# CS425 Project 3(Report)
# k-Nearest-Neighbors and Decision Trees

Ksenia Burova

November $6^{th}$, 2017

**Abstract:** In machine learning, k-Nearest-Neighbors (kNN) algorithm is one of the nonparametric estimation methods, and in nonparametric methods we assume that similar data inputs have similar data outputs. In this project, we were given data file with different attributes that correspond to breast cancer diagnosis. The goal of the project was to run both, kNN and Decision Trees algorithms, to see how well they perform in predicting a tumor being benign or malignant. There were several performance metrics to be used to analyze results. Some of those metrics were displayed in plots.

## I. Project Data.

In this project, we were given a file *breast-cancer-wisconsin.data* which had 11 attributes:

1. Sample code number: id number
2. Uniformity of cell size: 1 - 10
3. Uniformity of Cell Shape: 1 - 10
4. Marginal Adhesion: 1 - 10
5. Single Epithelial Cell Size: 1 - 10
6. Bare Nuclei: 1 - 10
7. Bland Chromatin: 1 - 10
8. Normal Nucleoli: 1 - 10
9. Mitoses: 1 - 10
10. Class: (2 for benign, 4 for malignant)

I removed a column with **id number** since it is not relevant in predicting tumor type. There were also some data with missing values. My choice was not to predict those values, instead, I excluded that data from the data set. I decided to use such method to avoid incorrect prediction that may influence experiment results.

I've placed all continuous values for this data into multidimensional array called **dataFeatures**, and kept classes in one-dimensional array called **dataLabels** for simplicity of implementation.

During experiments, data was split into 3 sets: training, validation and testing. First I've split the whole data into training and testing data sets with 70/30 proportion rule. Then training data was divided into training and validation parts with 70/30 proportion rule as well for cross validation purposes.

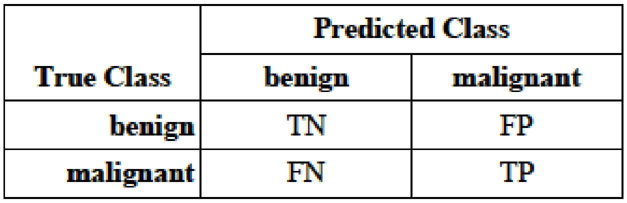
## II. Tools and Program.

I've used **python** programming language, **numpy** library for arithmetic, **sklearn** library for splitting my data into training, validation and testing datasets, and **matplotlib** library to built plots for this project. I have a program named **knn-dt.py** , that includes implementation of both parts of the project. I've split each algorithm into several functions for efficiency, and in the very end of my program I do main calls to run experiments and build plots.

## III. Performance metrics.

To evaluate performance of classification algorithms, we have to compare correct classifications with our classifications on evaluation data sets after algorithm is run. We are going to calculate and look at the following values:

- Confusion Matrix



| True Class | Predicted Class | |
|---|---|---|
| | benign | malignant |
| benign | TN | FP |
| malignant | FN | TP |

  where TP = # true positives, TN = # true negatives, FP = # false positives, FN = # false negatives.

- Accuracy = (TN + TP) / (TN + TP + FN + FP)
  **Accuracy** tells us what percent of data was classified correctly.

- TPR (true positive rate, recall, or sensitivity) = TP / (TP + FN)
  **Sensitivity** shows us rate at which true positives are not missed/overlooked

- PPV (positive predictive value or precision) = TP / (TP + FP)
  **Precision** is basically a fraction of relevant positive data from all the positive data we have, meaning have many of those diagnosed with bad cancer really have it.

- TNR (true negative rate or specificity) = TN / (TN + FP)
  **Specificity** is a rate that shows how well we can distinguish data that is truly negative,

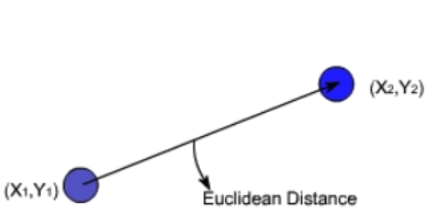
meaning that people have good tumor.

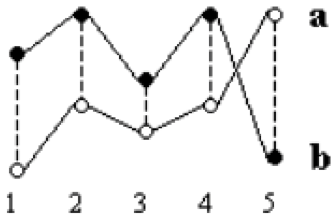
- F Score = PPV * TPR / (PPV + TPR)
  **F score** is a harmonic average of precision and sensitivity, it is basically used if we can't decide with of 2 metrics to use for performance measurements.

## IV. Part 1. k-Nearest-Neighbors.

kNN algorithm, as all nonparametric algorithms, is composed of finding the similar instances from the training set using one of distance measures and interpolating from them to find the right output. Distance measures may differ but in our experiment we are going to use Euclidian distance between two multidimensional data points:

$$d(x,y) = \sqrt{\sum_{i}^{n} (x_i - y_i)^2}$$





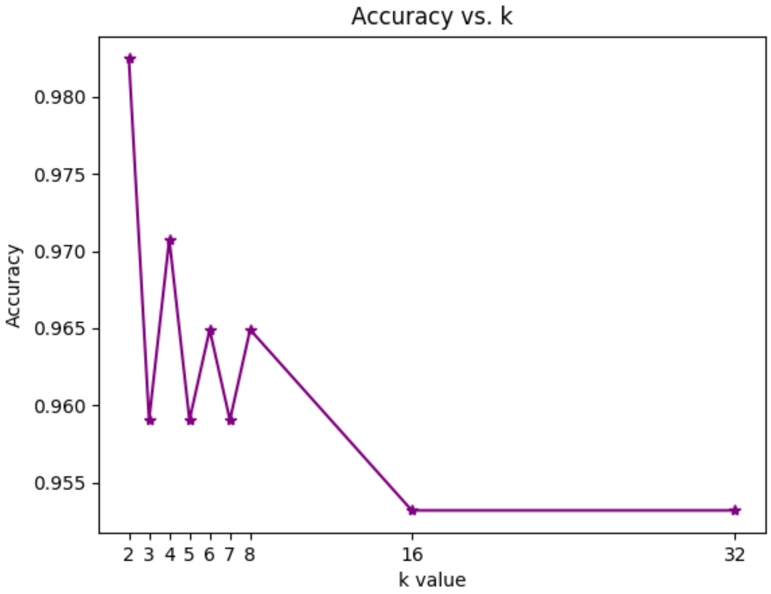
1. *Calculating Neighborhood.*
   The first part of the algorithm after splitting datasets, is to calculate distances for each evaluation(validation or testing) data point and all data points in training set. After that we choose k data points from training set which become k nearest neighbors, obviously. So, our neighborhood for each evaluation data point consists of training data points that are the closest distance-wise. We've used different values of k in this project to compare performance and determine the best ones. The values were following: 2, 3, 4, 5, 6, 7, 8, 16, 32.

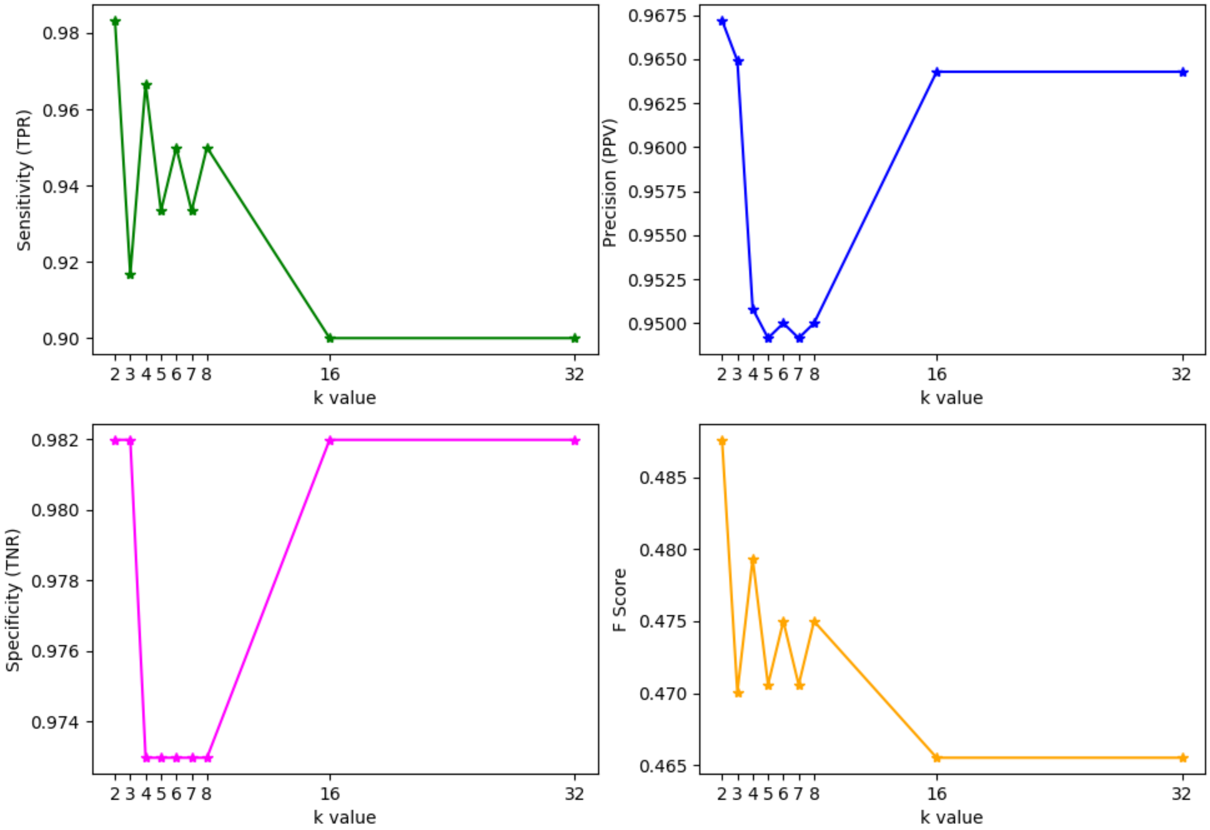
2. *Classification.*
   When neighborhood is known for each evaluation data point, we can classify those data points. What we do is counting votes of each neighbor for class and choose class with the maximum number of votes. For example, if we have three neighbors and two of them are benign and one is malignant, that means that two neighbors vote for class '2', and one votes for class '4'. Since we have more votes for class '2', our evaluation data point will be classified as benign. There are ties possible, in this case I always make evaluation data point to be classified as malignant. Not only it's the one of the most common ties resolutions, in my mind, it makes sense. If we can't be really positive on tumor being good it's better to mark it as bad.

3. *Performance.*
   After we are done with the major part of algorithm, we have to run it for multiple k values to determine the most optimal one. Before we run kNN for all values, we split training set into training and validation parts. We predict classification for validation

set and compare them with real values. For performance metrics, we use confusion matrix that records number of true-positive, true-negative, false-positive and false-negative value. Using metrics from confusion matrix, we calculate prediction accuracy and other relevant measurements of performance.

Results for k = 2, 3, 4, 5, 6, 7, 8, 16, 32 look as following:





## V. Part 2. Decision Trees.

## VI. Conclusions.