**Mini-Manual:**

In my project, I created a machine learning program that predicts whether someone has breast cancer or not. The program trains itself off a part of the dataset I found on the UCI Machine Learning Repository. This dataset of 699 instances, contains 10 attributes, whereas 9 of them affect the label. It then tests itself by predicting whether a patient has cancer or not, and compares its prediction with the actual result, ameliorating itself as it trains and tests more. I have implemented two different classifiers in my project, one being ‘extra\_trees’ and the other being ‘knnclassifier’. This was done in order to analyze the classifiers’ efficiency and provide some more insight into machine learning.

**Design Guide:**

In my program, I imported a few modules such as the random module, the KNN classifier, the extra trees classifier and the more general classifier module. Firstly, I arranged the data in an organized way, splitting up the data and its respective label. Here, I converted the 2’s and 4’s to 0’s and 1’s, respectively, as I found it easier to work with and it basically did not have an effect on the runtime or cleanliness of the code.

My first function, split\_training\_test(dataset, number\_fold) starts by shuffling the dataset, so to reduce likeliness of having lots of data with the same labels in the same training set. This is done in order to allow the computer to learn from a varied set of data. It then assigns a length to the training set by subtracting the number of folds minus one and dividing it by the number of folds. This leads to the shuffled dataset being sliced into two lists: the training\_set (which contains the majority of the folds of data), and the testing\_set (which contains the leftover fold of data). Return: a tuple with the training and testing set.

The second function I created, calculate\_confusion\_matrix(predict\_label, true\_label) starts by initializing a 2x2 zero-matrix. It then increments the corresponding space of the matrix by 1 for each true positive, false negative, false positive and true negative. Return: the confusion matrix.

The third def statement, evaluate\_fitting(confusion\_matrix), assigns the corresponding value of a confusion matrix space, to its variable. For example, the first space in the confusion matrix represents the amount of true negatives, therefore **t**(rue)**n**(egative)= confusion\_matrix[0] [0]. Finally, with these values, we plug them into the true positive rate and false positive rate formulas. Return: a tuple with the true positive rate and false positive rate.

The fourth and final function I created, k\_fold(dataset, number\_fold,M,K,Nmin,P) does all the work. First, assigns the total tpr and fpr the value of 0, to allow us to increment as we go along. Then, if P=0, we notify the user that we are using the extra\_trees algorithm and if P=1 we notify the user that we are using the K-Nearest Neighbour algorithm. For each fold in the specified number of folds, we notify the user of the fold we are on and split the dataset using the split\_training\_test function. Then comes the training section:  
P=0: assign the variable ‘clas’ to the extra\_trees object  
P=1: assign the variable ‘clas’ to the knnclassifier object

We then use the .train method on the training set to train the machine.  
Next is the testing section. We generate a list of the data from the test\_set, called test\_data, and a list of the labels from the test\_set, called true\_label. We then generate another list, which consists of the machine’s label prediction based on the data from test\_data, called predict\_label. Next, we create the confusion matrix with the cacluate\_confusion\_matrix function, by using the real (true\_label) and predicted (predict\_label) label lists, and calculate the tpr and fpr using the evaluate\_fitting function. We then increment the total\_tpr and total\_fpr by the tpr and fpr of that fold.  
We calculate the average tpr and fpr by dividing the totals by the number of folds. Return: a tuple with the average tpr and average fpr.

Finally, we end by assigning variables certain values. Due to the number of instances in the dataset, we use a 5-fold. The M, K and Nmin are values given for the extra\_trees class, which was given by the professor. We start by assigning P the value of 0 in order to begin with extra\_trees. The average tpr and fpr are calculated using the k\_fold function and presented to the user. P is then incremented by 1 in order to recalculate the average fpr and tpr using the KNN class. The k\_fold function is called once more for this reason; the average tpr and fpr are presented and the program ends.

**Results:**

In the end, the average tpr and average fpr on the dataset is given for both the trees algorithm and the KNN algorithm. However, since they consistently vary, the code was run ten times in order to get a more clear average tpr and fpr. The results are in the table below.

|  |  |  |
| --- | --- | --- |
| RUN | **Trees (%)** | **KNN (%)** |
| 1 | 0.953515336121719 | 0.914898799313894 |
| 2 | 0.932288284884407 | 0.940820319323621 |
| 3 | 0.96971961632721 | 0.925503662691168 |
| 4 | 0.963636363636364 | 0.94761874821924 |
| 5 | 0.93882893888493 | 0.924168853480338 |
| 6 | 0.952836484690588 | 0.897001585654154 |
| 7 | 0.940922069425371 | 0.897127882599581 |
| 8 | 0.933438289720388 | 0.939880680954552 |
| 9 | 0.933993905954697 | 0.922280506949497 |
| 10 | 0.917378699006821 | 0.906700853629331 |
| AVERAGE | 0.94365579886525 | 0.921600189281538 |

In summary, the machine had a high accuracy in terms of predicting the label. While both algorithms returned relatively high TPR’s, the Tree algorithm averaged out a higher rate. However, due to the small size of the dataset (699 instances), this project mostly helps us better understand supervised learning in general, as opposed to the advantages of one algorithm over another.

**Sources:**

Dataset: Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository [http:// www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.  
bagging.py, classifier.py, decision\_tree.py, extra\_trees.py, knnclassifier.py: Vincent, Robert. (2019). Assignment 4, Programming Techniques and Applications. [https://www-mpo- ovx.omnivox.ca/cvir/dtrv/ListeTravauxEtu.aspx] Montreal, QC: College Marianopolis.