

**Using Machine Learning (K nearest neighbors classifier (KNN)) to detect breast cancer**

**Introduction**

Machine Learning algorithms have been used for several applications. One of them in the medical field. This project applies a Machine Learning algorithm (Python in Jupyter Notebooks) to perform tumor classifying according to the tumor characteristics. [1]

**Part I: the dataset**

The following libraries were imported:

*# importing libraries*

**import** sklearn

**import** pandas **as** pd

**import** numpy **as** np

**from** sklearn.datasets **import** load\_breast\_cancer

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.metrics **import** accuracy\_score

**import** matplotlib.pyplot **as** plt

The dataset was loaded from scikit-learn databases using the following code: [2]

*# load a toy dataframe from https://scikit-learn.org/stable/datasets/toy\_dataset.html*

cancer **=** load\_breast\_cancer()

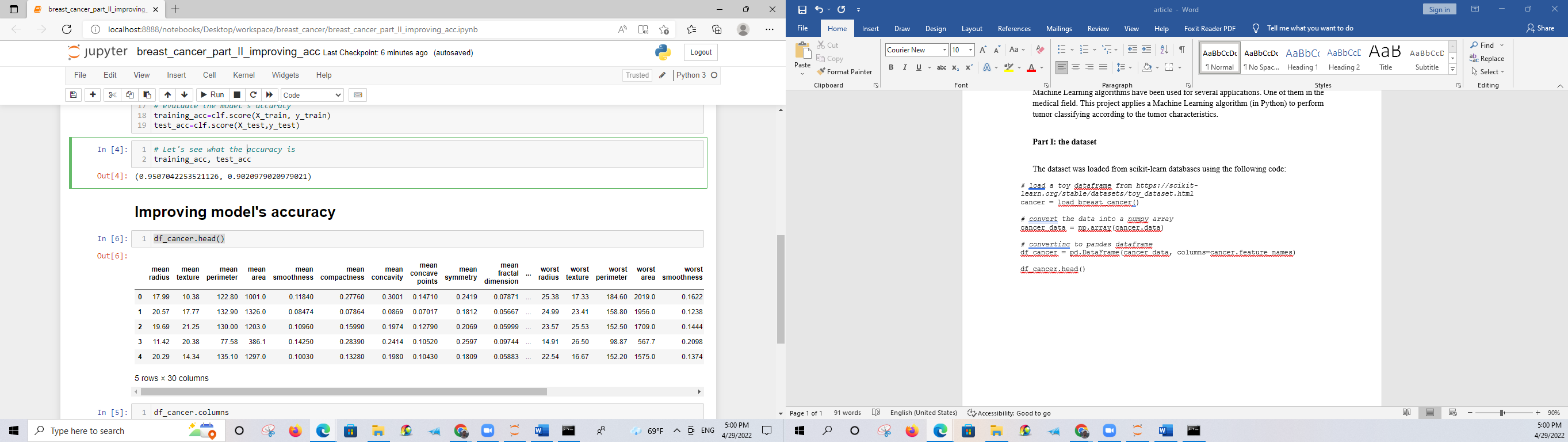
*# convert the data into a numpy array*

cancer\_data **=** np**.**array(cancer**.**data)

*# converting to pandas dataframe*

df\_cancer **=** pd**.**DataFrame(cancer\_data, columns**=**cancer**.**feature\_names)

df\_cancer**.**head()



df\_cancer**.**columns

Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',

'mean smoothness', 'mean compactness', 'mean concavity',

'mean concave points', 'mean symmetry', 'mean fractal dimension',

'radius error', 'texture error', 'perimeter error', 'area error',

'smoothness error', 'compactness error', 'concavity error',

'concave points error', 'symmetry error', 'fractal dimension error',

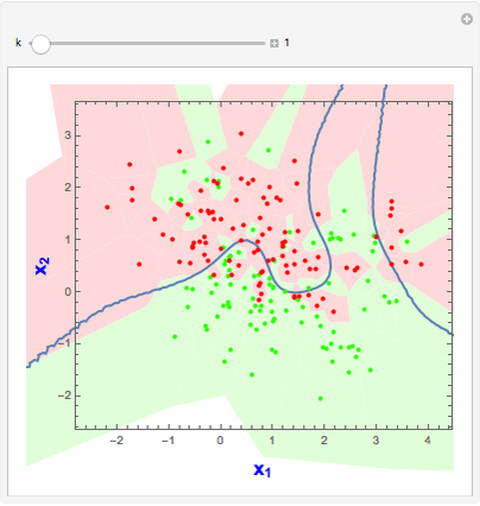
'worst radius', 'worst texture', 'worst perimeter', 'worst area',

'worst smoothness', 'worst compactness', 'worst concavity',

'worst concave points', 'worst symmetry', 'worst fractal dimension']

**Part II: the model**

Then the model was built using the scikit-learn algorithm: KNN.



*# split the data*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(cancer**.**data, cancer**.**target, stratify**=**cancer**.**target)

*# building and training the model*

clf **=** KNeighborsClassifier(n\_neighbors **=** 5)

clf**.**fit(X\_train,y\_train)

**Part III: evaluating the model**

The model’s accuracy was evaluated as following:

*# evaluate the model's accuracy*

training\_acc**=**clf**.**score(X\_train, y\_train)

test\_acc**=**clf**.**score(X\_test,y\_test)

*# Let's see what the accuracy is*

training\_acc, test\_acc

(0.9436619718309859, 0.958041958041958)

We can see that the model showed a pretty good accuracy in the test set, almost 96%.

**Part IV: improving the model performance**

Let’s see how we can improve the accuracy even further.

*# let's see the model's parameters*

clf**.**get\_params()

{'algorithm': 'auto',

'leaf\_size': 30,

'metric': 'minkowski',

'metric\_params': None,

'n\_jobs': None,

'n\_neighbors': 5,

'p': 2,

'weights': 'uniform'}

One of the most important parameters of this model is the 'n\_neighbors'. So, let’s see if we can improve the model accuracy iteration over several values of this parameter.

*# let's test several parameters*

n\_neighbors **=** [1,5,10,20,50,100]

test\_accuracy **=** {}

train\_accuracy **=** {}

**for** n **in** n\_neighbors:

clf **=** KNeighborsClassifier(n\_neighbors **=** n)

clf**.**fit(X\_train,y\_train)

test\_acc**=**clf**.**score(X\_test,y\_test)

training\_acc**=**clf**.**score(X\_train, y\_train)

test\_accuracy[n] **=** test\_acc

train\_accuracy[n] **=** training\_acc

*# let's see the test accuracy*

**for** k, v **in** test\_accuracy**.**items():

print("test accuracy for {} n\_neighbors is {} %"**.**format(k,round(v**\***100,3)))

test accuracy for 1 n\_neighbors is 94.406 %

test accuracy for 5 n\_neighbors is 95.804 %

test accuracy for 10 n\_neighbors is 95.105 %

test accuracy for 20 n\_neighbors is 96.503 %

test accuracy for 50 n\_neighbors is 93.007 %

test accuracy for 100 n\_neighbors is 93.706 %

*# let's see the train accuracy*

**for** k, v **in** train\_accuracy**.**items():

print("train accuracy for {} n\_neighbors is {} %"**.**format(k,round(v**\***100,3)))

train accuracy for 1 n\_neighbors is 100.0 %

train accuracy for 5 n\_neighbors is 94.366 %

train accuracy for 10 n\_neighbors is 93.192 %

train accuracy for 20 n\_neighbors is 92.488 %

train accuracy for 50 n\_neighbors is 90.845 %

train accuracy for 100 n\_neighbors is 90.376 %

**Conclusions**

We can see that the highest accuracy was achieved with 20 n\_neighbors and we can also see that this model overfits with fewer n\_neighbors, showing an accuracy of 100% with 1 n\_neighbor.

The model performance for this task was quite good.

The model accuracy improved up to 20 n\_neighbors and then start declining.

**References**

1. MÜLLER, A. C.; GUIDO, S. **Introduction to Machine Learning with Python**. [s.l.] O REILLY, 2016.

2. PEDREGOSA, F et. al.. Scikit-learn: Machine Learning in Python. **JMLR**, v. 12, n. 85, p. 2825−2830, 2011.