WDBC Coursework

Davide Mecugni
scydm2@nottingham.edu.cnk
University of Nottingham
Machine Learning (COMP3055 UNNC)

December 22, 2023

Abstract

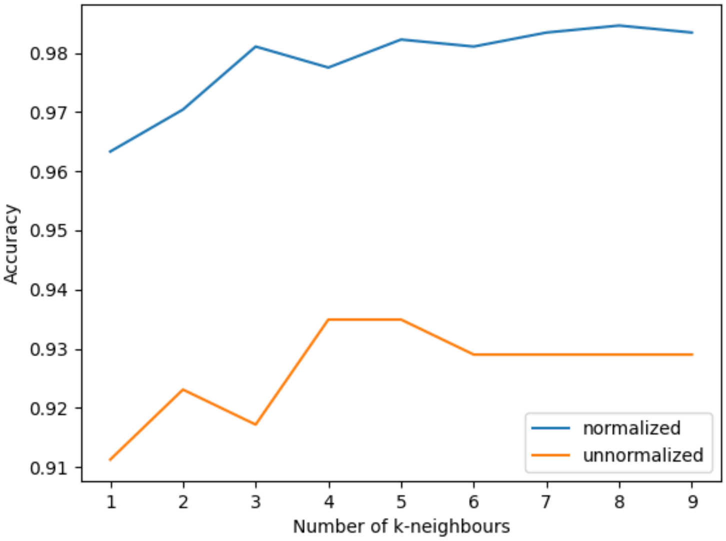
WDBC contains 569 instances of breast cancer data collected by the University of Wisconsin. Each instance is labeled as M (malignant) or B (benign). This Coursework aims to analyze this dataset.
Task 1 retrieves the data from the wdbc.data file and creates various copies of different dimensions using PCA.
Task 2 analyses the reduced datasets using KNN.
Task 3 uses a Deep Neural Network to classify the instances.
Task 4 discusses the advantages and disadvantages of each method.
All the code is written in Python. All the results are obtained using fixed seeds(in Numpy, Pytorch, and random) so that the exact results can be replicated every run.
The code is run on an HP Pavillion laptop using AMD Ryzen 7 5700U at 1.80 GHz.

Task 1

Task 1 was completed using the CSV python module for retrieving the dataset from the wdbc.data file. Once the dataset is retrieved it can be further split into training/test using the random library for pseudo-randomic splitting. The pseudo-random nature ensures that the results are replicable across different runs. The feature reduction was achieved using the PCA module from sklearn.

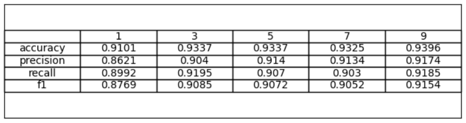
Task 2

Once I retrieved the dataset in task 1, I applied KNN to the different reduced normalized datasets and the full dataset. I used a normalized dataset because the results proved to be much better than the unnormalized one.
Here is the accuracy of KNN on the 30 features dataset normalized and unnormalized:



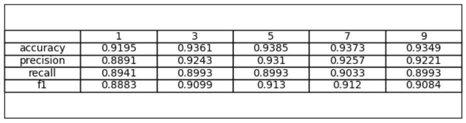
Normalization is required for KNN otherwise the weight of bigger attributes skews the final result. Normalization makes every attribute weight the same, which is a correct hypothesis if further information about the attributes is not available.
I generated tables and graphs to help me understand the best number of features to use for further analysis.
To get more stable results, I applied a 5-fold cross-validation, averaging the final results. Without using cross-validation, I obtained precisions and other values at 100%, after using cross-validation I obtained more reasonable results. The different folds are obtained by modifying the random seed.
The tables and graphs show the various averaged metrics between the 5 folds for each reduced dimension dataset at different k values:

Statistics for 5 dimentions



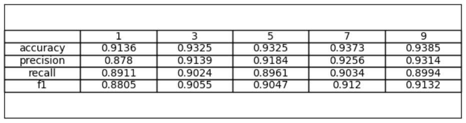
	1	3	5	7	9
accuracy	0.9101	0.9337	0.9337	0.9325	0.9396
precision	0.8621	0.904	0.914	0.9134	0.9174
recall	0.8992	0.9195	0.907	0.903	0.9185
f1	0.8769	0.9085	0.9072	0.9052	0.9154

Statistics for 10 dimentions



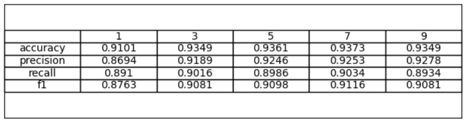
	1	3	5	7	9
accuracy	0.9195	0.9361	0.9385	0.9373	0.9349
precision	0.8891	0.9243	0.931	0.9257	0.9221
recall	0.8941	0.8993	0.8993	0.9033	0.8993
f1	0.8883	0.9099	0.913	0.912	0.9084

Statistics for 15 dimentions



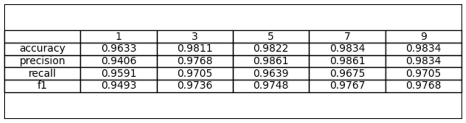
	1	3	5	7	9
accuracy	0.9136	0.9325	0.9325	0.9373	0.9385
precision	0.878	0.9139	0.9184	0.9256	0.9314
recall	0.8911	0.9024	0.8961	0.9034	0.8994
f1	0.8805	0.9055	0.9047	0.912	0.9132

Statistics for 20 dimentions

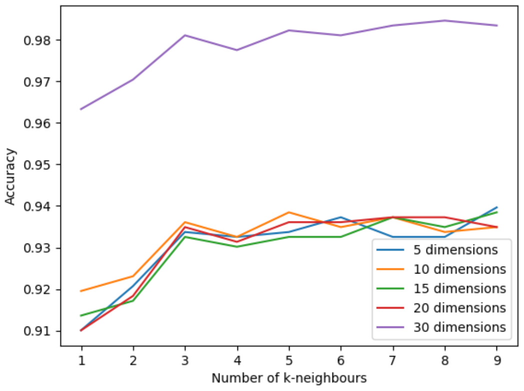


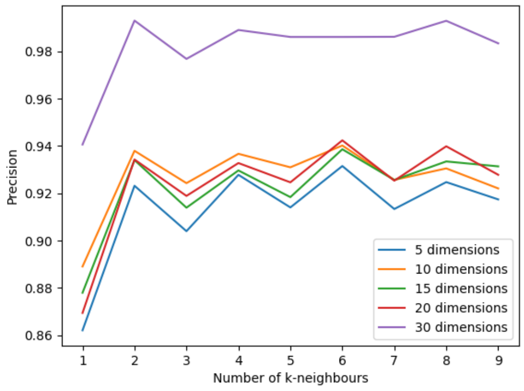
	1	3	5	7	9
accuracy	0.9101	0.9349	0.9361	0.9373	0.9349
precision	0.8694	0.9189	0.9246	0.9253	0.9278
recall	0.891	0.9016	0.8986	0.9034	0.8934
f1	0.8763	0.9081	0.9098	0.9116	0.9081

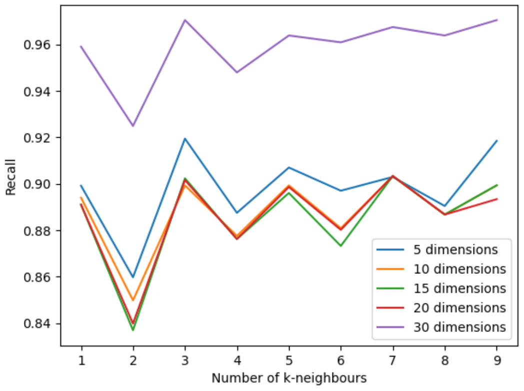
Statistics for 30 dimentions

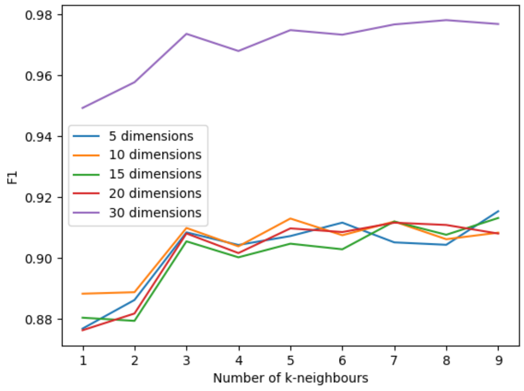


	1	3	5	7	9
accuracy	0.9633	0.9811	0.9822	0.9834	0.9834
precision	0.9406	0.9768	0.9861	0.9861	0.9834
recall	0.9591	0.9705	0.9639	0.9675	0.9705
f1	0.9493	0.9736	0.9748	0.9767	0.9768









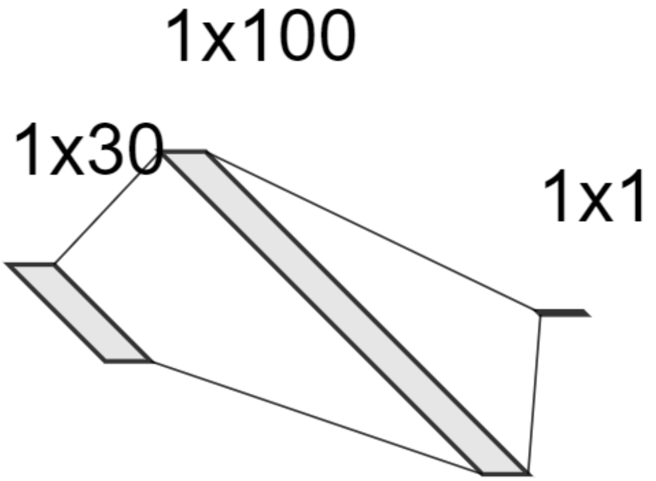
As can be easily seen in the graphs, the best metrics are achieved using 30 features.
In particular, the best performance on test data is achieved using KNeighbors with K=9 and 30 features, scoring:

Accuracy	98.34%
Precision	98.34%
Recall	97.05%
F1	97.68%

The computing time on my machine for compression and KNN on all 5 datasets is 0.59s.

Task 3

The MLP binary classifier is coded in Python using the Pytorch framework.
I tried different implementations, and the one that worked the best is the following:

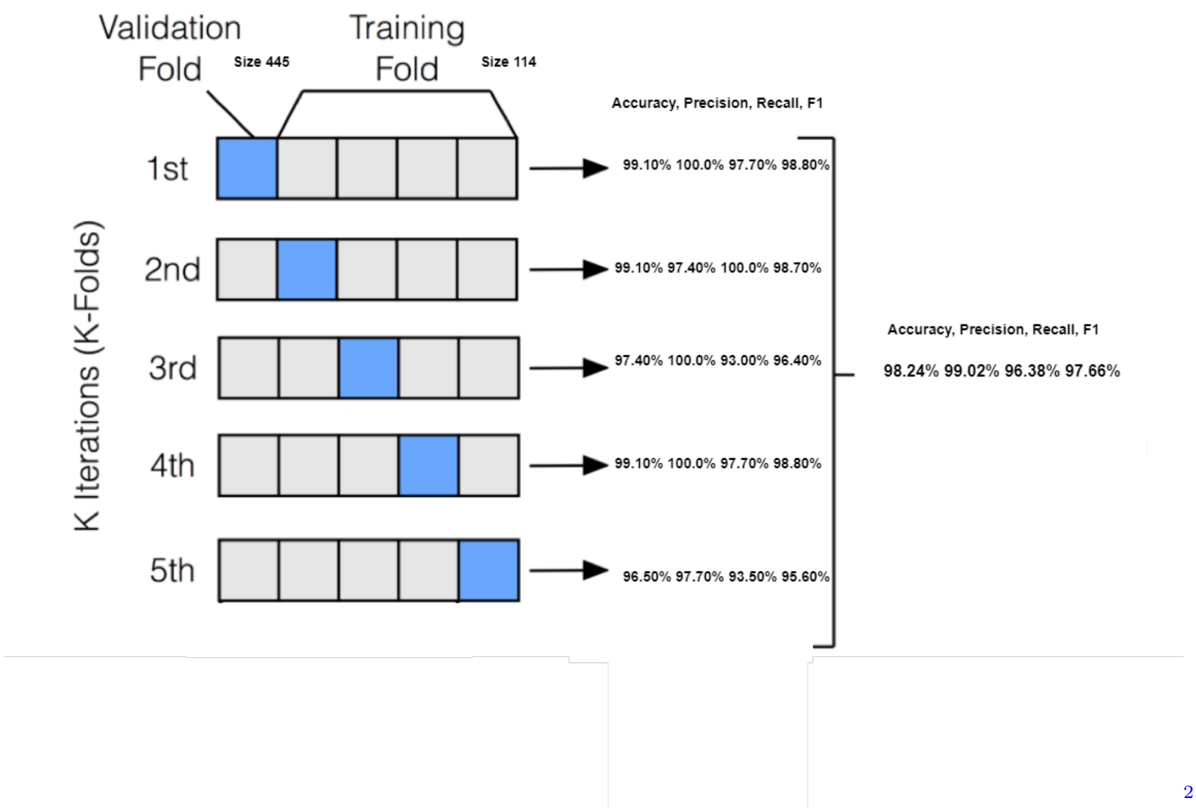


The design is a fully connected neural network of size 30x100x1. The hidden network uses batch normalization and dropout at 0.5. The final layer uses dropout at 0.5 because it proved to be better performing in testing.
The model uses Stochastic Gradient Descent, Binary Cross Entropy loss function, and sigmoid activation function[1]. Since the use of scheduler has improved the metrics only on training data and not on test data, I have decided not to use one.
It is further observed that the best performance is achieved with a batch size of 100 and training for 100 epochs.
The learning rate is 0.5.
Further explanation of the choices made is provided later.
The model is trained and tested using 5-fold cross-validation. The final averaged results on normalized test data between the 5 folds using this architecture are:

Accuracy	98.24%
Precision	99.02%
Recall	96.38%
F1	97.66%

[1] The choice is influenced by an article by Nghi Huynh at link

Here are the different metrics for the 5 folds and the averaged results:



[2]

Different choices have been tested to reach this final network design.
Following are some variations tried using 100 epochs size if not stated differently. All the subsequent choices follow the best hyperparameters found before. The metrics shown are accuracy, precision, recall, and f1.

Different designs

Here are the different designs tested during development and their metrics:

30x30x1	97.86%	98.54%	95.88%	97.16%
30x60x1	97.68%	98.08%	95.88%	96.94%
30x100x1	**98.24%**	**99.02%**	**96.38%**	**97.66%**
30x30x30x1	97.88%	98.56%	95.88%	97.16%
30x60x60x1	97.70%	98.08%	95.90%	96.96%
30x100x100x1	97.52%	97.62%	95.88%	96.72%
30x30x30x30x1	98.06%	98.08%	96.80%	97.42%
30x60x60x60x1	97.89%	98.08%	96.34%	97.18%
30x100x100x100x1	97.54%	97.19%	96.34%	96.75%

A network design that is too small will not fully encode all the patterns inside the data. On the other hand, a too-big network requires longer epochs to train and is more prone to overfitting. A too-deep network might suffer from gradient stagnation, where the backpropagation step doesn't affect the initial neurons as much as it affects the ones closest to the output.
The design chosen uses one hidden layer because it proved to be the best design in testing.

[2] Image for Kfold cross-validation at link

Different activation function

Here I tested different activation functions for different connections. The last activation function uses Sigmoid in every test, following the common architecture for binary classifiers.

ReLu	96.13%	95.30%	94.51%	94.82%
LeakyReLU	95.96%	94.51%	94.98%	94.60%
Tanh	96.84%	95.35%	96.37%	95.82%
Sigmoid	**98.24%**	**99.02%**	**96.38%**	**97.66%**

Other activation functions proved to be less performing than the sigmoid activation function.

Different batch sizes

Here are the different batch sizes tested during development and their accuracy, the learning rate is proportional to the batch size to get consistent results:

25	98.04%	99.48%	95.42%	97.38%
50	97.00%	97.64%	94.42%	95.96%
100	**98.24%**	**99.02%**	**96.38%**	**97.66%**
200	98.04%	98.54%	96.34%	97.4%
400	97.88%	98.56%	95.82%	97.12%

A bigger batch size moves the gradient more stably, a smaller batch size frequently changes the gradient and could find local minima that are not global.

Different epochs

10	97.36%	99.00%	94.08%	96.46%
50	97.70%	99.46%	94.48%	96.86%
100	**98.24%**	**99.02%**	**96.38%**	**97.66%**
200	98.06%	98.62%	96.34%	97.44%
1000	97.52%	97.58%	95.94%	96.7%

A short epoch length might mean the network has not been exposed to enough data to make correct predictions. If the epoch length is too long it could cause overfitting on the training data. This could cause a lower performance on the test data.

In my implementation training for more than 100 epochs proved to increase overfitting. This can be explained given that the network over-learns patterns in the training data that are not present in the general dataset.

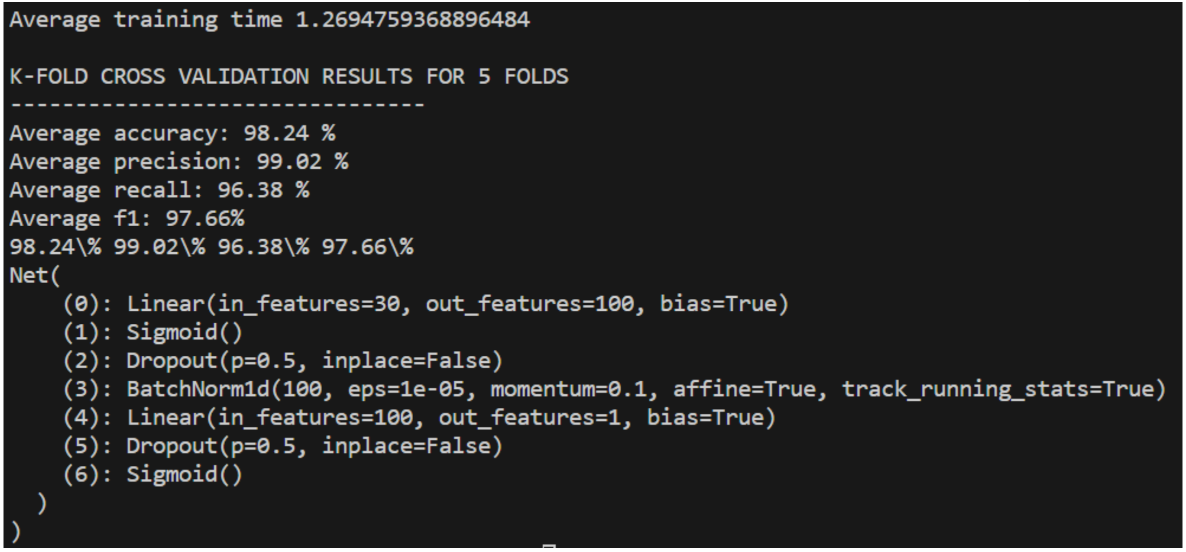
Different Learning rates

0.01	94.37%	96.76%	87.74%	92.01%
0.10	97.54%	99.02%	94.51%	96.67%
0.20	97.54%	98.07%	95.44%	96.72%
0.50	**98.24%**	**99.02%**	**96.38%**	**97.66%**
0.75	97.19%	97.15%	95.50%	96.23%

A learning rate that is too low might make the training too slow. A learning rate that is too high might not lower the loss function in later epochs, circling the local minima without reaching it.

Final observations

The average time for training a single MLP on my machine is 1.27s.
All the trained models are available in TorchScript in the .pth files for easier replicability.



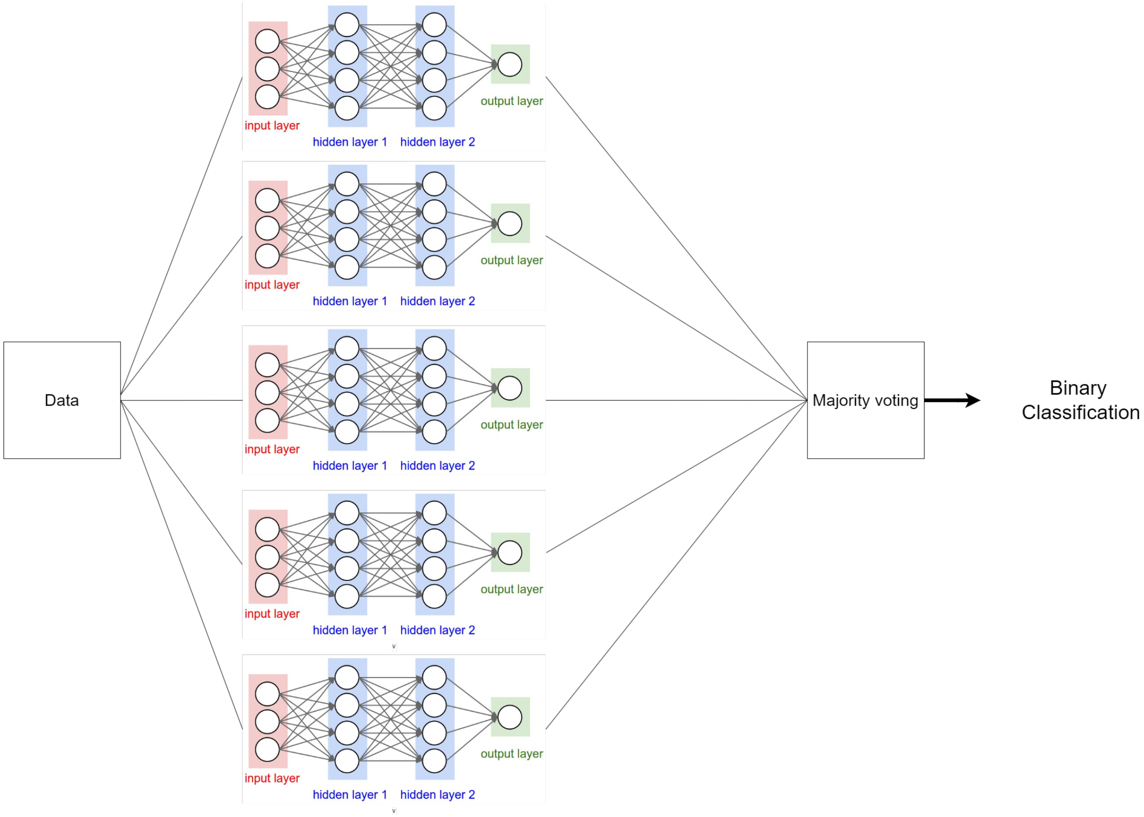
```
Average training time 1.2694759368896484

K-FOLD CROSS VALIDATION RESULTS FOR 5 FOLDS
--------------------------------
Average accuracy: 98.24 %
Average precision: 99.02 %
Average recall: 96.38 %
Average f1: 97.66%
98.24\% 99.02\% 96.38\% 97.66\%
Net(
    (0): Linear(in_features=30, out_features=100, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.5, inplace=False)
    (3): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): Linear(in_features=100, out_features=1, bias=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Sigmoid()
  )
)
```

This is the final screenshot of the metrics on 5-fold cross-validation on test data. The network design is shown using PyTorch framework.

Task 3 Optional Ensemble Network

I tried to implement an optional ensemble of 5 MLPs trained in the previous task. The final binary classification is done using a voting system where the label with the majority of labels is chosen.
Here is the design:



[3]

The final results of the ensemble network on the entire dataset are:

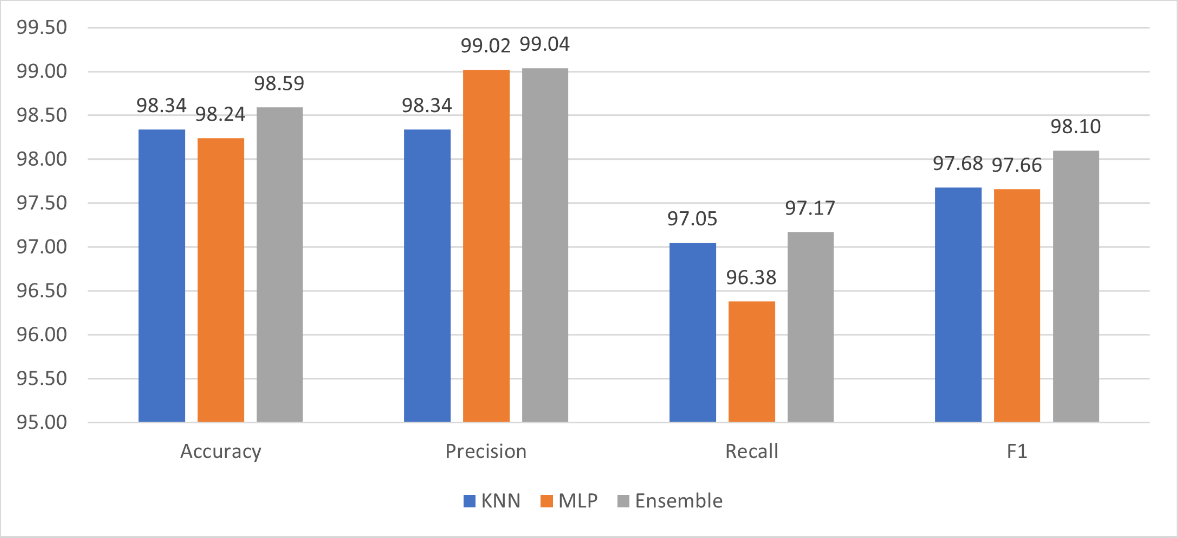
Accuracy	98.59%
Precision	99.04%
Recall	97.17%
F1	98.10%

It must be noted that by calculating the metrics on the entire dataset the final result is somewhat skewed. This is because the testing data has been seen by the different MLPs in training.
Total time for classification on the entire dataset: 8.61s.

[3] The image for a single NN can be found at link

Task 4

Figure 1: 5-fold cross-validation metrics on test data



It must be noted that the computational time is greater for MLP compared to KNN. This is justified by the fact that KNN just stores the dataset for training while MLP must do the forward and backpropagation steps. Deeper NN designs tested required far greater training times.
Given that KNN is an instance-based ML algorithm, it finds the closest recordings in the training data to the data seen in testing. While this method is computationally efficient, it doesn't generate any underlying model of the data. MLP can identify complex relationships within data, creating a model of the underlying phenomenon.
KNN could suffer from overfitting if the k is too low(ex. one). The most performing k was 9 showing that a larger k solves the problem of overfitting.
Given that the model is based on the data seen in training, MLP can suffer from overfitting. Using regularization of input data, batch normalization, and dropout my implementation of MLP managed to have low overfitting. Overfitting has also been avoided using the epoch length that gave the best results in the development phase.
The most important metric in a binary classifier of this type is recall, given the cost of a false negative(the patient is told they don't have breast cancer when in reality they have it).
The KNN classifier proved to have a better recall metric, suggesting that it is better performing than the MLP for the task of cancer labeling. The precision of the MLP classifier proved to be better than the KNN precision metric. This suggests that MLP is better for correct positive labels(if the patient is told they have cancer, it is more probable that cancer is present for MLP than KNN).
The F1 is fairly close for KNN and MLP but, as already stated, recall could be a better indicator for this type of binary classifier.
Optional
The best metrics achieved in this report are using the optional ensemble network design I proposed. In particular, the recall(97.17%) is better than KNN and the single MLP design. Using an ensemble network design gives better metrics than a single MLP because the different starting weights and different training samples make the ensemble network less prone to overfitting.