as a number of neighbors, metric for calculating the distance between the points, 'minkowski' is a key word for calculating the Euclidean distance and the last parameter, p, is a power parameter that was used in metric parameter.

- Confusion matrix results

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[84  3]
 [ 1 49]]
```

In red rectangle we have the confusion matrix.
Number of predicted and true classes are: TP = 84, FP = 3, FN = 1, TN = 49.

- Computing the accuracy with k-Fold Cross validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 97.44 %
Standard Deviation: 1.85 %
```

When training lKNN, we get the accuracy of 97.44% and standard deviation of 1.85%. Accuracy is greater than 95% which means that we didn't make a bad choice for choosing KNN method and our standard deviation is: 1.85%.

2.3 RANDOM FOREST CLASSIFICATION

Random forest can be used for regression too.

It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random forests has a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.[4]

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.[5]

The best way to understand the algorithm is to follow the steps:

1. Pick at random K data points from the training set.
2. Build the decision tree associated to these K data points.
3. Choose the number Ntree of trees you want to build and repeat steps 1 and 2.

4. For each data point, make each one of your Ntree trees predict the category to which the data point belongs, and assign the new data point to the category that wins the majority vote.

- Training Random Forest model on our dataset

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

Here, we used the RandomForestClassifier class to create the instance of RandomForestClassifier model. RandomForestClassifier has three parameters: n_neighbors as a number of trees in the forest, criterion which is the function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain and random state

- Confusion matrix results

```
: from sklearn.metrics import confusion_matrix, accuracy_score
  cm = confusion_matrix(y_test, y_pred)
  print(cm)
  accuracy_score(y_test, y_pred)
```

```
[[83  4]
 [ 3 47]]
```

```
: 0.948905109489051
```

In red rectangle we have the confusion matrix.
Number of predicted and true classes are: TP = 83, FP = 4, FN = 3, TN = 47.

- Computing the accuracy with k-Fold Cross validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X, y = y, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 96.50 %
Standard Deviation: 2.27 %
```

When training Random Forest, we get the accuracy of 96.50% and standard deviation of 2.27%. Accuracy is greater than 95% which means that we didn’t make a bad choice for choosing the Random Forest method.

3. CONCLUSION

In this paper, we explained the use of these three algorithms, how we use them and what are their benefits. We trained these algorithms on a dataset that has 683 rows. Each of them has different results. In the following table we have summarized the final results:

Method	**Accuracy**	**Standard deviation**
Logistic Regression	96.70 %	1.97%
KNN	97.44 %	1.85%
Random forest	96.50 %	2.27%

We can see that, according to accuracy and standard deviation, the best result gives us KNN. KNN is a fast algorithm that gives great results when we use it on the small dataset, which was the case here.

Now, lets see the table with pros and cons for the each model:

Method	**Pros**	**Cons**
Logistic Regression	Probabilistic approach, gives information about statistical significance of features	The logistic regression assumptions
KNN	Simple to understand, fast and efficient	Need to choose the number of neighbors K
Random forest	Powerful and accurate, good performance on many problems, including nonlinear	No interpretability, overfitting can easily occur, need to choose the number of trees

[1] https://www.macadamian.com/learn/a-practical-application-of-machine-learning-in-medicine/

[2] https://www.coursera.org/learn/machine-learning/supplement/AqSH6/hypothesis-representation

[3] https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[4] https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[5] https://towardsdatascience.com/understanding-random-forest-58381e0602d2