Classification of Breast Cancer cells with Gaussian Naive Bayes and Neural Networks

Matthijs Muis
Janna van Assen

ABSTRACT

In this report the Breast Cancer Wisconsin (Diagnostic) Data Set [3] is analyzed with two different methods. The first approach uses the Gaussian Naive Bayes classifier (GNBC) with feature selection using a hierarchical clustering approach proped by Blanquero et al. [1]. There are computational problems with this approach, but we content ourselves with an approximately optimal classifier on 5 features. The second method makes use of neural networks, the sklearn Multi-layer Perceptron classifier in specific, to classify the cancer cells in the data set. For this method we will take a practical approach and look at the optimal parameters for this classification algorithm.

ACM Reference Format:
Matthijs Muis and Janna van Assen. 2024. Classification of Breast Cancer cells with Gaussian Naive Bayes and Neural Networks. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 DATA SET

1.1 Overview

The data set being studied is the Breast Cancer Wisconsin (Diagnostic) Data Set. It contains one target feature, ‘diagnosis‘, which has values the values M (Malignant cell) and B (Benign cell), and 30 continuous features. These features were computed from "digitized images of a fine needle aspirate (FNA) of a breast mass" and "describe characteristics of the cell nuclei present in the image." [3]. These 30 features represent 10 characteristics of the cell nuclei:

(1) radius
(2) texture
(3) perimeter
(4) area
(5) smoothness
(6) compactness
(7) concavity
(8) concave points
(9) symmetry
(10) fractal dimension

The mean value, standard error and the worst/largest value of these 10 nuclei characteristics were computed for each image, resulting in 30 features.

Conference'17, July 2017, Washington, DC, USA
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.2 Historical note

This data set was created and analysed in 1995, using a method that is still widely used today: the decision tree classifier. The tree classification was formulated as a linear hyperplane separation problem, and was solved by formulating and directly solving a linear program, rather than using a heuristic algorithm like ID3 or CART. The original paper describing this method is *Breast cancer diagnosis and prognosis via linear programming*. [2].

The analysts found that the best predictive accuracy was obtained using one separating plane in the 3-D space of Worst Area, Worst Smoothness and Mean Texture. The estimated accuracy of the classifier was 97.5%, using repeated 10-fold cross validations. As of November 1995, the classifier has correctly diagnosed 176 consecutive new patients.

1.3 Practical summary

The data set has 30 input features and 1 target feature. All input features have continuous, numerical values. The data set contains no missing values. There is a total of 569 described cells in the data set, of which 357 benign and 212 malignant. This indicates we will not have to compensate for a class imbalance.

2 EXPLORATORY DATA ANALYSIS

A Principal Component Analysis was performed on the data, with as many components as the rank of the data matrix (i.e. a full SVD). Figure 1 shows that 98.2% of the variance is explained by the first principal component, where the second component only explains 1.6%, and the third explains 0.16%.

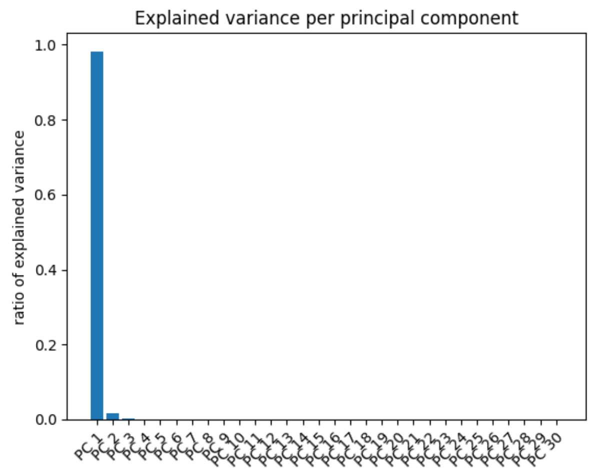

Figure 1: Ratios of explained variance per principal component.

Scatter plots of the data projected onto PC1 coupled with PC2,PC3 or PC4 (Figure 2) show that the benign cells form a much less heterogeneous group than the malignant cells. Furthermore, it becomes apparent that the classes are not linearly separable in these two-dimensional projection spaces. It however is noticeable that there is a clear difference between the mean centroids and within-cluster variance in these projection spaces.

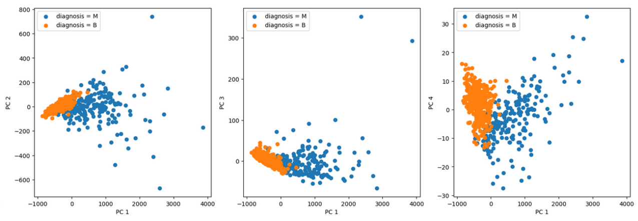

Figure 2: Scatter plots of the data projected onto the spaces spanned by $\{PC1, PC2\}$, $\{PC1, PC3\}$, $\{PC1, PC4\}$ respectively.

A projection of the data onto span($\{PC1, PC2, PC3\}$) is plotted in Figure 3. We notice that the sets may already (almost) be linearly separable in this space. This is a good foundation for the original author's decision of applying a tree classifier.

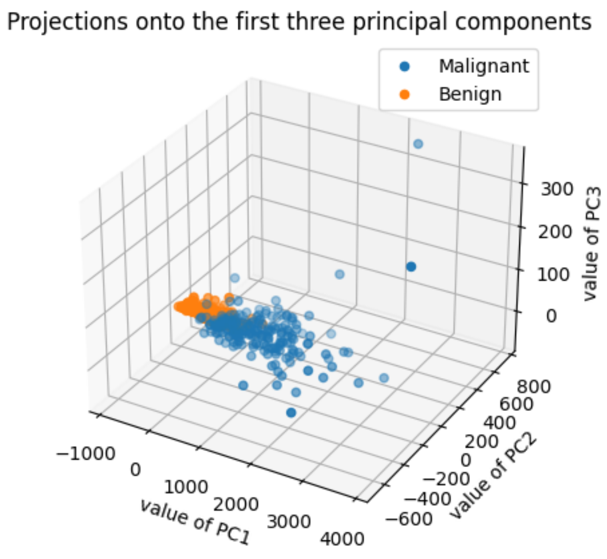

Figure 3: Scatter plots of the data projected onto span($\{PC1, PC2, PC3\}$)

3 GAUSSIAN NAIVE BAYES CLASSIFIER

The naive Bayes classifier is a classification model for class data $\mathcal{D} = \{(c_i, x_i)\}_{i=1}^{N}$, where we assume (c_i, x_i) are i.i.d. realizations of a marginal of i.i.d. stochastic variables $(C_i, X_i) \overset{i.i.d}{=} (C, X)$. We write $\mathcal{C}$ for the outcome space of $C : \Omega \to \mathcal{C}$ and $\mathcal{X}$ for the outcome space of $X : \Omega \to \mathcal{X}$. $\mathcal{X} \subset \mathbb{R}^p$, and we denote $x_i = (x_i^1, x_i^2, ..., x_i^p)$. Further, $\mathcal{C}$ is finite, and for a binary classification problem, $\mathcal{C} = \{0, 1\}$. In general, the joint distribution of (C, X) may be very complex. The classification problem consists of understanding how C depends on X

The naive Bayes classifier (NBC) is based on the following assumption on the joint distribution of (C, X):

$$\mathbb{P}(X^j \in B_j | \cap_{i \neq j} \{X^i \in B_i\}, C = c) = \mathbb{P}(X^j \in B_j | C = c)$$

In words, given the event $\{C = c\}$, the variables X_i are independent. Another equivalent formulation is:

$$\mathbb{P}(\cap_{i=1}^{p} \{X^i \in B_i\} | C = c) = \prod_{I=1}^{p} \mathbb{P}(X^i \in B_i | C = c)$$

This assumption makes it possible to derive a probability distribution for C conditional on X by using Bayes' theorem: if $B = B_1 \times B_2 \times ... \times B_p$, then:

$$\mathbb{P}(C = c | X \in B) = \frac{\prod_{i=1}^{p} \mathbb{P}(X^i \in B_i | C = c)\mathbb{P}(C = c)}{\sum_{c' \in \mathcal{C}} \prod_{i=1}^{p} \mathbb{P}(X^i \in B_i | C = c')\mathbb{P}(C = c')}$$

In introductory treatments of the NBC, X^i is often a discrete random variable with a finite value space $\mathcal{X}_i$. This assumption is not at all necessary, as the above formula applies to any random variable as long as for each i, B_i has a well-defined probability measure $\mathbb{P}_{X^i}(B_i)$. In reality, one often assumes that X^i is either discrete or real and absolutely continuous. The first means that any subset of $2^{\mathcal{X}}$ can be measured, and that the measure $\mathbb{P}_{\mathcal{X}}$ is fixed by choice of $\mathbb{P}_{\mathcal{X}}(\{x\})$ on singletons in $2^{\mathcal{X}}$, i.e. by fixing a probability mass function.

Absolutely continuously distributed real random variables, on the other hand, have their probability distribution given by the integral:

$$\mathbb{P}(X^i \in B | C = c) = \int_B p_c^i(x)dx$$

Where we only need to fix, for every $i \in \{1, .., p\}$, $c \in \mathcal{C}$, the density $p_c^i : \mathbb{R} \to \mathbb{R}$. That would completely specify all the probabilities in the model.

The Gaussian NBC (GNBC) assumes $X^i | \{C = c\}$ follows a Gaussian probability, in other words $p_{ic} = \phi_{(\mu_{ic}, \sigma_{ic}^2)}$. This gives $4p$ model parameters $\{\mu_{ic}, \sigma_{ic}^2\}_{i=1,..,p, c \in \mathcal{C}}$, which are estimated from the data and then plugged into the prediction formula. The prediction formula receives a reformulation in terms of densities rather than probabilities:

$$\mathbb{P}(C = c | X = x) = \frac{\prod_{i=1}^{p} p_c^i(X^i)\mathbb{P}(C = c)}{\sum_{c' \in \mathcal{C}} \prod_{i=1}^{p} p_{c'}^i(X^i)\mathbb{P}(C = c')}$$

3.1 Advantages of GNBC

As opposed to a decision tree classifier (i.e. the separating hyperplane used the original method), a prediction by GNBC is a probability that is derived from theoretically clear assumptions about the conditional distribution of the data. The probability can quantify how (un)certain the model is of its prediction. This uncertainty cannot be recovered from a decision tree prediction.

3.2 Limitations of GNBC

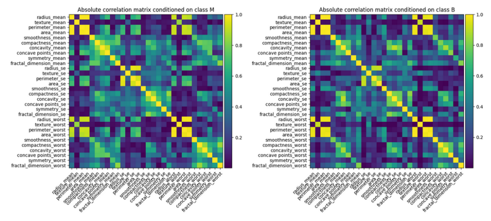

Figure 4: Heatmap plot of the absolute values of correlation matrices of the features, conditioned to the class variable.

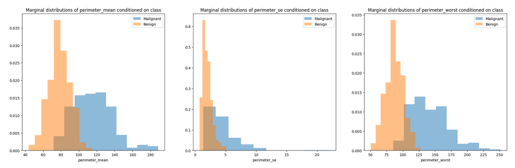

Figure 5: Plots of the marginal distribution of feature perimeter, conditioned on class.

There are two clear limitations of applying the GNBC to this particular data set:

(1) The features in the data set seem to be correlated, and this also holds true for their distributions conditioned on the class variable 'diagnose'. This can be shown in Figure 4, which is a simple plot of the two conditioned covariance matrices. Especially the features for 'radius', 'perimeter' and 'area' are strongly positively correlated.
(2) The standard-error features 11-20 are computed standard deviations of radius elements, perimeter elements etc. per image. Squared deviations $(Z - \bar{Z})^2$ are statistically known not to be normally distributed, but rather chi-squared distributed if their underlying random variable Z is normal. This is indeed observed in the actual data: we found this by plotting the marginal distributions of each feature, conditioned to the class M or B. See Figure 5 for one of these plots, for the feature `'perimeter'`: especially the _se-variants of the feature shows the characteristic skewness of a chi-squared distribution.

The first reason implies that the NBC assumption of independence of the features X conditioned on C is violated if we simply include all features in the model.

For these reasons, care must be taken in selecting appropriate variables. This shall be discussed further under *Feature selection*.

3.3 Feature selection

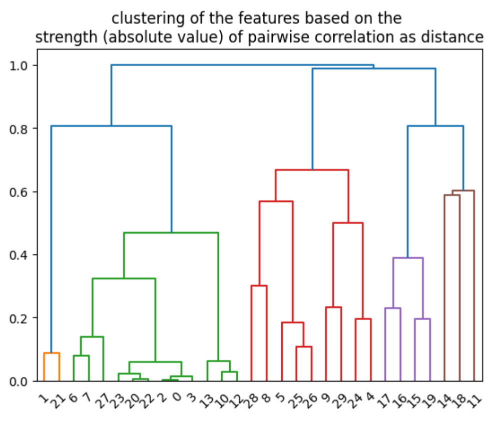

Figure 6: The dendrogram corresponding to the clustering of the features, where as distance is taken one minus the absolute value of their pairwise correlations.

The principal concern in feature selection was assuring conditional independence of the selected features. We adopted the approach proposed in [1], which makes clever use of hierarchical clustering to select subsets of features with maximum correlation between features:

(1) Calculate the 30×30 correlation matrix Cor of the features (the original paper uses the mutual information H since it applies to nominal class data).
(2) Define a distance D on the features $D_{ij} = 1 - |Cor_{ij}|$ (The original paper uses $D_{ij} = \frac{H^* - H_{ij}}{H^*}$ where H^* is the maximum entry of the matrix H. However, for our purposes, a large negative correlation should also be avoided: we, in general, need two features i, j to be more similar when Cor_{ij} is close to 1 in absolute value).
(3) We now want clusters of features $\mathcal{I}$ to have their smallest absolute correlation $\min_{i \neq j \in \mathcal{I}} |Cor_{ij}|$ maximized. This means that $\max_{i \neq j \in \mathcal{I}} D_{ij}$ should be as small as possible, so a complete linkage clustering is performed.
(4) We divide the data in n flat clusters, where n is the number of features that we want to include in the model. From each flat cluster, a feature is selected.

The result of the clustering in step 2 is displayed in Figure 6.

This approach aims to minimize the correlations among n features. The randomized approach makes it somewhat arbitrary, but it is in general too computationally expensive to try every combination of n features, even when we restrict our choice to one feature per cluster as in the above algorithm:

- If all nonempty subsets of features are considered, this yields $2^{30} - 1$ possible feature sets.
- When only considering feature sets where each feature is picked from a separate cluster, i.e. the approach of [1], this gave 52683 different feature sets to consider if n was allowed to be anywhere in the range $[1, 28] \cap \mathbb{N}$

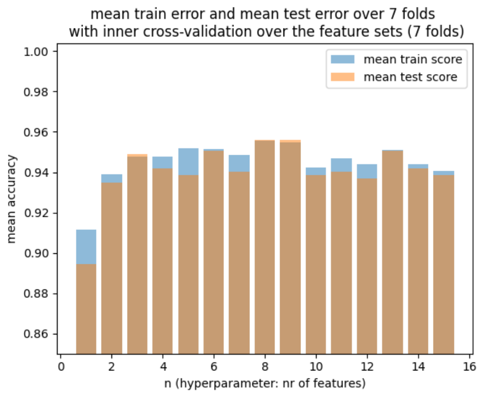

Figure 7: The mean accuracy (calculated over k=7 outer folds, with k=7 inner folds for selection of the features from an *n*-feature-subset.

Although the heuristic gives a great reduction in the possible feature sets, it is still too large. This is why we randomly select 10 feature sets from the flat clustering into n clusters, for $n = 1, \ldots 15$, and we compare these using a nested cross-validation, where:

(1) With the inner fold, we select for each choice of n the best feature set of the 10 candidate feature sets for that n.
(2) In the outer fold, we compute the accuracy on the test set.
(3) We take the mean accuracy over each fold of the outer partition, thereby giving a mean train accuracy and mean test accuracy for each n.
(4) We plot n against the mean train accuracy and mean test accuracy.

3.4 Open Problems

We discovered that there is a problem with our execution of the approach of [1], lying in the fact that we reduce the number of feature sets considered even further by just randomly filtering to 10 candidate sets per value of n.

The hierarchical clustering of correlated Blanquero assures that if a set of features is considered, all its subsets are as well. I.e. if we have a feature set $\{f_1, f_2, f_3, f_4, \ldots, f_n\}$, where f_i comes from the i-th cluster, there are also flat clusterings (partitions) into $n-1, n-2, \ldots 1$ feature sets, say $\pi^k = \{B_1, \ldots, B_k\}$, $k = 1, \ldots, n-1$, and these clusterings π^k will have any set $S \subset \{f_1, \ldots, f_n\}$ in the product of their partition blocks $\prod_{i=1}^{k} B_i$, so every subset S of $\{f_1, \ldots, f_n\}$ in the end. In other words, for every n-feature set that the algorithm considers, all $n-1, \ldots 1$-feature sets that are a subset of this feature set are also considered. This ensures that the training error is a monotonous function of the number of features n, i.e. increasing n will always increase the degrees of freedom of the fit.

However, considering every subset of features of size n took too long. So we selected, as described, for every n, randomly 10 candidate feature sets. This also meant that the feature sets in `candidate_feature_sets[n]` do not have all their subsets also represented in the to be considered feature sets. This was a mistake: it means that the train score will not increase monotonously with the number of included features, because the feature sets with rising n are not included within each other: informally, for a good fit on n features S, the fit on $n+1$ feature $S \cup \{f\}$ will always have a better train accuracy, but in our random sampling approach, $S \cup \{f\}$ may never be considered, and only worse fits on $n+1$ features are considered, hence the hyperparameter - train - test -accuracy curve (Figure 7) is not monotonous for the train accuracy. Therefore, it does not show much useful information. We even decided not to show a curve, but a barplot for the reason that there is no clear functional dependence: There is no "graph" if we expect no clear relationship between n and the train/test errors.

What we could have done to otherwise, isr:

(1) Increase computational resources, such that it becomes feasible to iterate through all 49880 feature sets rather than just a pick of 150 sets. Considering all feature sets ensures that every n-cardinality feature set is contained in a $n+1$-cardinality feature set, so that the train error will be a monotonous function of n. This would make overfitting visible and allows us to make a bias-variance tradeoff decision.
(2) Use a different kind of feature selection algorithm such as RFE (recursive feature elimination), where we recursively eliminate features from the set and thus assure that for every pair of feature sets that is considered one set is contained in the other.

We decided not to do any of this, because the first option is impossible anyway, and the second option required us to drop the in its own right very interesting proposed algorithm of [1], which we decided against since:

(1) it took a lot of time to implement and explain.
(2) RFE will very likely violate the Naive Bayes assumption, so is not a satisfying solution in the end. It is nice if one wants to show some well-known, traditional plots of overfitting, but it does not make sense from the perspective of the model assumptions, which requires the selected features to be independent conditional to class C.

3.5 Optimal GNBC

For now, we will content ourselves with the choice of n that has the best mean **test** score: this is a classifier with n=6, see Figure 7. It has an estimated accuracy of 96%. Note that the feature set can be freely chosen from `feature_sets[5]`, and this choice is considered part of the model fit, not of the hyperparameter fit. But for the interested reader, we will also mention the feature set that, in the average case, gave the best test error:

```
'texture_worst', 'radius_worst', 'concavity_worst',
'fractal_dimension_se', 'smoothness_se'
```

To show that this feature set indeed has low correlation between features, we also include a plot of the covariance matrix. This is figure 8.

4 MULTI-LAYER PERCEPTRON CLASSIFIER

Neural networks are often used for classification tasks. In this study the sklearn multilayer perceptron classifier was used. We assume readers are already well familiar with the working of (multilayer) perceptrons and will therefore gloss over the theory and focus on how the parameters of a neural network influence its performance,

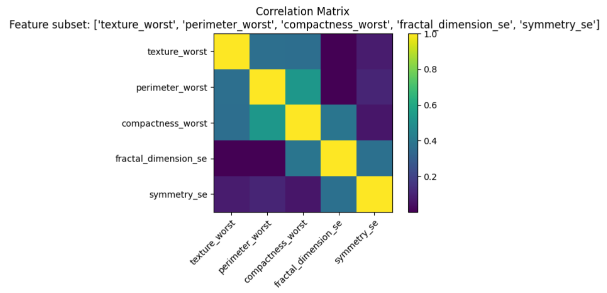

Figure 8: Absolute value of correlation matrix (as a heatmap) of the final, best generalizing of the considered feature sets.

and take a more practical approach. We will look in depth at how an optimal neural network classifier can be found, and look at the results that can be achieved for this data set.

4.1 Method

Our goal is to find the best multi-layer perceptron classifier for this data set, so doctors can determine even better whether a clump of breast cells is malignant or benign tumor. For this, we will have to wade to seemingly infinite possible ways to set the parameters of our classifiers. To start determining what classifier is best, it first needs to be determined what we define to be a good classifier.

In this case, the best measure of performance appears to be the accuracy in 10-fold cross validations. We choose this measure because this allows us to compare our results to those achieved with decision trees in the 1995 study, where the same measure was used. Here an accuracy of 97.5% was reached. Our aim is to reach or even beat this percentage.

Because it is unrealistic to experiment with all possible parameters within the scope of this paper, there will be a few characteristics of our neural network that will be kept constant. These include the activation function, for which we will use a rectified linear unit function ('relu'), which is a popular choice in scientific literature. We will also consistently use a stochastic gradient-based optimizer solver ('adam') to find the minimum of our cost function. The solver will have the parameters alpha=0.9, beta1=0.99 and beta2=0.99995. These values are chosen seemingly arbitrary, based on current scientific knowledge and experience. We used a maximal amount of iterations of 300.

The parameters of the neural network that will be varied are the learning rate and the sizes of the hidden layers containing neurons. The learning rate determines how rapidly the weights can change between iterations. We do not want the learning rate to be too small for our application, as this will cause the neural network to have a slow convergence. However, a too big learning rate will cause our weights to oscillate, and never properly converge to the optimal weights. We must find the optimal middle ground for this data set. The amount of neurons influences the complexity that our model is able to express. More neurons, in multiple hidden layers, will therefore make our classifier more expressive, and able to better capture patterns within our data. However, with more neurons in our network, we also have more weight paramters that need to be fit. This can cause overfitting of the model, and therefore decrease accuracy. This is why it is also import to find a balance in the amount of neurons and their distribution over the layers of the network. But first of all, we investigate whether it is better to fit the neural network on the normal data, or to fit the model on the data that was preprocessed with PCA. An advantage of using the model on preprocessed data is that important connections between features would have already been made, causing the process to be more efficient and effective. An advantage of using the non-preprocessed data would however be that this way, individual features could be better expressed within the neural network. Our hypothesis, based on experience, is however that using data that has been preprocessed with PCA will be a better choice. For the use of PCA, we will look at the optimal amount of principal components to use as input for the network. Using more parameters will mean a larger dimensional input for the neural network, and therefore also more weight parameters to fit for the network. Next to that, as we saw in the data exploration, most higher number principal components have little influence on outcome anyways. An advantage of using more or all principal components is however that this means that our classifier will have more information to base its decision on.

4.2 Results

To determine whether it is best to use pre-processed data, we tested classifiers identical in all but the input data (non-preprocessed, or preprocessed, where the first 1 to 30 principal components were used). For these classifiers a default learning rate of 0.001 and a hidden layer size of (30,) were used. The results are visible in Figure 9. It becomes apparent that it is best to use PCA in the preprocessing stage, and that using the first 5 or more principle components gives us the best results. From now on, we will optimise the rest of the parameters using the first 5 principle components, as this gives the highest accuracy in this test, and it is the simplest model that preforms well, using the principle of Occam's razor.

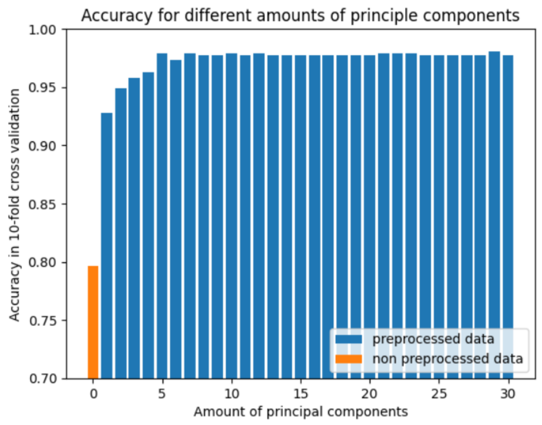

Figure 9: Accuracy in 10-fold cross validation for non-preprocessed data and for preprocessed data, where different amounts of principle components were used.

Secondly, we varied the learning rate. We did this while keeping the amount of principle components used constant at 5, and while keeping the hidden layer size shape constant at (30,). Analysing

Figure 10, we conclude that learning rates between 0.001 and 1 give comparably good results, and other learning rates are either too big or too small. We will further use a learning rate of 0.001, as the optimal accuracy was found with this value.

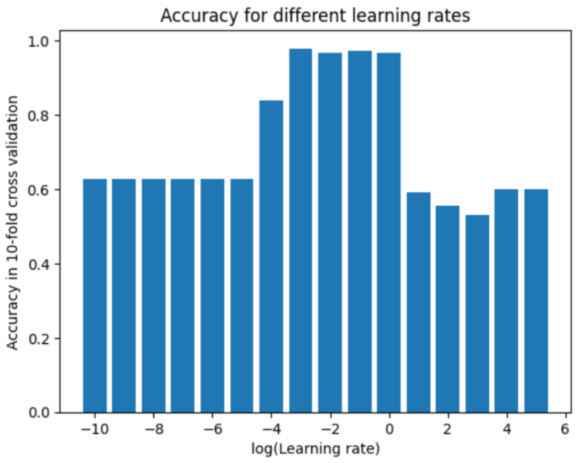

Figure 10: Accuracy in 10-fold cross validation for different learning rates, ranging from 10^{-10} to 10^{5}.

As a third action, we look at the amount of neurons in the hidden layers of the neural network. The amount of principle components used remained constant at 5, and the learning rate at 0.001. We first only used one hidden layer, and looked at the optimal amount of neurons in this layer. Results of this are visible in Figure 11. We see that from 18 neurons upward, we get similarly good results. It is interesting to notice that adding more neurons does not improve accuracy, but neither does it decrease it. Doing the same experiments for the amount of neurons in a second and third layer, we find that it is best to not include a third (of higher) layer of neurons. This is possibly because this causes the network to have too much weight parameters, while not adding any complexity that is needed for the problem. The optimal amount of neurons in a second layer gives similar results as for the first layer: As long as more then approximately 15 neurons are put in the layer, the accuracy becomes better. We do however notice that too much neurons in the network indeed decrease accuracy.

Using the results found before we experimented even more, until we found our best model so far, using the first 10 principle components, two hidden layers with each 30 neurons and a learning rate of 0.001. This model has an accuracy of 98.2% in 10-fold cross validation.

5 CONCLUSION

Our reference accuracy was the one found in the 1995 study, that used a decision tree classifier to find an estimated accuracy of 97.5%. With the Gaussian Naive Bayes Classifier, an optimal classifier was found that correctly classified 96% of the data points. With the multi-layer perceptron classifier we managed to find an accuracy of 98.2%, which is a slight improvement of the found accuracy in the original study. Even a slight improvement could, however, save lives in medical research.

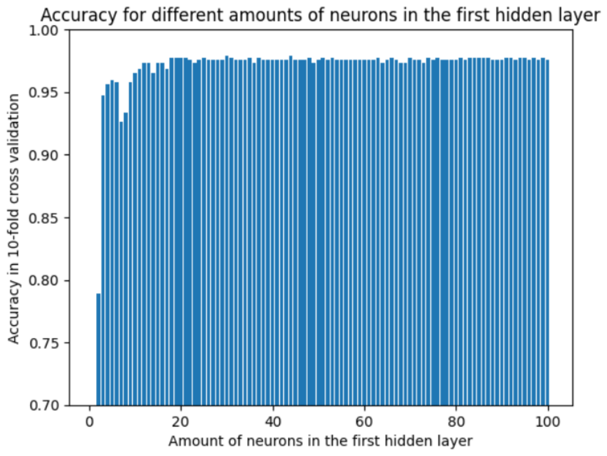

Figure 11: Accuracy in 10-fold cross validation for different amounts of neurons in the first hidden layer, ranging from 1 to 100.

6 DISCUSSION AND FUTURE RESEARCH

We think it very much possible to improve our classifiers on the data even further. If we had had more computational power, we could have improved our Gaussian Naive Bayes Classifier by trying all $2^{30} - 1$ possible feature sets, as was explained in chapter 3. This might give us an even better match for our data set, that will lead to higher accuracies. The neural network classifier could also be improved by trying even more different combinations of hyper parameter values. A point of critism for the research we have preformed here, is that we have only looked at how the learning rate, amount of principal components and amount of neurons vary seperately, namely while the two other values were kept constant. This is imprecise, as those three factor may vary codependently. A higher learning rate might for example have a different amount of principal components that is ideally combined with it, than a lower learning rate. The neural networks could also be further explored by also varying the solver method, amount of iterations, and the activation function.

REFERENCES

[1] Rafael Blanquero, Emilio Carrizosa, Pepa Ramírez-Cobo, and M. Remedios Sillero-Denamiel. 2021. Variable selection for Naïve Bayes classification. *Computers Operations Research* 135 (2021), 105456. https://doi.org/10.1016/j.cor.2021.105456
[2] W.N. Street O.L. Mangasarian and W.H. Wolberg. July-August 1995. Breast cancer diagnosis and prognosis via linear programming. , 570-577 pages.
[3] Mangasarian Olvi Street Nick Wolberg, William and W. Street. 1995. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B.