Microarray data analysis in prediction of breast cancer metastasis

(Analiza mikromacierzy DNA w predykcji występowania przerzutów raka piersi)

Stanisław Wilczyński

Praca magisterska

Promotor: dr Jan Chorowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

30 sierpnia 2019

Abstract

One of the major problems of modern medical studies is the prediction of future metastases of breast cancer diagnosed patients. Although metastases are the main cause of death of such subjects and despite the effort in uncovering the underlying mechanism, there was only a minimal improvement in the field in recent years. A tool for distinguishing people having high risk of developing a distant metastasis from the low risk group could greatly reduce the number of patients unnecessarily receiving chemotherapy. Data mining seems to be a great solution for automating the process of such diagnosis as it could allow to predict the outcome of the disease based on hidden, almost impossible to detect correlations or dependencies within the data. One of the most appealing candidates for highly informative data is the level of gene expression collected from the tumor cells. Unfortunately, the analysis of gene expression can be problematic, due to the number of features (measurements of expressed genes) being much higher than the number of samples (patients). In this thesis, we focus on comparing different classification, dimensionality reduction methods in order to find the ones applicable in the field of prediction of future metastases of breast cancer diagnosed subjects and showing that these techniques could help in providing automatic diagnosis and deciding about potential highly toxic treatment.

Jednym z głównych celów innowacyjnych badań medycznych jest przewidywanie wystąpienia przerzutów raka piersi. Mimo tego, że są one jednym z częstszych i nierzadko śmiertelnych powikłań oraz znacznych nakładów na analizę czynników je wywołujących, w ostatnich latach nie zaobserwowano znaczącego postępu w ich prognozowaniu. Narzędzie pozwalające rozróżnić pacjentów z grupy wysokiego ryzyka wystąpienia przerzutów pozwoliłby znacznie zredukować liczbę chorych, którzy niepotrzebnie zostają poddani chemioterapii. Metody eksploracji danych wydają się świetnym kandydatem do procesu automatyzacji takiej diagnozy ze względu na możliwość uwzględnienia niewidocznych dla człowieka korelacji między czynnikami wpływającymi na rozwój choroby. Jednym z najbardziej popularnych rodzajów danych służących do tego typu badań są rekordy dotyczące poziomu ekspresji genów, pobrane z komórek rakowych. Niestety, taka analiza może być skomplikowana ze względu na liczbę próbek (pacjentów) będącą znacznie mniejszą niż liczba zmiennych (poziomy ekspresji genów). W tej pracy skupiamy się na porównaniu różnych metod klasyfikacji oraz redukcji wymiarowości, w celu znalezienia tych najlepiej radzących sobie z przypisywaniem pacjentów do odpowiednich grup ryzyka. Pokażemy również, że mogą one pomóc w stworzeniu automatycznego mechanizmu diagnostycznego decydującego o wysłaniu chorego na toksyczną terapię.

Contents

1 Introduction **7**

1.1 Overview 7

1.2 Gene expression data 8

2 High dimensional data analysis **11**

2.1 Model assessment and selection 11

2.2 Classifiers for high dimensional data 17

2.2.1 Logistic regression and its modifications 17

2.2.2 Discriminant Analysis and Nearest Shrunken Centroid 21

2.2.3 Linear Support Vector Classifier 24

2.2.4 Ensemble learning 27

2.3 Dimensionality reduction 30

2.3.1 Feature extraction - unsupervised methods 31

2.3.2 Feature extraction - supervised methods 35

2.3.3 Feature selection 38

3 Exploratory analysis **41**

3.1 Basic statistics, visualizations and normalization 41

3.2 Correlation study 47

4 Implementation details and results **49**

4.1 Implementation 49

4.2 Classification performance 52

4.2.1 Regularized classifiers and feature extraction 52

4.2.2 Feature selection 55

5 Conclusions **59**

Bibliography **61**

Chapter 1

Introduction

1.1 Overview

One of the major problems of modern medical studies is the prediction of future metastases of breast cancer diagnosed patients. Although metastases are the main cause of death of such subjects and despite the effort in uncovering the underlying mechanism, there was only a minimal improvement in the field in recent years. The main concern is the lack of reliable method to classify patients to *poor prognosis* and *good prognosis* groups. The first one refers to people that would develop a distant metastasis within five years since first tumor appearance. According to [46] a tool for distinguishing between these two classes could greatly reduce the number of subjects unnecessarily receiving chemotherapy. Data mining seems to be a great solution for automating the process of such diagnosis as it could allow to predict the outcome of the disease based on hidden, almost impossible to detect correlations or dependencies within the data. One of the most appealing candidates for highly informative data is the level of gene expression collected from the tumor cells. There are numerous examples of such data proving to be useful in the field of tumor classification ([41], [18], [15], [30]). Unfortunately, the analysis of gene expression can be problematic, due to the number of features (measurements of expressed genes) being much higher than the number of samples (patients), as the extraction of gene expression data is costly. Such relation causes a phenomenon called *curse of dimensionality* ([22, p. 22-26]). In order to decrease this effect the dimensionality reduction techniques are often applied ([3], [29], [33]). Dimensionality reduction techniques even allow to use with success one of the most popular and state-of-the-art techniques such as neural networks ([27]), which usually do not work in the settings where number of features is significantly higher than number of samples. Other important techniques build sparse representation ([21]) or use strong regularization ([22, p. 649-666]) in constructed models. However, in scope of prediction of metastases data mining methods are not so popular. In fact, current research focus mostly on purely biological markers ([49], [5]), using prior knowledge based on protein-protein interaction networks

([12]) or operating on very small samples (less than 100 patients) and utilizing only basic classification methods ([46], [50]). Such reluctance to adopt more advanced models is usually caused by the requirement of having a tool that is easy to interpret. In terms of gene expression data it refers to being able to highlight the genes having the highest influence on the response.

In this thesis we present different classification and dimensionality reduction methods, discuss their advantages and weaknesses and apply them to the gene expression data. The main contribution is showing that these techniques could help in providing automatic diagnosis and deciding about potential highly toxic treatment. The code for all the experiments performed is available at `github.com/sjwilczynski/geneExpr`.

In section 1.2 we explain the medical notions in the simplified form to make it understandable for a person with little biological knowledge and describe how our data set was collected and processed. In section 2 we discuss the methods used for classification and dimensionality reduction. In section 3 we statistically and graphically inspect our data. In section 4 we present the results of our analysis together with implementation details and discuss them. We also consider medical applications and show how specific requirements in the field imply some tuning and choices of our models.

1.2 Gene expression data

Biological background

In this section we briefly explain biological notions used throughout the thesis, based on [32] and [9, chapters 11,12]. The *cell* is the smallest structural and functional unit of a living organism that is able to run a biological processes (e.g metabolism, growth, reproduction). Every cell contains a *DNA* (deoxyribonucleic acid) which is a chemical compound carrying the information needed to develop and direct the activities of the cell. DNA is composed of two chains coiling around each other, which are made of four basic chemical units called *nucleotides*. A *DNA sequence* is a particular arrangement of these nucleotides within a chain. This arrangement is what codes a unique characteristic of an organism. A *genome* is an entire DNA sequence of a living creature. A *gene* refers to a specific fragment of genome that has a distinct purpose, especially carries instructions for producing a RNA molecule and then a protein. *Proteins* and *RNA molecules* are basic functional molecules in organisms, that among many applications within a cell control chemical reactions, regulate growth and participate in communication in biological pathways. In case of DNA mutation, the information contained can be corrupted, causing a production of undesirable protein and disruption of standard flow of biological process. Such disturbance, if undetected by immunological system can lead to diseases develop-

ment.

Gene expression is the process during which the instructions coded in gene are used to synthesize its products. The level of expression corresponds to the amount of molecules created during this process. The gene is *upregulated* if it synthesizes more, and *downregulated* if synthesizes less when compared with such production in some reference conditions, assumed to be a homeostasis. The state-of-the-art technique for measuring relative expression levels is gene expression profiling using DNA microarrays. The main application of microarrays is to discover how gene functioning changes in response to genetic and environmental variations. *Microarray* is a glass or plastic surface with microscopic spots, placed in regular intervals. Each spot contains picomoles of a specific DNA sequence, known as *probe*, which interacts with tested and reference gene expression product, resulting in emitting measurable signal. By using products from two different cells (e.g cancer and healthy) on the same microarray we can measure the ratio of expression. These ratios are later gathered to form a vector of expression ratios of different genes. In order to conduct a research aimed at specific disease we collect vectors for different individuals with common condition and form a matrix. This matrix is referred to as *gene expression data*. As stated in [24] there are two main sources of variations in microarray measurements. The first one called *interesting variation* corresponds to a case when, for example, highly upregulated gene causes an illness and there is a noticeable difference between diseased and normal tissue. However, the other one called *obscured variation* is induced by discrepancy in the process of sample preparation, microarray manufacturing or even collecting the results from an array. Due to such diversity of possible factors influencing the outcome it is crucial to apply normalization methods before actual data analysis in order to avoid misleading or even completely incorrect conclusions.

A tumor is a mass formed by a group of abnormal cells. This abnormality refers to disruption of the division and apoptosis processes. However, not all tumors are cancerous (malignant). Malignant tumors are characterized by uncontrolled growth and division that damages nearby tissues. The spreading of cancerous cell from primary source to other parts of the body is called metastasis. Simplifying, it can be described as partial detachment of the tumor and its transport with blood to other part of the organism. Such secondary pathological sites are called metastases and are particularly dangerous. The gene expression data is of great interest in cancer related research as the alteration of cancerous cell division pattern is usually caused by either a mutation in a set of genes or a change in the expression level.

Data description

The data set used in the thesis consists of 969 patients. Each patient is represented by the measurements of the expression level from his cancerous cell. Such measurement has its unique identifier (e.g *GSM177885*) referring to Gene Expression

Omnibus ([13], [4]). Gene Expression Omnibus is a public database storing various forms of high-throughput functional genomic data. The repository is maintained by US organization National Center for Biotechnology Information and the data is uploaded and reviewed by scientific community. The *GSM* prefix indicates that this identifier corresponds to a single sample/cell. Other prefixes refer to series of data gathered within the same conditions (*GSE*), platform (specific microarray technology) used to perform the experiment (*GPL*), collections of samples assembled by GEO staff (*GDS*). In case of our data set, it was previously studied in [5] and was assembled from 10 public microarray data sets, namely: GSE25066, GSE20685, GSE19615, GSE17907, GSE16446, GSE17705, GSE2603, GSE11121, GSE7390, GSE6532. Each series was obtained using Affymetrix Human Genome HG-U133 Plus 2.0 and HG-U133A arrays, which are the types of the mentioned earlier platforms. As explained before it is essential to preprocess data before the analysis. Therefore, each data set was first normalized using quantile normalization described in [7], then further processed using robust multi-array average (RMA) probe summary algorithm from [24]. Due to the fact that these series could have different number of genes expression measured they were combined together on the basis of HG-U133A array probe names resulting in reduction of the number of variables to 12179. Each variable corresponds to a level of expression of a specific gene (e.g. RPL41). Each patient is also labeled. Label 1 corresponds to people who experienced metastatic event within 5 years from being cancer diagnosed. 0 corresponds to people who had the first follow up after 5 years or no metastasis at all. There are 393 and 576 samples with labels 1 and 0 respectively.

Chapter 2

High dimensional data analysis

As stated in the introduction our aim is the prediction of future metastases which in terms of our data set translates to binary classification problem using the labels. There are numerous examples where classification methods solve very complex problems e.g customer target marketing, document categorization and filtering or computer vision tasks ([1]). However, in the field of biomedical studies not all of them are applicable. The reason is the high dimensional, complex structure of our data, particularly the relation between the number of samples and number of features: $n \ll p$. Such relation is caused by the relatively low availability of samples and high costs of extracting, processing and labeling new ones. These activities have to be done in laboratory by professionals, in comparison to creation of data sets containing images where anyone can participate in e.g paid labelling procedure. Apart from that huge number of features causes a phenomenon called *curse of dimensionality*. As described in [25, chapter 6.4.3] the increase in the number of variables can improve the model only if the added features are signals, i.e. associated with the response. On the other hand, if they do not influence the response at all, they introduce noise. This may lead to overfitting (high variance) and significantly larger test error. The source of this problem is that with increasing dimension the sample space becomes very sparse and all pairwise distances between data points are alike. Furthermore, huge number of variables introduces more complexity in most models as the number of parameters to estimate grows. In extreme cases it may even lead to lack of convergence in parameter estimation procedure and in consequence lack of model stability. In order to tackle this issue we will use robust classifiers and dimensionality reduction methods described in further sections.

2.1 Model assessment and selection

Before introduction of dimensionality reduction and classification techniques we focus on showing how to choose from a variety of different models and how to assess them. These two tasks are quite different and there exists a variety of methods with

distinct tradeoffs. Model assessment typically refers to estimation of generalization performance of the model - its predictive power on new, unseen data. In scope of model selection we can distinguish two task. The first one is the choice of model *hyperparamters* - parameters of the model which are not estimated during training phase but rather provided as an input. An example is a k in k-Nearest-Neighbour classifier or any regularization constant (e.g section 2.2.1). The other one is the choice of learning algorithm - not only we want to find the best parameters but also be able to determine which classifier is best-suited for our problem. One key difference between model assessment and selection is that in case of the latter we only need to measure relative performance of the models - they may be biased on condition that bias influences all models equally. In case of model assessment we are only interested in unbiased estimators of generalization performance.

In order to further elaborate on evaluation techniques we introduce two key notions: model *bias* and *variance*. Bias refers to the magnitude of difference between model's mean prediction and mean true value. Big bias indicates that our model is not able to take into account most of the dependencies within our data set - hints that it is either too simple for the data or its assumptions are not held (undefitting). Variance is a measure of model's prediction instability. Big variance suggests that it is very sensitive to small perturbations in the data set - indicates that it is too complex to our data and has too many degrees of freedom (overfitting). These two values are usually complementary. To see this, consider a model assessment task. We cannot estimate generalization performance using the data set used for fitting the model, as on this set we can only measure the bias that tends to decrease with model complexity. However, large variance may cause the model to perform poorly on new data. Therefore, we split our data to two parts train set and test set. On the train set we fit the model and on the test set we check its prediction performance. As the latter set was not used during fitting process, if the set was split in reasonable manner, the prediction accuracy on the test set is a good estimate of the generalization performance. Such split allows us to capture the crucial statistical phenomenon, i.e *bias-variance tradeoff* (figure 2.1).

The figure shows in blue prediction error curves on the train set and in red prediction error curves on test set for 100 different splits. The solid curves are the mean curves. On the x axis we have a model complexity. The left side of the plot is region of high bias and low variance - model does not capture dependencies in the data but the predictions are very stable. Then as the model is getting more complex and accurate for the data, the variance and bias together decrease. However, for some model complexity the variance reaches its minimum and starts increasing while bias still decreases. That is where the name of the phenomenon comes from - in order to minimize the prediction error we cannot only use more elaborate models and decrease bias. We need to find a point for which the sum of contributions of errors resulting from bias and variance is minimized. That is our goal in the process of model assessment - choose the one capturing most of the regularities in our data

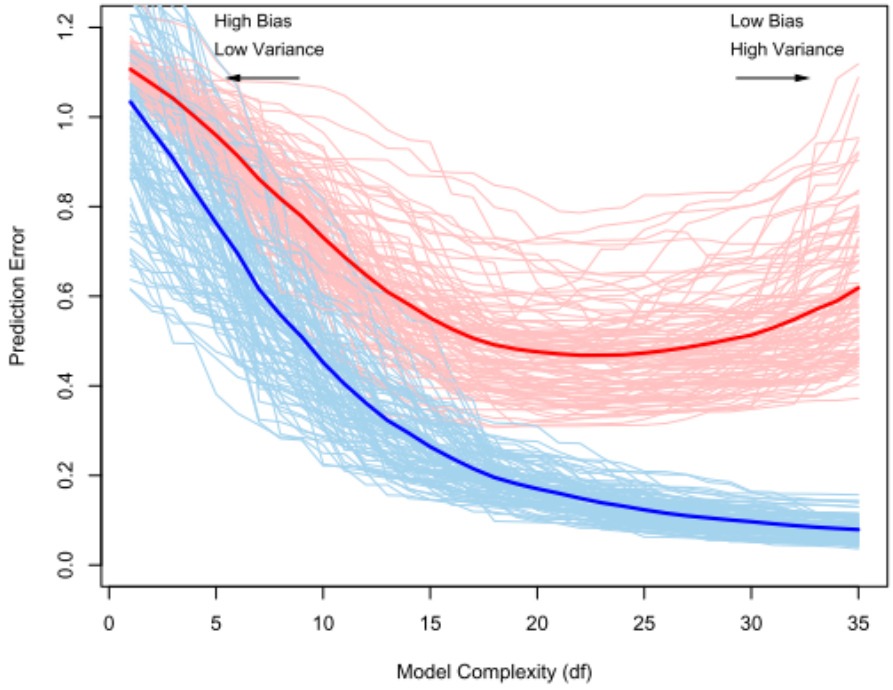

Figure 2.1: Bias variance tradeoff. The picture comes from [22] - figure 7.1

but also with good generalization performance. The question arises of how to fairly split our data set to train and test sets. The fully random approach may result in proportions of classes in both sets to be different from the proportions in the original one. This may result in biased evaluation of model performance. In order to discard this problem we need to use *stratified* sampling - the data is split based on class labels to two groups and the random sampling takes places in both of them to keep the same proportions of classes in train and test sets. The sizes of the corresponding parts are the other issue. We want a test set to be representative so that it provides good estimation of generalization performance, but we also want to keep the training part as large as possible as the number of data samples needed to properly train the model increases with model complexity. The typical choices of train/test split ratio are 2/1, 3/2, 3/1, 4/1 but the decision depends on specific problem.

Nonetheless, train/test split is not enough when we want to tune hyperparameters of the model and estimate its performance on unseen samples. If we used the performance on the training set to select the model, we would choose the one that overfits most on the train data instead of the one that generalizes best. On the other hand, if we use the performance on the test part, it would be positively biased estimate of generalization performance. Although the model did not use test set during learning process, it was used during selection, favouring models performing well on this particular samples from population. Therefore, instead of two-way split

we require a three-way split, i.e train/validation/test. The models learn using only the train part with different values of hyperparamters. Then we compare the performance of these models on the validation set and select the best one. Finally, the generalization performance is estimated using the test part. Again the split should be performed using stratified sampling and typical train/validation/test ratios are: 2/1/1, 3/1/1.

However, two-way split is not the only method for the estimation of the generalization performance and train/validation split is not the only method for model selection. The most popular alternative technique is k-fold cross-validation. It tackles the issue of holdout method, i.e not using a big part of data for training and another larger part for testing. We divide the data set in stratified manner to equally sized k folds. In the i-th iteration we train the model using all folds but i-th and then utilize i-th fold as a test set. After k repetitions we have k estimates of the generalization performance and we use their mean as the final estimate. If we apply cross-validation for model selection we do the above steps for each choice of hyperparameters and use the mean performance on test folds as the selection metric. The important properties of cross-validation are:

- Instead of providing single estimate of generalization performance it gives k values - we can use them not only to compute mean estimate but also standard deviation in order to see how stable is our estimate.
- Each data point is used both in training and in test/validation phase but never in both simultaneously - this is particularly important for small data sets when we do not want to introduce pessimistic bias resulting from too few samples used for model training.
- It is much more computationally demanding then train/test split method - it fits k model instead of one.

The last point is the reason why cross-validation is rarely seen in deep learning problems - training a neural net is usually too costly to be performed that many times especially for hyperparameter tuning. Also the data sets are often large enough to be able to holdout a validation and test parts without risking that too few samples will be used in learning phase to train the model properly. Cross-validation has one more issue - how to choose k. As stated in [36] with increasing k the bias of performance estimator decreases but variance and computation cost grow. On the other hand, for small data sets small values of k result in higher variance because of the effect of random sampling. Therefore, typical choice for very limited data sets is n, where n is the number of samples. In other cases, $4, 5, 10$ are most commonly used.

So far we discussed model assessment and selection in terms of hyperparameter tuning. However, the most interesting issue in our study is the selection of the

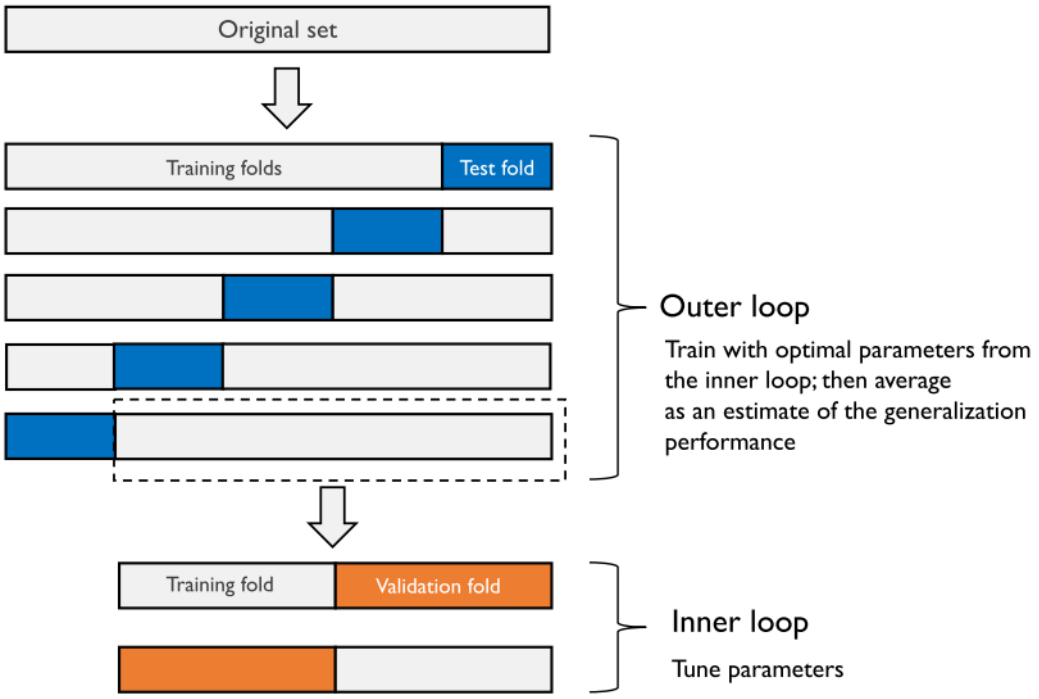

Figure 2.2: Nested cross validation - picture comes from [36], figure 22

best algorithm. As it turns out we cannot apply standard cross validation for this task as it usually results in biased outcomes ([36], [11]). Therefore, we use *nested cross-validation* introduced in [47]. It is presented graphically in figure 2.2.

The main idea is a usage of two nested cross-validation loops. In the inner loop we tune model hyperparameters and in the outer loop we estimate its performance in order to compare fairly different learning algorithms. For our particular task we would apply 5x2 setup (5 folds in outer loop, 2 in the inner one). As we use this technique for selection among different methods we still keep the holdout test set to later assess the performance of the chosen approach.

Performance metrics

Although we presented a few model selection and assessment techniques we still have to refer to performance metrics as they are key part of this process. The simplest one is the accuracy - percentage of correctly classified samples among the data. However this metric is not perfect, particularly in case of imbalanced classes. Imagine an artificial binary classification problem where one of the classes is represented only by 10% of the data. Then a classifier that always assigns the other label has 90% accuracy - quite high for a model that does not capture any regularities in our data and is completely useless. Therefore, we use more reliable metrics for both model

comparison and generalization performance estimation. Let us assume that we have a trained model and a data set on which we want to evaluate it. Let us introduce four factors:

- True positives (TP) - number of samples that are from class 1 and model predicted 1.
- False positives (FP) - number of samples that are from class 0 but model predicted 1.
- False negatives (FN) - number of samples that are from class 1 but model predicted 0.
- True negatives (TN) - number of samples that are from class 0 and model predicted 0.

In case of our data set class 1 correspond to patients experiencing metastatic event within 5 years from the first diagnosis. Using these four values we can define three classification metrics:

- Precision - $\frac{TP}{TP+FP}$
- Recall (or true positive rate (TPR), sensitivity) - $\frac{TP}{TP+FN}$
- F1-score - harmonic mean of precision and recall

By using our case as an example precision measures how many patients that are assigned to class 1 really experienced a metastatic event. On the other hand, recall measures how many of the patients who experienced a metastatic event actually are assigned to class 1. In complex classification problems there is tradeoff between these two metrics - optimizing one of them results in decrease in the other. However, in our case we are more interested in having high recall - as the error of not assigning the patient to high risk group and in consequence not applying a therapy in a case he is to experience a metastatic event is much more harmful than applying treatment for patient of low risk. The first error may result even in patient's death. Still, for model selection it is crucial to have a metric that we can apply for direct comparison. One of the potential choices is F1-score, combining both precision and recall in one measure, but in our study we use area under the curve (AUC) of receiver operating characteristic curve (ROC). ROC curve is the name for the plot of TPR against False Positive Rate (FPR, $\frac{FP}{FP+TN}$). Using the AUC we can measure how good is our model at separating positive from negative class and also find a good threshold for decision boundary (by default if our model returns probabilities it is 0.5). To draw the ROC curve we need to rank our samples decreasingly by classifier score (assuming that higher score refers to greater certainty of positive class). We start at point $(0, 0)$ and assume that we have *pos* samples labeled as 1 and *neg* samples

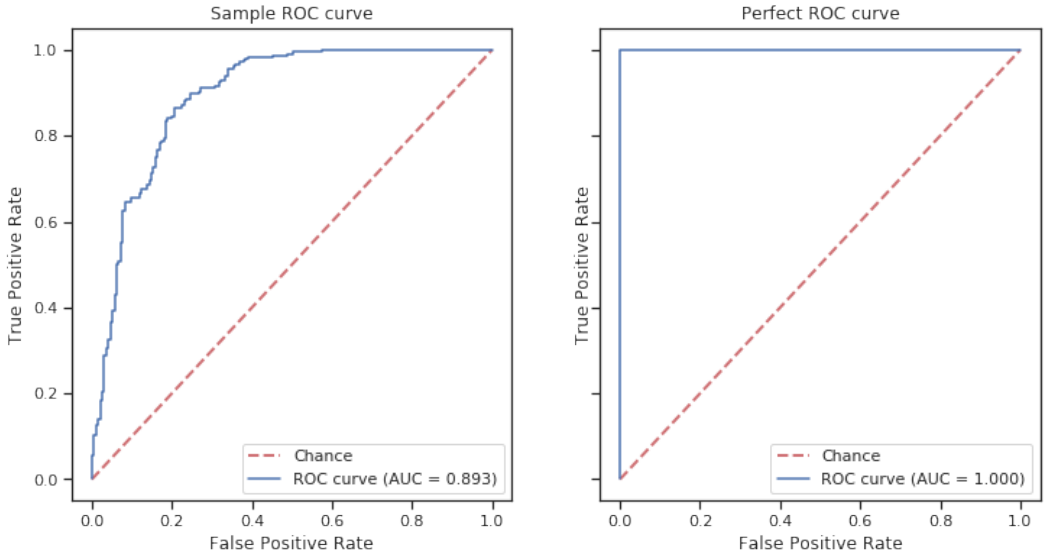

Figure 2.3: Example ROC curves

labeled as *neg*. Starting from the top of the ranking if the label is 1 we move up by $\frac{1}{pos}$ and if the label is 0 we move right by $\frac{1}{neg}$. To visualize this on figure 2.3 we present two example curves - the one on the right is the curve for the perfect classifier and the one on the left is a curve for more an artificial classification problem modeled using logistic regression.

If we used a random classifier for balanced classes we would obtain a ROC curve equal to $x = y$ line. Therefore this line is treated as baseline and is present in the plots. The AUC for the perfect classifier is 1 and for the baseline is 0.5. The AUC may be interpreted as expectation that uniformly drawn positive sample is ranked before a uniformly drawn negative sample. As mentioned before in our problem we care much more about high TPR than low FPR, therefore we will exploit the features of ROC and we will adjust decision threshold values.

2.2 Classifiers for high dimensional data

2.2.1 Logistic regression and its modifications

One of the most popular classifiers for binary response is logistic regression (LR). It is a counterpart of linear regression for regression problems and was proven to be valuable method for microarray analysis ([54]). The model aims at estimating the conditional probability of class membership. It is described by the following

formula:

$$
\begin{aligned}
&P(y = 1|x, \beta) = \sigma\left(\beta^T x\right) \\
&\sigma(t) = \frac{1}{1 + e^{-t}} \\
&x = (1, x_1, \ldots, x_p)^T \\
&\beta = (\beta_0, \ldots, \beta_p)^T
\end{aligned}
$$

where x is the vector of features with additional 1 as first coordinate to abbreviate the notation, β is the vector of coefficients including intercept, σ is logistic sigmoid function and is used to squeeze the output to range $(0, 1)$ and P is the conditional probability of class 1 of a sample described by variables x. The interpretation of logistic regression coefficients is quite different from linear regression due to the usage of sigmoid function. In terms of the latter an increase of one unit of variable x_i corresponds to increase of the output by β_i. The similar relation for LR can be expressed using log odds (the ratio of probabilities of event occurring and not occurring):

$$
\log\left(\frac{P(y = 1|x, \beta)}{1 - P(y = 1|x, \beta)}\right) = \log\left(\frac{P(y = 1|x, \beta)}{P(y = 0|x, \beta)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p
$$

Above equation clearly indicates that an increase of one unit of variable x_i corresponds to increase of the log odds by β_i and this implies and increase of odds ratio by a factor of $\exp(\beta_i)$. Similarly β_0 corresponds to the log odds in the baseline conditions. Such relatively simple and probabilistic interpretation made the model very popular. However, it has some disadvantages. First of all, it does not incorporate a possibly complicated correlation structure or interactions between variables. Secondly, it could perform worse with highly correlated features or with predictors having no influence on the response. Moreover, coefficients could be infinite in case of complete separation, i.e when one feature can distinguish completely between classes. Nonetheless, LR is widely used, also due to uncomplicated learning scheme. Similarly to linear regression coefficients in LR are calculated using maximum likelihood estimation. The likelihood function takes the form:

$$
\begin{aligned}
L(\beta) &= \prod_{i=1}^{n} P(Y = y_i|x_i, \beta) \\
&= \prod_{i=1}^{n} P(Y = 1|x_i; \beta)^{y_i}(1 - P(Y = 1|x_i; \beta))^{(1-y_i)} \\
&= \prod_{i=1}^{n} \sigma\left(\beta^T x_i\right)^{y_i} \left(1 - \sigma\left(\beta^T x_i\right)\right)^{(1-y_i)}
\end{aligned}
$$

And the negative log likelihood is:

$$
nll(\beta) = -\sum_{i=1}^{n} y_i \log \sigma\left(\beta^T x_i\right) + (1 - y_i) \log\left(1 - \sigma\left(\beta^T x_i\right)\right)
$$

In the two above equations $x_1, \dots x_n$ denote samples and y_i corresponding labels. In order to maximize the coefficients we calculate the first derivative of the negative log likelihood function as in [22, chapter 4.4.1]:

$$\begin{aligned}\frac{\partial nll(\beta)}{\partial \beta} &= \sum_{i=1}^{n} \left(\sigma\left(\beta^T x_i\right) - y_i\right) x_i \\ &= \left[\left(\sigma(X\beta) - Y\right)^T X\right]^T\end{aligned}$$

We can clearly see that the right side is not linear in β and in fact equation $\frac{\partial nll(\beta)}{\partial \beta} = 0$ cannot be solved analytically. Therefore, the maximizing coefficients are found using Newton-Raphson method, extensions of gradient descent like SAG ([37]) or SAGA ([14]) and other e.g. Liblinear ([17]).

However, classic logistic regression has one more drawback. The maximization of likelihood function usually provides good (in terms of model usability) estimates of the model parameters. Still, such estimation may be unstable (model with high variance) when the number of data samples is relatively small. The problem is even more evident when $n < p$. Without any constraints in order to fit $p+1$ coefficients we need at least $p+1$ data points, otherwise there are infinitely many solutions. For our data set $n \ll p$ and LR cannot work in such settings. In order to tackle this problem we consider a more probabilistic (Bayesian) point of view - instead of maximizing the likelihood, we will maximize posterior probability of model parameters:

$$\begin{aligned}P(\beta|X,Y) &= \frac{P(Y|\beta,X)P(\beta)}{P(Y)} \\ \hat{\beta} = \arg\max_{\beta} P(\beta|X,Y) &= \arg\max_{\beta} P(Y|\beta,X)P(\beta)\end{aligned} \tag{2.1}$$

The first equation is derived directly from the Bayes rule and we can see that in the equation 2.1 we can neglect the denominator as it is independent of β. So the first term on the right side of 2.1 is the likelihood function and the second is the prior distribution over the values of β. The maximum likelihood estimation is a special case of maximum a posteriori estimation, i.e. when we assume uniform distribution for model parameters. Choice of MAP estimates over MLE estimates has a significant effect on our learning scheme. Instead of minimizing the negative log likelihood we minimize the expression:

$$\begin{aligned}J(\beta) &= -\log P(Y|\beta,X)P(\beta) = -\log P(Y|\beta,X) - \log P(\beta) \\ &= nll(\beta) - \log P(\beta) = nll(\beta) - \sum_{i=0}^{p} \log P(\beta_i)\end{aligned} \tag{2.2}$$

Of course $\arg\max_\beta P(\beta|X,Y) = \arg\min_\beta J(\beta)$. The above equation indicates that the mentioned influence depends only on the choice of prior distribution for

model parameters. Two popular alternatives are zero-mean normal distribution and zero-mean Laplace distribution:

$$\mathcal{N}(0, \sigma^2) \sim \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$
$$Lap(0, b) \sim \frac{1}{2b} \exp\left(-\frac{|x|}{b}\right)$$

Putting these two distributions into 2.2 gives us following versions of J to minimize:

$$J_{\mathcal{N}}(\beta) = nll(\beta) + \sum_{i=0}^{p} -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \frac{\beta_i^2}{2\sigma^2}$$
$$J_L(\beta) = nll(\beta) + \sum_{i=0}^{p} -\log\left(\frac{1}{2b}\right) + \frac{|\beta_i|}{b}$$

We can simplify these equations as constants can be discarded during minimization:

$$\arg\min_{\beta} J_{\mathcal{N}}(\beta) = nll(\beta) + \frac{1}{2\sigma^2}\sum_{i=0}^{p}\beta_i^2 = nll(\beta) + \frac{1}{2\sigma^2}\|\beta\|_2^2$$
$$\arg\min_{\beta} J_L(\beta) = nll(\beta) + \frac{1}{b}\sum_{i=0}^{p}|\beta_i| = nll(\beta) + \frac{1}{b}\|\beta\|_1$$

where $||\cdot||_2, ||\cdot||_1$ denote euclidean (l_2) and absolute (l_1) norms respectively. As we often want to calibrate the influence of the prior we rewrite the above to form:

$$\arg\min_{\beta} J_{\mathcal{N}}(\beta) = nll(\beta) + \frac{\lambda_2}{2}\|\beta\|_2^2$$
$$\arg\min_{\beta} J_L(\beta) = nll(\beta) + \lambda_1\|\beta\|_1$$

as increase in either of λ makes the prior distribution more significant. This two schemes are called L_2 and L_1 regularizations. It is also worth noticing that their common general structure can be written as:

$$\hat{\beta} = \arg\min_{\beta} \mathcal{L}(X, y, \beta) + \lambda Pen(\beta) \tag{2.3}$$

where the first term is called loss function (in our case negative log likelihood) and the second term is called the penalty, where λ is the tuning parameter which controls the magnitude of the effect of regularization.

We can see that these two regularizations aim at pushing the coefficients towards zero in order to reduce overfitting and enhance model generalization capabilities. Especially in case of our data, regularization allow us to use LR when $n \ll p$. Yet

these two schemes differ considerably. As described in [31, chapter 13.3.1] the l_1 norm favours sparse models (lots of coefficients equal to 0) and l_2 favours models with lots of coefficients having small (but most of the times nonzero) values. The intuition behind this is visible in the formula for the penalty. Consider an example vector $x = (1, \varepsilon)$ where $0 < \varepsilon < 1$. Then the norms are $\|x\|_1 = 1 + \varepsilon, \|x\|_2^2 = 1 + \varepsilon^2$. Now we check how the norms change when we subtract the vectors $(\delta, 0)$ and $(0, \delta)$:

$$\|x - (\delta, 0)\|_1 = 1 - \delta + \varepsilon, \quad \|x - (\delta, 0)\|_2^2 = 1 - 2\delta + \delta^2 + \varepsilon^2$$
$$\|x - (0, \delta)\|_1 = 1 - \delta + \varepsilon, \quad \|x - (0, \delta)\|_2^2 = 1 - 2\varepsilon\delta + \delta^2 + \varepsilon^2$$

We can see that regularizing the first, larger coordinate in l_2 results in much more significant decrease in norm than doing so for the smaller one. Therefore, it is not very probable that L_2 regularization will truncate any tiny coefficients to 0 as doing so has almost negligible influence on the norm value. In case of l_1 the difference in norms is always the same, regardless the actual value of coefficients. That is why L_1 regularization is likely to reduce parameters to 0 if they have minor effects on the likelihood.

In terms of linear regression these two methods are called ridge regression and LASSO. There also exists a combination of these two i.e. Elastic Net where the penalty is the linear combination of L_1 and L_2 regularization. As different models can use such regularization in the further chapters we will refer to them as L_1, L_2 and Elastic Net penalties.

2.2.2 Discriminant Analysis and Nearest Shrunken Centroid

As stated in [22, chapter 18.2] there are two good classifiers that can be used as baseline in high dimensional setting. The first one is Regularized Discriminant Analysis (RDA), an extension of Linear Discriminant Analysis (LDA) ([22, chapter 4.3]). The assumption of LDA is that samples from each class come from multivariate normal distribution with the same covariance matrix Σ. In terms of two classes it can be written as:

$$x_i \sim \begin{cases} \mathcal{N}(\mu_0, \Sigma) \text{ if } y_i = 0 \\ \mathcal{N}(\mu_1, \Sigma) \text{ if } y_i = 1 \end{cases}$$

where x_i, y_i denote i-th sample and label in the data set respectively. Let us denote corresponding densities as f_0 and f_1. Assuming a prior probabilities of class membership $\pi_0, \pi_1, \pi_0 + \pi_1 = 1$ one can look at the log ratio of class membership

probabilities for sample x_i:

$$\begin{aligned}
\log \frac{P(Y=1|x_i)}{P(Y=0|x_i)} &= \log \frac{f_1(x)}{f_0(x)} + \log \frac{\pi_1}{\pi_0} \\
&= \log \frac{\pi_1}{\pi_0} - \frac{1}{2}\left(\mu_1 + \mu_0\right)^T \Sigma^{-1} \left(\mu_1 - \mu_0\right) + x^T \Sigma^{-1} \left(\mu_1 - \mu_0\right) \\
&= \delta_1(x_i) - \delta_0(x_i), \text{ where} \\
\delta_k(x) &= x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k
\end{aligned} \tag{2.4}$$

where δ_1, δ_0 are discriminant scores for classes $1, 0$ respectively. As the values of $\mu_1, \mu_0, \pi_1, \pi_0, \Sigma$ are not known, mean sample estimates, class proportions and sample covariance matrix are used. The assumption of equal covariance matrices is why this method is called *linear*. By looking at the ratio of class membership probabilities we can see that quadratic parts from exponent cancelled out resulting in the decision boundary (set where $P(Y = 1|X = x_i) = P(Y = 0|X = x_i)$) to be linear in x_i - a hyperplane. However, if we assume that the covariance matrices are not equal we get a quadratic decision boundary and a method called Quadratic Discriminant Analysis. Still, such formulation results in huge number of parameters as each covariance matrix is of size $p \times p$, which is even a problem in LDA. In order to tackle this issue regularization is introduced to form RDA and the covariance matrix is shrunk towards a diagonal one with λ as regularization parameter:

$$\hat{\Sigma}(\lambda) = \lambda \hat{\Sigma} + (1 - \lambda)\alpha I$$

In the above equation $\hat{\Sigma}$ is the sample covariance matrix and λ is the regularization constant. The value of λ is chosen using Ledoit-Wolf lemma ([28]) and α is the average eigenvalue of $\hat{\Sigma}$.

The other method is Nearest Shrunken Centroid (NSC) classifier, described in [43]. It is based on a very simple nearest centroid classifier: we calculate centroid for each class and then assign each sample to a class based on its distances to these centroid, i.e. to the one with the smallest euclidean distance. Such elementary algorithm doesn't work in settings where $n \ll p$, due to the curse of dimensionality as described in the beginning of this chapter. Therefore within class centroids coordinates are shrunken towards overall feature mean, making the assignment to classes dependent only on a small subset of variables. To formalize this intuition let us denote x_{ji} as the value of i-th variable in j-th sample where $i = 1, \ldots, p$ and $j = 1, \ldots, n$. We assume there are $K = 2$ classes and subset of sample indices for each class is C_k . Let $\bar{x}_i$ denote i-th feature mean and $\bar{x}_{ik}$ denote i-th feature mean

within class k. Then we can write:

$$\begin{aligned}
\bar{x}_{ik} &= \bar{x}_i + m_k s_i d_{ik}, \text{ where} \\
d_{ik} &= \frac{\bar{x}_{ik} - \bar{x}_i}{m_k \cdot s_i} \\
s_i^2 &= \frac{1}{n-K} \sum_{k=1}^{K} \sum_{j \in C_k} (x_{ij} - \bar{x}_{ik})^2 \\
m_k &= \sqrt{1/n_k - 1/n}
\end{aligned}$$

The values of m_k and s_i are chosen so that s_i^2 is a pooled variance and m_k assures that $m_k \cdot s_i$ is the estimated standard error of $\bar{x}_{ik} - \bar{x}_i$. With such formulation d_{ik} is a t-statistic for i-th feature comparing i-th variable mean with a mean within k-th class. The shrinkage is applied to d_{ik} creating shrunken centroids $\bar{x}'_{ik}$:

$$\begin{aligned}
\bar{x}'_{ik} &= \bar{x}_i + m_k s_i d'_{ik}, \text{ where} \\
d'_{ik} &= \operatorname{sign}(d_{ik}) \left(|d_{ik}| - \Delta\right)_+
\end{aligned}$$

This type of shrinkage is called soft thresholding and is just application of proximal operator used for solving problems with L_1 regularization. The advantage here is that with a proper choice of Δ parameter most of the coordinates of class centroid are equal to the coordinates of the whole data set centroid, leaving only few coordinates of each $\bar{x}'_{ik} - \bar{x}_i$ nonzero. Those few values will be the ones determining the assignment to classes. In terms of our data it translates to making prediction of metastasis dependent only on few chosen genes. It is very reasonable as in microarray analysis most genes does not differ significantly by expression levels between classes and the variability usually comes from random fluctuation. This method aims at removing the noisy features.

With a new sample $x^* = \left(x_1^*, x_2^*, \ldots, x_p^*\right)$ to classify we choose the class k with the lowest discriminant score, i.e:

$$\delta_k(x^*) = \sum_{i=1}^{p} \frac{\left(x_i^* - \overline{x}'_{ik}\right)^2}{s_i^2} - 2 \log \pi_k \tag{2.5}$$

The first part of this score corresponds to the normalized distance to shrunken centroid and the second one results from a simple prior distribution put on class assignment where $\pi_k = \frac{|C_k|}{n}$. By expanding the square in equation 2.5 we could see that discriminant score in NSC is very similar to discriminant score for LDA (as in equation 2.4). In fact, the distinction is that NSC assumes diagonal covariance matrix and uses shrunken centroids instead of sample means. The first one is particularly useful in high dimensional data as covariance matrix is of size $p \times p$ and may influence the stability of scores calculations and therefore usability of the model.

2.2.3 Linear Support Vector Classifier

Support Vector Classifier (SVC, [25, chapter 9]) is one of the best "out of the box" classifiers, known to be effective in high dimensional spaces. The technique is based on the concept of maximal margin classifier. Let us denote by $X_{n\times p}$ a data set consisting of n samples and p features and a set of labels $Y = (y_1, \ldots, y_n)$ and for simpler classifier rule formulation instead of $y_i \in \{0, 1\}$ we have $y_i \in \{-1, 1\}$. We assume our data is completely separable by a hyperplane and our aim is to find it. Such hyperplane can be described by equation:

$$\beta^T x + \beta_0 = 0 \text{ ,where } \beta = (\beta_1, \ldots, \beta_p)$$

Without loss of generality we can assume that samples with label 1 are the ones above the separating hyperplane. In such case the hyperplane has following property:

$$y_i(\beta^T x + \beta_0) > 0, \ \forall_{i=1,\ldots,n}$$

Such characteristic can be exploited to construct very simple classifier: assign new sample x^* to class 1 if $\beta^T x^* + \beta_0 > 0$ and to -1 if $\beta^T x^* + \beta_0 < 0$. Notice that the magnitude of $|\beta^T x^* + \beta_0|$ determines how far is the sample from our hyperplane and in consequence how confident we are about the assignment. Of course if the data is indeed completely separable there are infinitely many separating hyperplanes. However, we choose the maximum margin hyperplane, i.e the hyperplane that maximizes the distance of the closest point to the hyperplane and additionally our minimal confidence about sample assignment within the data. That distance to closest point is called the margin and the points that are exactly margin away from the hyperplane are called support vectors. The intuition for such naming is that even slight change of any of these points changes the hyperplane. Whereas, for all other samples the hyperplane is independent, i.e if they are perturbed, they do not affect the hyperplane unless they are within margin distance from it. The 2D example of few possible hyperplanes and the optimal one can be found on figure 2.4.

To formalize the problem of finding the maximum margin hyperplane we can write:

$$\begin{aligned} &\arg\max_{\beta_0,\beta} \ M \\ &\text{subject to } \|\beta\|_2^2 = 1, \\ &y_i\left(\beta_0 + \beta^T x_i\right) \geq M \ \forall_{i=1,\ldots,n} \end{aligned} \tag{2.6}$$

The inequality represents the condition that each point has to be at least by M far from boundary, the norm constraint is used for uniqueness as $\beta_0 + \beta^T x = 0$ represents the same subspace as $k \cdot (\beta_0 + \beta^T x) = 0$. Also it ensures that the perpendicular distance from x_i to hyperplane is equal to $y_i\left(\beta_0 + \beta^T x_i\right)$. So M is in

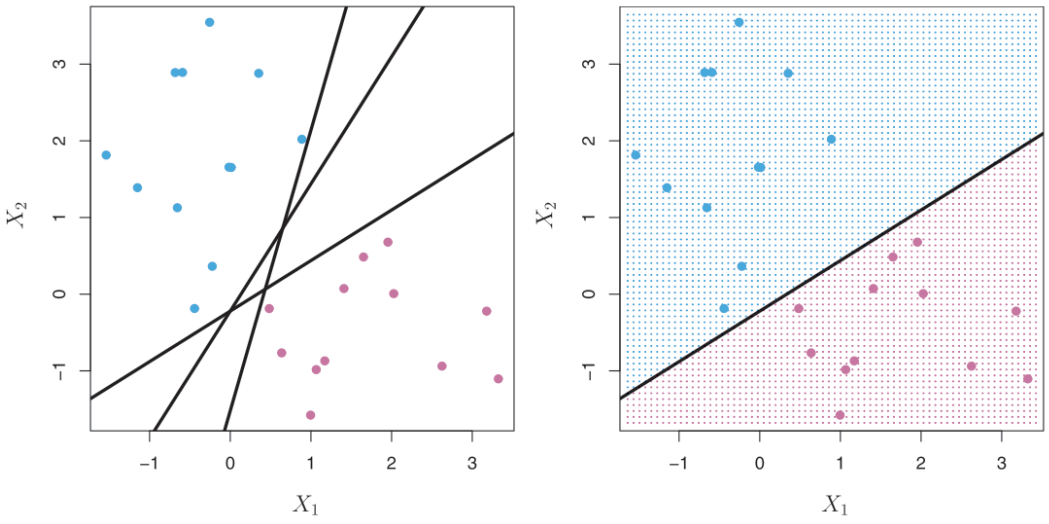

Figure 2.4: Example hyperplanes in 2D case. The picture comes from [25] - figure 9.2

fact the exact margin. This optimization problem can be rewritten to get rid of M. The inequality condition can be replaced by $y_i\left(\beta_0 + \beta^T x_i\right) \geq M\|\beta\|$ and thanks to the solution being invariant to scaling β, we can arbitrarily set $\|\beta\| = \frac{1}{M}$. In such case the problem can be formulated discarding M:

$$\begin{aligned} &\arg\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 \\ &\text{subject to } y_i\left(\beta_0 + \beta^T x_i\right) \geq 1 \ \forall_{i=1,\ldots,n} \end{aligned}$$

The above is a convex optimization problem and can be solved using Lagrange multipliers. However, in real world examples data is usually not linearly separable. SVC tackles this issue by finding a hyperplane that nearly separates all samples using *soft margin*. Another advantage is that even in case of separable data finding not necessarily perfect hyperplane allows the model to perform better on unseen data. At the figure 2.5 we can see that adding a single sample to our data set may change the hyperplane dramatically - possibly decreasing efficiency when classifying unseen samples due to very narrow margin.

Another issue is the decrease of confidence of our assignment for many samples near the margin. Therefore, we change our aim from perfect separation to greater confidence about most samples, ensuring robustness to individual points. We allow few data points to be on the wrong side of the hyperplane for the cost of penalty, which increases with the distance from the correct subspace. Using the formula it can be written as:

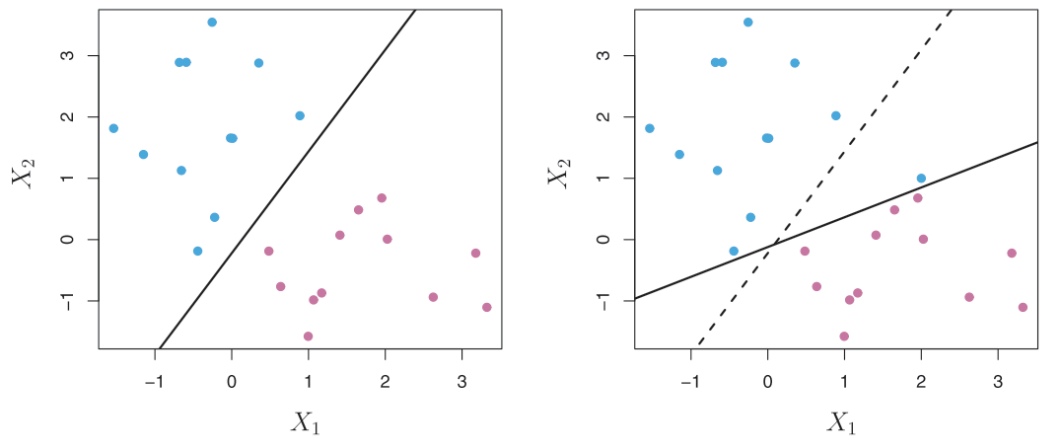

Figure 2.5: Change of maximum margin hyperplane when added one data point. The picture comes from [25] - figure 9.5

$$\begin{aligned} &\arg\max_{\beta_0,\beta} \; M \\ &\text{subject to } \|\beta\|_2^2 = 1, \\ &y_i\left(\beta_0 + \beta^T x_i\right) \geq M(1-\epsilon_i), \\ &\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i < C \; \forall_{i=1,\ldots,n} \end{aligned}$$

Again we can discard M by reformulating the problem to:

$$\begin{aligned} &\arg\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 \\ &\text{subject to } y_i\left(\beta_0 + \beta^T x_i\right) \geq 1-\epsilon_i, \\ &\epsilon_i \geq 0, \; \sum_{i=1}^{n} \epsilon_i < C \; \forall_{i=1,\ldots,n} \end{aligned}$$

For computational convenience the above can be presented in equivalent form:

$$\begin{aligned} &\arg\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{n} \epsilon_i \\ &\text{subject to } \epsilon_i \geq 0, \quad y_i\left(\beta_0 + \beta^T x_i\right) \geq 1-\epsilon_i, \; \forall_{i=1,\ldots,n} \end{aligned} \tag{2.7}$$

C is the hyperparameter determining how strong we penalize samples being on the wrong side of the margin. In case of $C = \infty$ SVC is equivalent to maximum margin classifier. On the other hand, small values of C result in allowing more points to be on the wrong side of the hyperplane. The ϵ_i are known as slack variables. The value of each ϵ_i determines the relative position to hyperplane. $\epsilon_i = 0$ indicates that x_i is

on the correct side of hyperplane, at least margin away from it. $0 < \epsilon < 1$ means that x_i is on the correct side but within margin distance from subspace. Finally, $\epsilon > 1$ means that x_i is on the wrong side. The notion of support vectors also gets extended to include points not only exactly on the margin but also the ones for which $\epsilon_i > 0$. The classification rule remains the same as for maximum margin classifier. Therefore, the decision on class assignment of new samples is based only on support vectors, i.e. potentially small subset of observations. This characteristic contrasts logistic regression and RDA as their decision rule always depend on all observations. However, some similarities to the first method can be shown. The conditions in equation 2.7 are equivalent to $\forall_{i=1,\dots,n}\epsilon_i \geq \max\big(0, 1 - y_i(\beta_0 + \beta^T x_i)\big)$. If we minimize the target that includes sum of ϵ_i then all inequalities will have to be equalities. Additionally denoting $\lambda := \frac{1}{C}$ we can write the equivalent form of our minimization problem:

$$\arg\min_{\beta_0,\beta} \sum_{i=1}^{n} \max\big(0, 1 - y_i(\beta_0 + \beta^T x_i)\big) + \frac{\lambda}{2}\|\beta\|^2$$

which is as in case of regularized logistic regression: a loss term + penalty term. This loss is called *hinge loss* and it behaves similarly to negative log likelihood loss in LR, as for the points far from decision boundary the latter gives very small values. It is also worth noticing that the choice of l_2 norm was motivated by interpretation of M in 2.6. Discarding M allows us to select norm almost arbitrarily, depending only on the desired properties of the final model. Therefore, we choose to use l_1 norm in order to enforce sparsity.

2.2.4 Ensemble learning

In this section we describe two classification techniques based on the concept of ensemble learning. Ensemble learning is a method of training multiple models and aggregating their results to produce a prediction for a sample. These models may be very simple and poor classifiers by themselves but a group of them may form a powerful tool.

Random Forest

In the Random Forest model, introduced in [8], the simple learners are decision trees. A single decision tree splits the feature space to a number of regions using feature value based rules and assigns a new sample to a class which is a majority in the resulting region for this data point. An example of classification decision tree can be seen on figure 2.6.

This tree has two internal nodes. The decision rule in the upper one, i.e root splits the data to two subtress based on *height* threshold, which is 180cm. The left node is not further split. In the right subtree we split the data again based

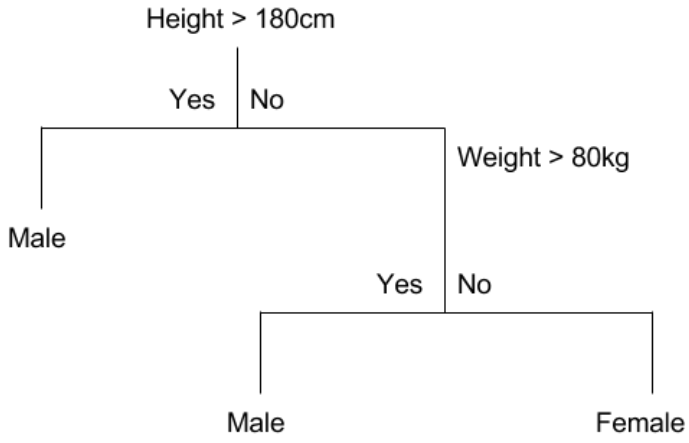

Figure 2.6: An example of a decision tree from `https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/`

on weight and two resulting nodes are also leaves. The label in the leaf indicates which class stands for the majority of training data in the corresponding region. For a new sample we traverse the tree from the root following the decisions in the nodes and assign it to a class that is a label of the leaf to which it falls. In general case, in the root of the tree we have one of our variables and some threshold - the data is then split according to the threshold to left part and right part of the tree. In each subtree either the process continues, a variable and a threshold are chosen and part of the data is split again or the splitting stops and the node remains the leaf of the tree representing one of regions in the feature space. As the rules are simple thresholds all the resulting segments are hyperrectangles. In order to learn such tree we have to have an algorithm for choosing a variable and a threshold in each node or deciding that the node is a leaf. Further in this section we consider only binary classification task. As usual we aim at minimization of misclassification error. It indicates that we would like the most common class in the leaf to be the vast majority of all the samples in this terminal node. Therefore, if we split the data to $R_1, \dots R_I$ regions we aim at minimizing $J_M = \sum_{i=1}^{I} 1 - \max(\hat{p}_i, 1 - \hat{p}_i)$, where $\hat{p}_i$ is the proportion of samples in the i-th region from the class 1. However, it turns out that such measure has poor computational properties and instead Gini index or entropy are used:

$$\text{Gini index: } J_G = \sum_{i=1}^{I} G_i = \sum_{i=1}^{I} 2\hat{p}_i(1 - \hat{p}_i)$$

$$\text{entropy: } J_E = \sum_{i=1}^{I} E_i = \sum_{i=1}^{I} -\hat{p}_i \log \hat{p}_i - (1 - \hat{p}_i) \log(1 - \hat{p}_i)$$

Both these criteria have small values if the node is pure - consists only few samples

of the other class and therefore are called *impurity* measures. What is more, if we assume a probabilistic interpretation of each node (we assign a sample with probability $\hat{p}_i$ to class 1 and with $1 - \hat{p}_i$ to class 0), then if we want to classify a randomly labeled sample according to class distribution in a leaf than the probability of misclassification is exactly $2\hat{p}_i(1 - \hat{p}_i)$. On the other hand, concept of entropy comes from the information theory and is natural measure of disorder.

Although the criterion to minimize has a very simple form it is in most cases computationally too complex to check all possible partitions. Therefore, a greedy approach is used, i.e recursive binary splitting. The process starts with a tree containing only root and all samples within it. The greediness comes from the fact that at root we choose a variable and threshold value which 'best' splits the samples to subtrees. Assume we use a Gini index as a measure of tree score and we split the root to $R_1(j, s) = \{x|x_{ij} < s\}, R_2(j, s) = \{x|x_{ij} \geq s\}$(where x_{ij} is the value of j-th feature for i-th sample. These means that we look for values of j, s minimizing $\frac{|R_1|}{|R_1|+|R_2|}G_{R_1} + \frac{|R_2|}{|R_1|+|R_2|}G_{R_2}$. We want to weigh each node by the number of samples so as to make difference between impurity of trivial split (half of each class in each node) and no split 0 and assert that larger nodes have bigger influence on our tree structure. In the resulting nodes the process is repeated until the stop condition is met. The choice of stop condition is important as without it we would split the nodes until each of them is completely pure. These may lead to the tree of large height and small number of data points in each leaf. In other words, the tree would overfit our data set and would have poor generalization performance. Popular choices of stop conditions are:

- minimal number of samples in the node in order to further split it
- maximal height of the tree
- minimal number of samples in the leaf (blocks splits creating node with less samples)
- minimal decrease in impurity - minimal difference between impurity for original node and sum of its children
- maximal number of leaves in the tree

The other approach is to instead of incorporating stop conditions, create a huge tree and cut off weakest splits. Such method is called tree pruning.

Although, decision trees have numerous advantages like being simple to visualize, understand and interpret or able to model complex relations between variables in the feature set, they also suffer from high variance and are prone to overfitting. We can look at the problem of sensitivity to data perturbation in other way. Imagine we split our data set randomly to two parts and fit a decision tree on each of them independently. Resulting decision trees will likely be much different, in contrast to

low variance models like linear/logistic regression. To tackle these issues Random Forest model is used that exploits the technique called bootstrap averaging. Bootstrap averaging (or bagging) exploits very simple property that the variance of the mean of n independent variables with variance σ^2 is equal to $\frac{1}{n}\sigma^2$. So in order to reduce variance of our model we can fit many models on different training sets and average the results or in case of classification assign a label which was chosen by a majority of the models. Of course, in most cases we don't have access to multiple data sets so we bootstrap new ones by taking random samples from the original one with replacement. Usage of a family of deep, not constrained trees was proven to greatly improve the accuracy and stability of decision trees. However, the resulting trees might not be independent or even be highly correlated, e.g. if there is a very informative feature that in all bootstrapped data sets is chosen to be used for splitting in the root. For a set of n i.d variables with pairwise positive correlation ρ the variance of average is $\rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$ and increasing number of variables will only reduce the second term. As a result Random Forest model introduces a feature subsampling scheme to reduce the correlation. In each tree, before each node is split only m predictors are considered instead of all variables. Usual value of m is $\sqrt{p}$. This means that most of the features are not even examined during a single split, resulting in decorrelated trees.

Random Logistic Regression

Although this feature subsampling scheme called feature bagging usually improves significantly nonlinear estimators, we also use it to create a model named in further sections as Random Logistic Regression (RLR). Instead of fitting simple logistic regression model we fit a number of LR using only a small subset of features for training. Then, in order to obtain the probability of class membership, we average probabilities from the estimators, hoping to obtain a classifier more robust than standard logistic regression.

2.3 Dimensionality reduction

In the previous section we saw a few classifiers applicable in the field of high dimensional data. For most of them the common part is usage of shrinkage that allow the classifier to be stable and prevent it from overfitting in the setting $n \ll p$. However, there is a family of other techniques called *dimensionality reduction* methods. They deal with the problem of high number of variables out of which only a fraction influences the response. Dimensionality reduction provides a new subspace consisting only of p' features, where $p' < p$. After reduction only these new predictors are used for building models and a number of standard classification techniques might be applied as the dimension can be reduced so that $n > p'$. Using dimensionality reduction before fitting a model usually improves its performance on unseen data due

to discarding of noisy features and reduces the computational complexity of training. Moreover, the stability of the model is enhanced as the number of coefficients to estimate is drastically decreased. Dimensionality reduction techniques can be split to two categories: feature extraction and feature selection. Feature extraction methods transform original variables, usually by taking their linear combinations. On the contrary, feature selection methods do not transform the original set of features, instead they select a small subset of initial predictors and in consequence keep their meaning resulting in better model interpretability. This is a great advantage, especially in the field of biomedical studies, but usually models based on selected variables perform worse than models based on the extracted ones. Therefore, in the following sections we present some methods from both groups.

2.3.1 Feature extraction - unsupervised methods

Principal Component Analysis

One of the most popular dimensionality reduction methods is Principal Component Analysis (PCA). It transforms linearly the set of p features to p' new uncorrelated variables so that the error of the projection is minimized causing the information lost during the reduction process to be as low as possible. It turns out that such formulation is equivalent to the task of maximization of variance of extracted predictors which can be interpreted as keeping the directions that allow us to distinguish between different samples. The new features are refereed to as principal components while the axes in corresponding vector space are called principal axes. To see the equivalence between variance maximization and projection error minimization let us look at the first principal axis. If we denote as X centered data matrix, $\Sigma = X^T X/(n-1)$ as sample covariance and w as the desired first principal axis then we want to maximize $Var(Xw) = w^T X^T Xw/(n-1) = w^T \Sigma w$. In order to make the problem well defined we also put a constraint on w: $\|w\|_2 = 1$. On the other hand, minimizing the error of projection can be written as $\min_w \|X - Xww^T\|_2^2$. By rewriting this we obtain:

$$\begin{aligned}
\left\|X - Xww^\top\right\|_2^2 &= \operatorname{tr}\left(\left(X - Xww^\top\right)\left(X - Xww^\top\right)^\top\right) \\
&= \operatorname{tr}\left(\left(X - Xww^\top\right)\left(X^\top - ww^\top X^\top\right)\right) \\
&= \operatorname{tr}\left(XX^\top\right) - 2\operatorname{tr}\left(Xww^\top X^\top\right) + \operatorname{tr}\left(Xww^\top ww^\top X^\top\right) \\
&= C - \operatorname{tr}\left(Xww^\top X^\top\right) = C - \operatorname{tr}\left(w^\top X^\top Xw\right) \\
&= C - (n-1)w^\top \Sigma w = C - (n-1)Var(Xw)
\end{aligned}$$

where C is a constant independent of w. In the above derivation we use the norm constraint on w and trace properties: linearity and a fact that $\operatorname{tr}\left(A^T B\right) = \operatorname{tr}\left(BA^T\right)$. So by looking at the last equation we can see that minimizing projection error is equivalent to maximizing variance.

The principal axes turn out to be eigenvectors of Σ. To see this let us note that Σ is symmetric and semi-definite and therefore can be diagonalized. As the value of $w^T\Sigma w$ does not depend on the basis (as changing bases is just a rotation) we can focus on our maximization problem in the basis of eigenvectors of Σ. Then $Var(Xw) = \sum_{i=1}^{p} \lambda_i w_i^2$ where $(\lambda_1, \ldots, \lambda_p)$ are the sorted decreasingly eigenvalues. Note that $\sum_{i=1}^{p} \lambda_i w_i^2 \leq \lambda_1 \sum_{i=1}^{p} w_i^2 = \lambda_1 \|w\|_2^2 = \lambda_1$ and it is clear that this value is reached for $w = (1, 0, \ldots, 0)$. To see that the following principal axes are the next eigenvectors notice that if v is the second principal axis then $v \perp w$ and $v \in Lin(b_2, \ldots, b_p)$ where $b_1, \ldots, b_p$ are eigenvectors corresponding to $\lambda_1, \ldots, \lambda_p$. Therefore $v_1 = 0$ (first coordinate in eigenvectors space) and we can use the same inequality argument as for the first axis and continue it for all the other principal axes. Of course for proper dimensionality reduction only first $k < p$ principal components are kept as reduced features. PCA is sensitive to scaling so it is advised to standardize data set before applying the method. Also centering is needed to make sure that the principal axes really correspond to maximal variance directions. It is also important to note that PCA is only able to capture linear dependencies between variables and might perform poorly if the correlation structure is more complex. The other issue is the choice of the number of features to keep. As for the standard rule of thumb we can choose k principal components so that they explain 95% of data variance. This means choosing k fulfilling:

$$k = \arg\min_{k'} \frac{\sum_{i=1}^{k'} \lambda_i}{\sum_{i=1}^{p} \lambda_i} \geq 0.95$$

because λ_i is equal to actual variance of i-th component. There also exist other popular methods as described in [26, chapter 6] or more novel based on Bayesian approach, especially for data sets, for which $n \ll p$, like Penalized Semiintegrated Likelihood (PESEL, [39]).

Sparse PCA

One significant drawback of PCA is that every principal component is a linear combination of all original features with usually all coefficients non-zero. It makes the new variables difficult to interpret, particularly in $n \ll p$ setting. To tackle this issue family of methods called sparse Principal Component Analysis (sPCA, [55]) uses a regression interpretation of PCA and imposes a penalty on loadings to regularize them. One of them *sparse PCA via regularized SVD*([38]) exploits Singular Value Decomposition to obtain sparsity. Using the same notation as for PCA, SVD can

be expressed as:

$$\begin{aligned} X &= UDV^T \\ r &:= rank(X) \\ U &= [u_1, \ldots, u_r] \\ V &= [v_1, \ldots, v_r] \\ D &= diag(d_1, \ldots, d_r), \quad d_1 \geq \ldots \geq d_r \geq 0. \end{aligned}$$

U, V are matrices satisfying $U^TU = I_r, V^TV = I_r$, whose columns are unit eigenvectors of XX^T, X^TX respectively and D is the diagonal matrix of singular values of X. Singular values of X are square roots of eigenvalues of X^TX. Using the relation between V and X^TX, we have that $XV = UD$ are the principal components. Moreover columns of V represent *loadings* - weights of each variable in linear combination representing PCs in terms of principal axes. sPCA aims at regularizing these loadings. To do this it exploits a property of SVD that it produces best k-rank matrix approximation:

$$\underset{rank(X_k)=k}{\arg\min} \|X - X_k\|_F^2 = \sum_{i=1}^{k} d_i u_i v_i^T$$

where $\|\cdot\|_F$ denotes Frobenius norm. This also means that $d_1u_1v_1^T$ is the best rank one approximation of X, $d_2u_2v_2^T$ is the best rank one approximation of residual matrix $X - d_1u_1v_1^T$ and so on. This property will be used to create an iterative procedure producing sparse loadings. In order to find them we solve the minimization problem:

$$\min_{u,v,\|u\|_2=1} \|X - uv^T\|_F^2 + \lambda\|v\|_1$$

Without the penalty term the solution is $u = u_1, v = d_1v_1$. The algorithm 1 is presented in [38] for solving this problem.

Algorithm 1 sPCA via regularized SVD

Compute SVD for X and set $v := d_1v_1,\ u := u_1$

Until convergence repeat:

1. $v = sign(X^Tu)(|X^Tu| - \lambda)_+$
2. $u = \frac{Xv}{\|Xv\|}$

After convergence standardize v: $v = \frac{v}{\|v\|}$

The soft thresholding in the loop of the algorithm is no surprise as it is an application of proximal operator used for solving problems with L_1 regularization for which gradient descent is not applicable. In order to calculate further loadings we apply the algorithm to the next residual matrices. The sparsity of the solution depends on the value of tuning parameter λ.

Multiple Latent Components Clustering

Multiple Latent Components Clustering (MLCC) deals with PCA disadvantages differently. The assumption used in PCA that the samples come from the same subspace of small dimension may be unjustified. A better one is to expect that the data comes from a union of low dimensional subspaces. The method tries to find these subspaces and for a fixed number of clusters (subspaces) performs an extension of k-means algorithm on features where each cluster centre is represented by a subset of principal components and the similarity between a factor and a subspace is calculated using Bayesian Information Criterion (BIC). The dimensionality of each subspace is determined using mentioned before PESEL. Additionally the number of clusters may be estimated automatically using a modified version of BIC (mBIC) for model selection. The method was purposely designed to handle data sets with low signal to noise ratio. Such case is very common in the gene expression data due to obscured variation and high variability of gene expression levels for genes not related with examined condition.

In more details the method assumes that our data $X_{n\times p} = [x_1, \dots x_p]$ comes from K subspaces. Each of the subspaces is represented by its dimensionality k_i, set of factors that generate the subspace F_i and each variable in a cluster is normally distributed. To formalize this we can write:

$$x_{ij}|F_i, c_{ij}, \sigma_i^2, \mu_i \sim N\left(\mu_i + F_i c_{ij}, \sigma_i^2 I_n\right)$$

where x_{ij} is the j-th variable in i-th cluster, c_{ij} is a coefficient in linear combination representing the feature in terms of factors and σ_i^2 is a magnitude of noise within i-th cluster. The entire algorithm is presented in 2.

Algorithm 2 Multiple Latent Components Clustering

Require: N - number of runs of the algorithm, $iter_{max}$ - maximal number of iterations

Standardize data matrix X

for $i \in \{1, \ldots, N\}$ **do**

1. Initialize clusters' centers
2. Until convergence or $iter_{max}$ times
   (a) For $j = 1, \ldots, p,\ j' = 1, \ldots, K$ compute $BIC_{jj'}$ which is the value of BIC for linear regression model $lm(x_j \sim F_{j'})$
   (b) For $j = 1, \ldots, p$ assign x_j to q-th cluster where $q = \arg\max_{j' \in \{1,\ldots,K\}} BIC_{jj'}$
   (c) For every cluster use PESEL to estimate its dimensionality k_i. Use PCA to compute the first k_i principal components and store them in F_i
3. Store mBIC for computed model

end for

Choose the model with the highest value of mBIC and use F_i for $i = 1, \ldots, K$ as new set of features

The initial choice of cluster centers is random - the algorithm selects K variables from the data as the initial factors. Similarly to k-means the MLCC only finds local maxima of mBIC, therefore many different initialization have to be tried to increase the chance of finding the global one. To estimate the number of clusters one can run MLCC for different values of K and choose the model with the highest value of mBIC as it penalizes dimensionality of the model. More details on this technique can be found in [51].

2.3.2 Feature extraction - supervised methods

The common disadvantage of the methods from the previous section is that they are unsupervised, i.e do not use in any way labels associated with samples in our data. In consequence there is no guarantee that the new set of features will be right for the classification task. This is even more evident for microarray data as many gene expression profiles can have high variability and be responsible for biological process completely separate from our concern. In this section we present two methods that perform dimensionality reduction in a supervised manner - taking into account the particular prediction task.

Partial Least Squares

Partial Least Squares (PLS) was primarily designed for regression problems and improving the properties of OLS estimator, allowing it to be identifiable in $n \ll p$ setting. Nonetheless, PLS was used with success in the analysis of microarray data for tumor classification ([33]). The aim of PLS can be nicely compared with the aim of PCA. Whereas, PCA looks for orthogonal directions that maximize variance, PLS finds orthogonal directions that maximize covariance of new features with the response. It can be formulated as:

$$\begin{aligned}&\text{For } k = 1, \ldots, p' :\\ &w_k = \underset{\|w_k\|_2=1}{\arg\max}\, Cov\left(Xw, y\right)\\ &\text{subject to } Xw_k \perp Xw_{k'} \text{ for } k' \neq k\end{aligned}$$

where p' is the number of features in the reduced space. One of the algorithms for finding such w_k is *non iterative partial least squares* and can be found in [22, chapter 3.5.2].

Forest Deep Neural Network

The other supervised dimensionality reduction method, that performs classification as well, namely Forest Deep Neural Network (FDNN, [27]) is much more novel. It aims at exploiting neural networks, one of the most powerful, state-of-the-art family of models applicable in the field of classification, in the analysis of high dimensional data. Neural networks (NN) consist of *neurons*. Each neuron is a computational unit - it takes linear combination of the input and applies non-linear transformation to produce one number - the output. Mathematically it can be written as

$$f(x) = \sigma\left(\sum_{i=1}^{p} w_i x_i + b\right)$$

where x in the input vector, w, b are neuron parameters, weights and bias respectively and σ is an *activation function*. Note that if σ is sigmoid function then a neural net consisting of one neuron is a logistic regression model. In general there are many neurons in a network grouped into *layers*. One layer can be expressed in a compact way using the equation:

$$f(x) = \sigma\left(Wx + b\right)$$

where W is a matrix, each row representing weights of one neuron, b is a vector of biases (one number for each neuron) and σ is applied elementwise. In fully connected NNs the output from one layer is just the input to the next one. The first layer (raw data passed to neural net) is the input layer, the last one is called output layer (usually uses different activation function than others) and all layers between are *hidden layers*. Of course, weights and biases in every layer have to

be learned. The criterion optimized by neural network training algorithm is loss function, e.g negative log likelihood as in logistic regression. The neural networks are trained using extensions of gradient descent with respect to loss function. As for logistic regression we can impute a prior distribution on weights of each neuron in order to regularize the model. For the purpose of our analysis we use a NN with two hidden layers, another regularization method, i.e dropout between hidden layers. Moreover we choose to optimize the negative log likelihood loss function using ADAM optimizer and the activation functions are ReLU and softmax in the hidden layers and output layer respectively. These are standard choices for neural network architecture and more details on the topic can be found in [19].

Neural networks usually do not work in the $n \ll p$ setting. In order to capture the complicated correlations within the data one have to use a NN with many neurons and layers. In order to learn that many parameters using gradient descent algorithm large number of samples (significantly larger than the number of features) has to be used for training. In order to tackle this issue a new representation of variables is created using Random Forest technique. The method fits Random Forest classifier consisting of p' decision trees $T_1, \ldots, T_{p'}$. Let $T_i(x_j)$ be the label assigned to j-th sample by i-th tree. Then the new representation of j-th sample is $x'_j = \big(T_1(x_j), \ldots, T_{p'}(x_j)\big)$ and the reduced data matrix is X'. The flow of data through the network can be expressed as:

$$
\begin{aligned}
P(y|x') &= softmax\left(W_{out} z_1 + b_{out}\right) \\
z_1 &= f_a\left(W_1 z_0 + b_1\right) \\
z_0 &= f_a\left(W_0 x' + b_0\right)
\end{aligned}
$$

where W_0, W_1, b_0, b_1 are weights and biases of hidden layers, W_{out}, b_{out} weight and bias of output layer and f_a is an activation function. Graphically the whole model is presented in figure 2.7. In our implementation T_i can be either a decision tree (Random Forest is used as representation builder) or Logistic Regression using subset of variables (Random Logistic Regression model is used). Also we create representation using probabilities returned by the each of the small models instead of one hot encoding.

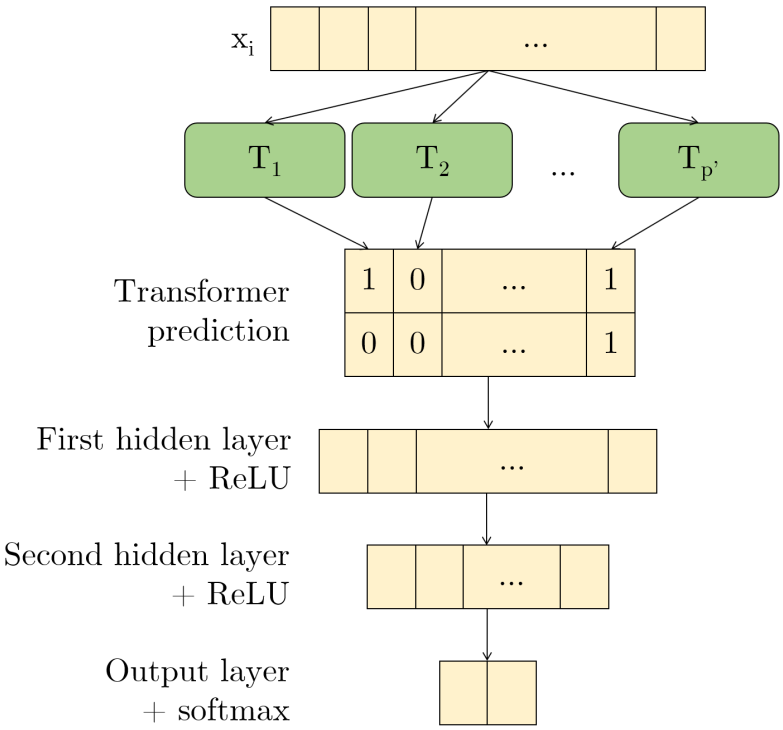

Figure 2.7: Graphical representation of the Forest Deep Neural Network

2.3.3 Feature selection

Significance analysis of microarray

Designed particularly for selection in the analysis of gene expression data, Significance analysis of microarray (SAM) was introduced in [45], [53] and implemented in R package *samr* ([42]). The motivation for the method is to test each gene statistically for significance. However, in the field of multiple hypotheses testing one cannot use standard level of significance, because if we set that each test with p-value lower than 0.01 yields significant gene for 12000 features we would expect 120 genes identified by chance, leading to high false discovery rate (FRD) - proportion of unrelated genes recognized as significant to all genes labeled as important. Relating this with performance metrics FDR is $1 - precision$. Therefore, SAM uses a modified version of t test and estimates FDR using random permutations of labels. In more details let us denote as $X_{n \times p}$ our data, where x_{ji} is the level of expression of i-th gene in j-th sample and $y = (y_1, \ldots, y_n)$ are the labels corresponding to each sample. For

each gene we compute a statistic:

$$
\begin{aligned}
d_i &= \frac{r_i}{s_i + s_0} \text{ ,where} \\
r_i &= \bar{x}_{i1} - \bar{x}_{i0} \\
C_k &= \{j : y_j = k\},\ n_k = |C_k|,\ \bar{x}_{ik} = \frac{\sum_{j \in C_k} x_{ji}}{n_k} \text{ , for } k = 0, 1 \\
s_i^2 &= \frac{(1/n_1 + 1/n_2)\left(\sum_{j \in C_0} (x_{ji} - \bar{x}_{i0})^2 + \sum_{j \in C_1} (x_{ji} - \bar{x}_{i1})^2\right)}{n_1 + n_2 - 2} \\
s_0 &\text{ - exchangeability factor}
\end{aligned}
$$

Here d_i is a score for each gene, C_0, C_1 are sets of samples labelled as $0, 1$ respectively, $\bar{x}_{i0}, \bar{x}_{i1}$ represent means of level of expression of i-th gene in each group and s_i is standard deviation for r_i. By taking a closer look one may realize that formula for d_i is very similar to a formula for d_{ik} in NSC classifier. The difference is that this time we do not compare mean of each class with the overall mean but we rather compare class means with each other. Removal of s_0 would reduce the procedure to a standard t-test, however this exchangeability factor is introduced to tackle the issue of high variance of r_i in case of small values of expression.

The next step is selection of important genes according to d_i scores and estimation of FDR. We compute order statistics $d_{(1)} \leq d_{(2)} \cdots \leq d_{(p)}$. Then we take B permutations of y and for each permutation we compute corresponding scores $d^{*b}_{(i)}$, order statistics $d^{*b}_{(1)} \leq d^{*b}_{(2)} \cdots \leq d^{*b}_{(p)}$ and expected order statistics $\bar{d}_{(i)} = \frac{\sum_{i=1}^{B} d^{*b}_{(i)}}{B}$. After that we fix a threshold Δ and all genes for which $|d_{(i)} - \bar{d}_{(i)}| > \Delta$ are deemed significant. For genes that have no impact on the response $|d_{(i)} - \bar{d}_{(i)}|$ should be small and therefore should be discovered using the permutation procedure. To estimate FDR let us denote by $cut_{up}(\Delta)$ the smallest score among genes for which $d_{(i)} - \bar{d}_{(i)} > \Delta$ and $cut_{low}(\Delta)$ the highest score among genes for which $d_{(i)} - \bar{d}_{(i)} < -\Delta$. For permutation b the number of falsely discovered genes is the number of genes for which $d^{*b}_{(i)}$ is either above $cut_{up}(\Delta)$ or below $cut_{low}(\Delta)$. Finally we multiply quantiles of obtained FDR by a constant $\hat{\pi}_0$ to get final estimation of FDR and its quantiles. For more details (e.g calculation of $s_0, \hat{\pi}_0$) refer to [45].

Recursive Feature Elimination

The other method uses SVC and its computed weights to determine which features are significant using Recursive Feature Elimination (RFE), applied with success in the field of cancer classification ([20]). The algorithm is quite simple - we fit the model with all features, discard the feature with the lowest weight, fit the model again with one variable less, discard the one with lowest weight and repeat the process until 1 predictor is left. This procedure allows us to obtain a ranking of genes. Removing one variable at a time is very important - the initial weights of full

model if used for ranking creation would result with sub-optimal order of features as described in the paper.

Chapter 3

Exploratory analysis

3.1 Basic statistics, visualizations and normalization

In this section we present the results of exploratory analysis. Although preprocessing has already been applied, such analysis should be performed to get a deeper understanding of the data. It involves analyzing main statistical characteristics, study of missing values and outliers, visualization, feature removal and normalization. Due to earlier preprocessing (quantile normalization and RMA introduced in 1.2), we expect our variables to follow normal distribution. To assess accuracy of this statements we plot values of four common statistics: mean, standard deviation, skewnees and kurtosis. Skewness measures the asymmetry of the distribution and is equal to the third standardized moment, whereas kurtosis corresponds to the heaviness of the tails of the distribution and is equal to fourth standardized moment. High kurtosis indicates heavy tails or outliers when calculated from sample. For normal distribution skewness and kurtosis are 0 and 3 respectively. Below are the mathematical formulas for these statistics:

$$Skew[X] = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$$

$$Kurt[X] = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$$

In the code we use *scipy.stats.kurtosis* to calculate the kurtosis which subtracts 3 from the result to make 0 the value for normal distribution.

In figure 3.1 we can see that the data is not standardized. The variables are sorted decreasingly by means varying from 6.01 to 13.79. Also standard deviations vary from 0.07 to 2.41. Therefore, before applying any dimensionality reduction method we should standardize our data. For example, in PCA the variance explained by an extracted feature hugely depends on its scale and lack of standardization could lead to over or underestimation of its effect. Skewness and kurtosis evidence against our normality assumption. They vary from -2.99 to 13.10 and -1.38 to 282.88

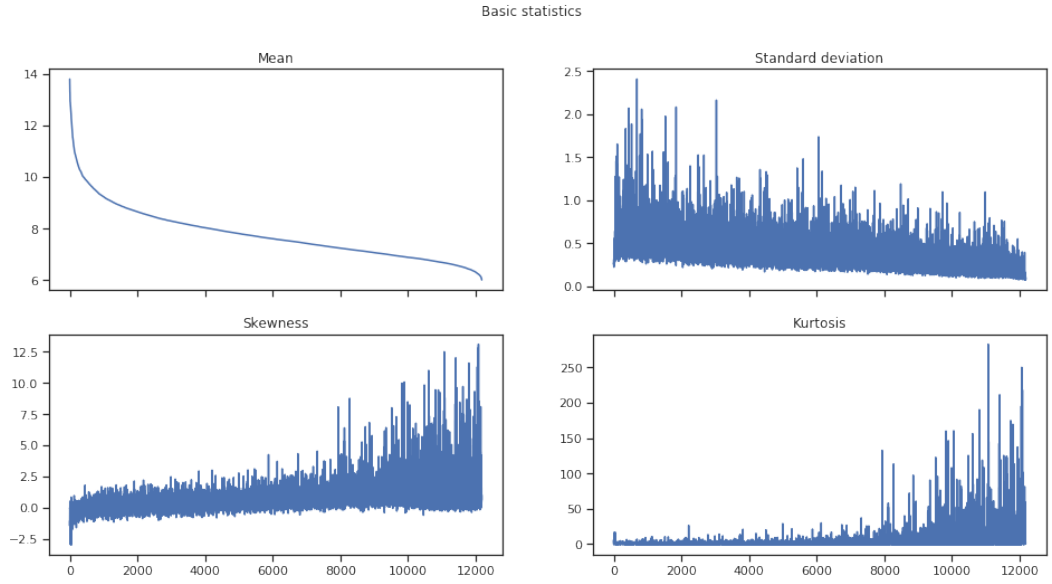

Figure 3.1: Main statistical characteristics of the data

respectively. Especially, the kurtosis implies presence of outliers. In order to get more insight on the distribution of our variables in figure 3.2 we plot the histogram and boxplots of genes with maximum skewness and kurtosis and in figure 3.3 we plot histograms and boxplots of randomly chosen genes from the data.

By looking at the figure 3.3 we conclude that although the distributions are visibly skewed and even multimodal, they do not indicate that further transformation is needed to able to use models assuming normality. However, we can clearly see from figure 3.2 that high skewness or kurtosis can be caused by just a few outlying values. In order to verify this for other variables we calculate the 95% quantile for both skewness and kurtosis. They are equal to 2.15 and 9.77 respectively. It indicates that only very few genes have statistics indicating strong non normality. What is more, the number of genes with kurtosis or skewness above 95% quantile is 710 which is not significantly larger than the number of genes with high kurtosis only. This signifies that the extreme values of either of these statistics appear on the small subset of genes. In order to assess the influence of such genes we consider following data sets: data set consisting of genes only with values of skewness and kurtosis below the 95% quantile (D_1), the complement of the previous data set (D_2), 50 data sets extracted by randomly choosing 710 variables from the original data (D_{random}). We split each of them to train and test parts and fit logistic regression with L1 penalty and regularization hyperparameter chosen using cross validation as described in section 2.1. We compare the scores on the test set to check how these genes with outlying values impact our prediction. For the random data sets we fit the model for each of them and take the mean scores. The results are presented in the table 3.1.

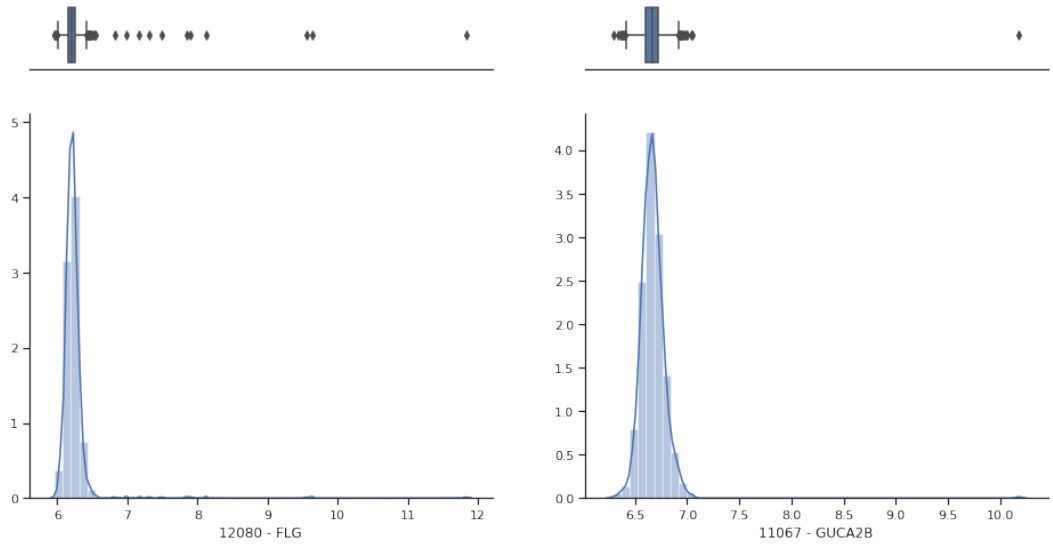

Figure 3.2: Histograms of genes with maximum skewness and kurtosis

	ROC AUC	Precision	Recall	F1-score
D_1	0.827	0.718	0.712	0.715
D_2	0.742	0.659	0.577	0.615
D_{random}	0.787	0.700	0.616	0.655

Table 3.1: Scores of logistic regression with subsets of variables

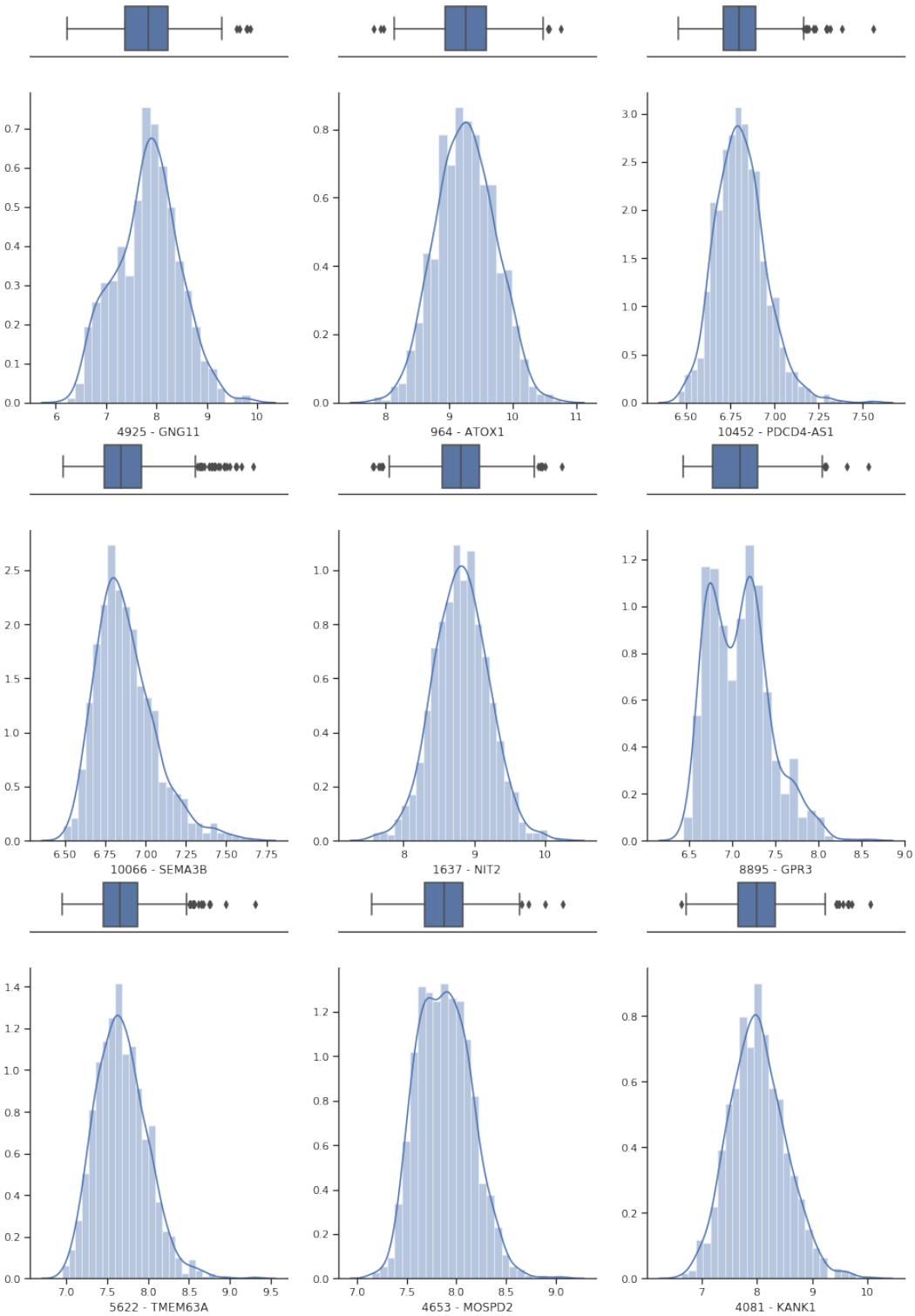

Figure 3.3: Histograms of randomly chosen genes from the data

From this table we can clearly see that the genes in D_2 do not have big predictive power. The mean scores from D_{random} are higher for every considered metric. It is also worth noting that randomly chosen variables bring less information for prediction than all the genes in D_1. The next step is checking how these genes with extreme values impact stability of our classifier.

In order to check it we use six different transformations on the data, then fit logistic regression model with L1 penalty and the same regularization parameter. We choose this classifier because its weights are sensitive to outliers and could easily indicate the impact of genes with extreme values. The compared transformations are:

1. No transformation
2. Standardization - from each variable we subtract its mean and divide by standard deviation
3. Quantile transformation to normal distribution - for each variable we calculate the empirical cumulative density function (CDF) and use it to project original data. The projected data is truncated to 10^{-7} and $1 - 10^{-7}$ then transformed using inverse CDF of normal distribution. The formula is $y_i = \Phi^{-1}(F(x_i))$, where F and Φ represent empirical CDF and normal distribution CDF respectively. This transformation is more robust to outliers than scaling but may deform distances within and across features.
4. Quantile transformation to uniform distribution - similar to the previous one but we transform data to match uniform distribution. We use it to check how the weights of logistic regression will be affected when an unreasonable transformation is applied to our data.
5. Truncating most extreme values - for each feature we calculate 1% (q_1) and 99% (q_{99}) quantiles and replace values smaller then q_1 with q_1 and values greater then q_{99} with q_{99}
6. Quartile standardization - for each variable we subtract its median and divide by inter quartile range which is equal to difference between 75% quantile and 25% quantile

The plots of the weights of these classifiers are presented on figure 3.4. We can see that the difference between no transformation and truncating extreme values (4-th plot in the first column) is very small. It indicates that these genes with very high skewness and kurtosis do not impact our prediction significantly and do not need to be truncated or removed from the data. What is more application of different transformation aiming to normalize our features (plots $4-6$-th in the second column) resulted in similar logistic regression coefficients. Taking into account figure 3.3 and 3.4 we conclude that there is no justification of using other transformation of our

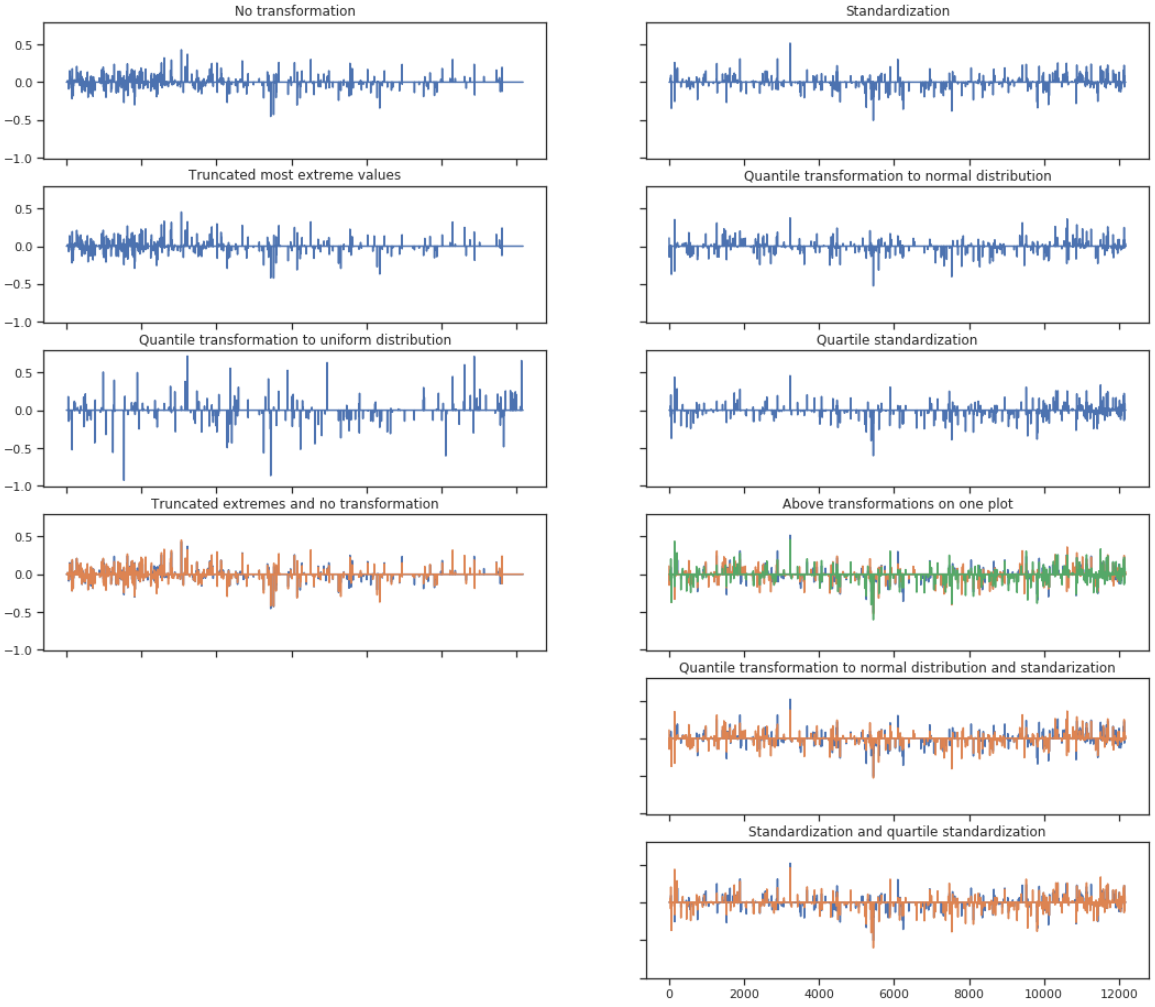

Figure 3.4: Weights of logistic regression with different data transformations

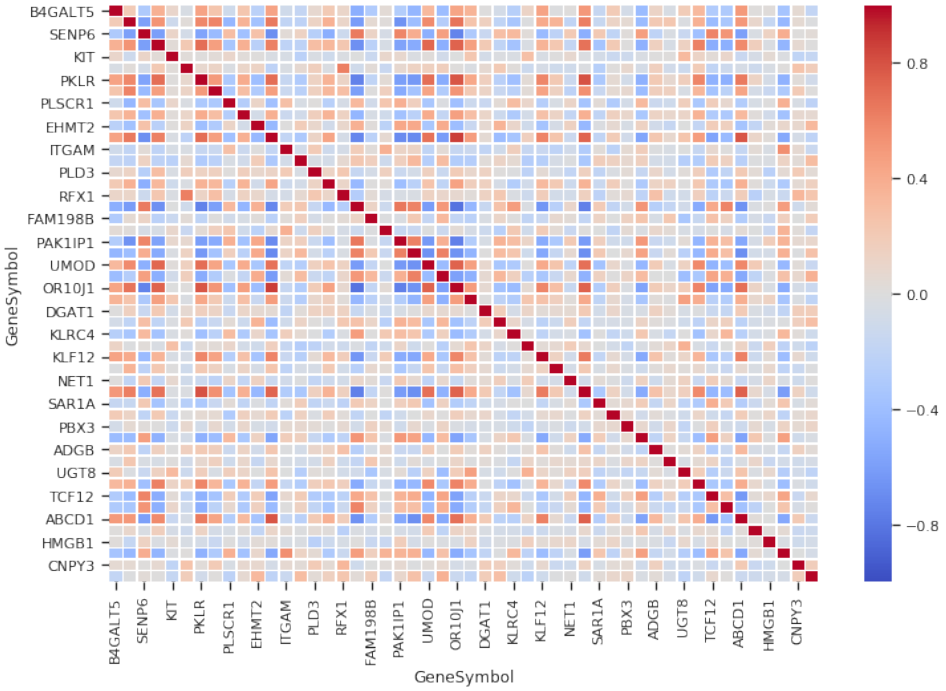

Figure 3.5: Correlation of randomly chosen subset of 50 genes

data set before further analysis. We will only standardize the data before applying dimensionality reduction methods that are sensitive to features' scales.

3.2 Correlation study

In data mining we usually aim at removing strongly correlated variables, due to the possibility of causing instability in our models ([44]). In case of gene expression data this usually refers to genes which are co-regulated and involved in the same genetic pathway. In order to get notion of such correlation structure we present (figure 3.5) the correlation between randomly chosen 50 genes.

Although most variables seem to have correlation close to 0 we can clearly see some significant ones, even though we chosen randomly only 0.41% of our variables. We may also look at the histogram of correlations of features with response (figure 3.6). We can see that majority of variables has low correlation coefficient indicating that the dependencies between the response and other variables are hidden and rather complex. These two figures strongly indicate that dimensionality reduction methods (both feature selection and extraction) can be exploited on our data.

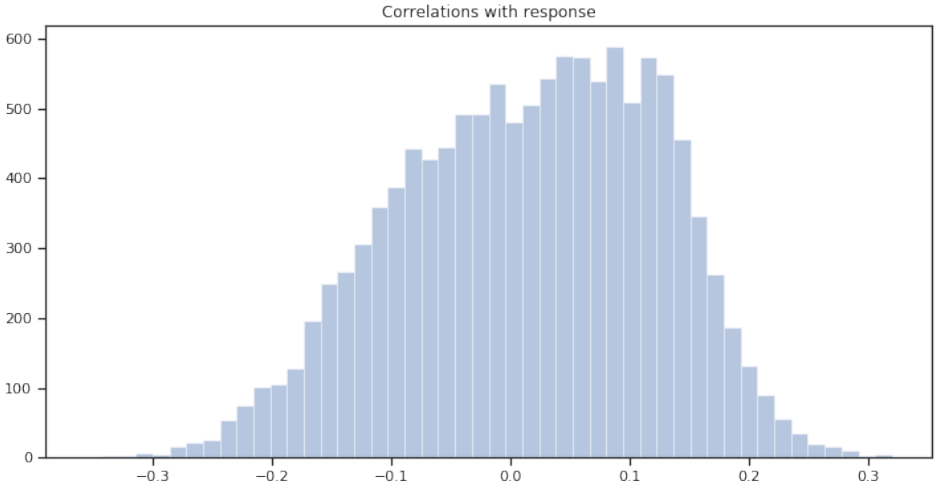

Figure 3.6: Histogram of correlations of features with response

Chapter 4

Implementation details and results

4.1 Implementation

As mentioned in the section 1 the implementation of all experiments for this thesis can be found at `github.com/sjwilczynski/geneExpr`. We chose to use *Python* programming language and *Scikit-learn* library ([35], [10]) as they together provide a wide variety of components and features for implementing machine learning pipelines. Scikit-learn has a few very appealing advantages:

1. Open-source - code of the library is available at `github.com/scikit-learn/scikit-learn`
2. Unified learning and inspection API for different models
3. Ready to use implementation of most methods mentioned in section 2
4. Possibility of composing a sequence of steps into one classifier

The first point is very important as in case of any problems or ambiguities on how some function or model works one can always check the code and adjust it to one's requirements. The second convenience, the unified API is particularly useful. All models expose a *fit* method for learning from given data set. Additionally, all dimensionality reduction or normalization classes expose a *transform* method which converts the initial set of features to the new one. Finally, all classifiers expose a *predict* method for assigning a new sample to one of the classes and most of them also implement a *predict_proba* which instead of hard assignments provides class membership probabilities. Apart from that, fitted models are easy to inspect as all parameters are available through attributes. Scikit-learn also provides hyperparameter tuning tools which together with unified API make model selection task almost effortless.

Moreover, we only needed to implement from scratch Random Logistic Regression due to the variety of models available in the library. As for the other adaptations, we use *skorch* package to allow us to exploit powerful neural network library, i.e *PyTorch*([34]) as a scikit-learn class, *easyspc* python package as a base for SPCA via regularized SVD implementation and R package *varclust* ([40]) together with python package *rpy2* to use MLCC from Python. Last but not least, scikit-learn exposes a class *Pipeline* that can be utilized to combine few steps of model fitting, i.e standardization with dimensionality reduction and finally classification. It is especially critical in terms of model and algorithm selection. If we use either normalization or dimensionality reduction both these steps should be performed within cross-validation loop. In the opposite case the information from hold-out fold would leak into training process in the form of features that were extracted using also the hold-out part of the data. It may lead to incredibly biased estimates returned from cross-validation as stated in [22, chapter 7.10.2] and [2]. Furthermore, hyperparameters for the classifier should be selected together with hyperparamters for corresponding dimensionality reduction method to enhance the performance. Both these issues are resolved by incorporating *Pipeline* class.

There is one more concern for implementation of hyperparameter tuning - how to search the space of hyperparameter values. There are three main approaches. Usage of manual search can be justified only if the researcher has a prior knowledge on which values are probable to work well. In our case as we are considering lots of different learning algorithms this technique is not suitable. The next one, grid search, is probably the most common method of searching hyperparameter space. For a given model one declares some hyperparameters to optimize, for each of them providing a number of values that should be checked. Then one compares models fitted using every possible combination of the hyperparameters. This technique is very simple, intuitive and reliable when searching through low dimensional space of hyperparameters. However, assume that we want to optimize five different hyperparameters and each of them could be assigned one of 10 values. This leads to testing 10^5 distinct models which can be computationally unfeasible. Such constraint would lead to reduction of the number of checked values, potentially risking omitting the most valuable ones. We can also notice that if a part of hyperparameters does not influence the model's performance significantly, many of the fitted classifiers will be tested unnecessarily, effectively wasting our time and computational resources. In order to tackle these issues a third way is often used, i.e random search introduced in [6]. Like for grid search we may specify values to test for each parameter or the distribution from which to sample a value. The first case is treated as uniform discrete distribution. We also declare a number of iterations of random search and in each iteration for each parameter a value is sampled from the corresponding distribution. It turns out that given the same restricted computational time for grid search and random search, the latter yields considerably better results. This is caused by the much higher effectiveness of searching through the parameter space

as now unimportant hyperparameters does not cause exponential increase in computation time. Moreover, experiments show that in most cases 60 iterations were enough to obtain values of hyperparameters very close to optimal ones. Taking the above into consideration and the fact that we compare a variety of different models and hyperparameter optimization is performed within the inner loop of nested cross-validation, we use random search method (implemented in scikit-learn) and 100 iterations. Below we list all tuned parameters for each model:

- LR: L_1 regularization constant
- Elastic Net: ratio between L_1 regularization constant and L_2 regularization constant
- RDA: no tunable parameters
- NSC: threshold Δ
- linear SVM: regularization constant λ
- Random forest: number of decision trees, maximal depth of each tree, minimal number of samples in a leaf, splitting criterion (Gini index or entropy)
- Random Logistic Regression: L_1 regularization constant, number of LR models used, number of features randomly chosen for each regression
- PCA: number of dimensions to keep
- SPCA: number of dimensions to keep, L_1 regularization constant, number of iterations of regularizing loop
- MLCC: number of clusters, maximal dimension of each subspace (upper bound for PESEL)
- PLS: number of dimensions to keep
- FDNN: learning rate, dropout rate, sizes of hidden layers, weight decay magnitude, number of simple classifiers for transformer (Random Forest or Random Logistic Regression)

By default all dimensionality reduction methods are followed by Logistic Regression classifier.

To mention other libraries extensively used in the thesis, for visualizations we utilize *matplotlib* ([23]) and its extension *seaborn*. Moreover, as some of the performed computations can be quite resource demanding we use machines at the Department of Mathematics and Computer Science of the University of Wrocław. In order to do that effectively we use *JupyterLab* - a powerful IDE for Python notebooks allowing nested visualizations and sequential execution of small chunks of code in interactive sessions on remote computer using ssh tunnels and a browser.

As many Python packages differ with APIs quite significantly from version to version we created a dedicated *Anaconda* environment and keep its configuration in the file *environment.yaml* in the Github repository. This allows to reproduce the configuration and in consequence results from this thesis on other machines. Finally, as for example running nested cross-validation for all models could take up to few days we utilize *papermill* package that allows remote execution of Python notebooks with proper parametrization (e.g number of iteration in random search) that could change for different runs without any alternation of the code. All these libraries and frameworks help us perform experiments, visualization, test new models and ideas effectively, as well as keeping the code clean readable and all the results reproducible.

4.2 Classification performance

4.2.1 Regularized classifiers and feature extraction

In table 4.1 we present the result of nested cross-validation for all models. The values in the table are means of the metrics from the outer loop, together with their standard deviations. We can see that the performance of most models is quite similar. However, PLS and RLR seem to be the best models having highest mean ROC AUC score with modest standard deviation, signifying good properties of bagging even for linear models. On the other hand, methods that performed poorly are NSC, LR and SVC. It may indicate that these models are not suitable for our problem, for example independence assumption in NSC is too much of a simplification. Nonetheless, in the section 3 we observed that LR could perform nicely. This may be caused by the too small amount of samples available for training, resulting in pessimistic bias as mentioned in 2.1. Therefore, for final tuning and assessment we choose PLS with LR as classifier, RLR and SVC. We use standard cross-validation procedure on training set to select good hyperparameters for the chosen models. The procedure returned 150 as the number of dimensions to keep for PLS and $C = 0.945$ as value of inverse of regularization constant for L_1 regularized LR classifier following PLS. For SVC, $C = 0.0341$ as value of inverse of regularization constant was picked resulting in expected strong regularization for method without any dimensionality reduction. For RLR we got $n\_variables = 250$ as the number of features to be sampled for each model, $n\_estimators = 100$ as the number of used LR classifiers and $C = 3.4$ as value of inverse of regularization constant.

The ROC curves on the test set for the chosen models are displayed in figure 4.1. The ROC AUC scores on test set are 0.818, 0.831, 0.815 for PLS, SVC and RLR respectively indicating that the pessimistic bias resulting from limited number of samples in the inner loop of nested cross-validation was quite significant. On the fourth plot the three curves are presented together. It shows that the performance of these methods is very similar. For further diagnosis in figure 4.2 we demonstrate precision and recall vs decision thresholds plot.

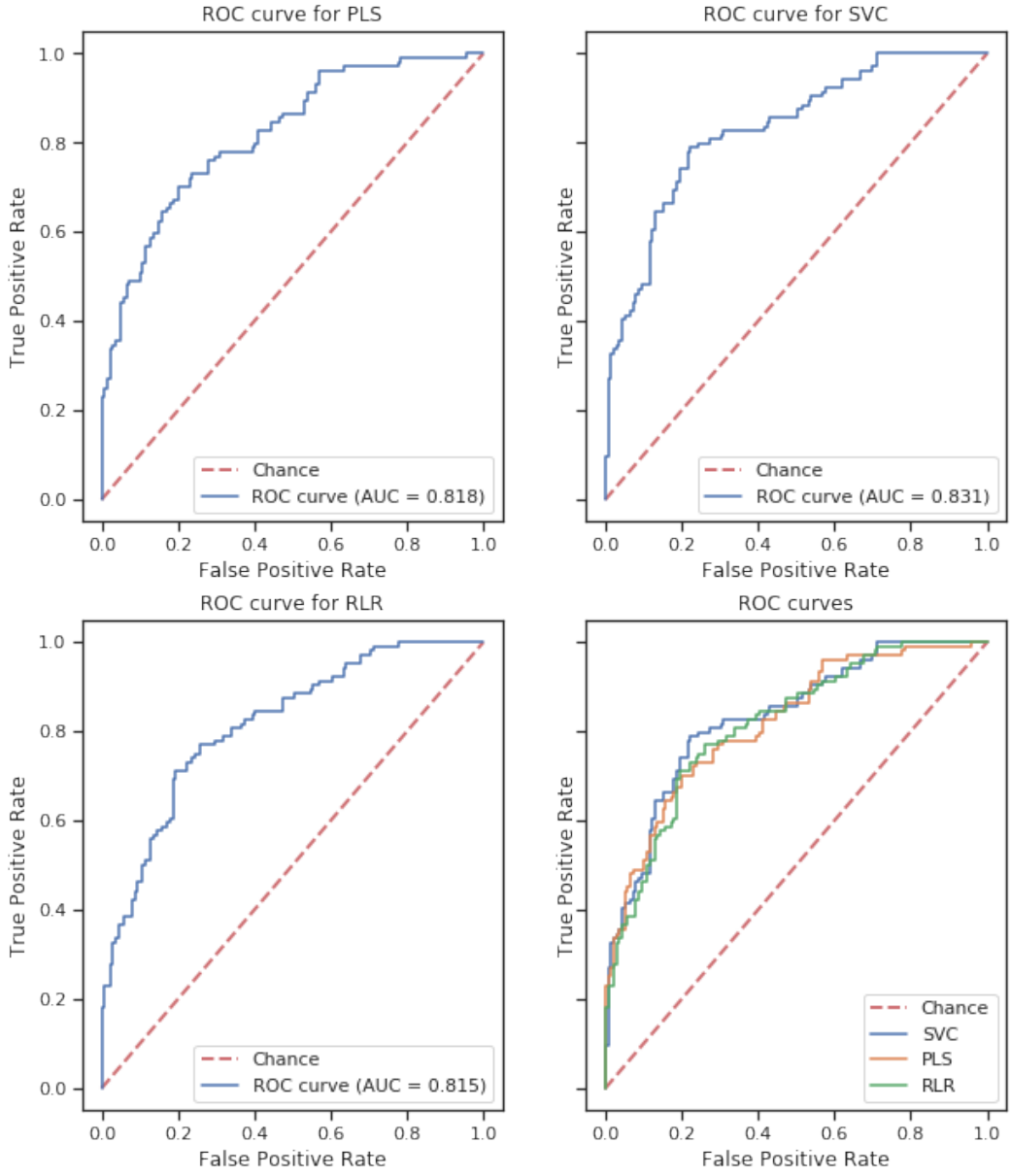

Figure 4.1: ROC curves for PLS, SVC and RLR

	ROC AUC	Precision	Recall	F1
LR	0.777 ± 0.020	0.673 ± 0.045	0.578 ± 0.050	0.622 ± 0.047
Elastic net	0.791 ± 0.031	0.703 ± 0.179	0.551 ± 0.292	0.519 ± 0.194
RDA	0.797 ± 0.019	0.673 ± 0.047	0.602 ± 0.051	0.635 ± 0.045
NSC	0.669 ± 0.048	0.589 ± 0.055	0.630 ± 0.068	0.607 ± 0.056
Random forest	0.797 ± 0.032	0.737 ± 0.045	0.533 ± 0.036	0.617 ± 0.028
RLR	0.803 ± 0.028	0.701 ± 0.045	0.602 ± 0.020	0.647 ± 0.027
SVC	0.766 ± 0.028	0.661 ± 0.050	0.547 ± 0.023	0.598 ± 0.033
PCA	0.793 ± 0.032	0.669 ± 0.042	0.644 ± 0.067	0.655 ± 0.052
SPCA	0.797 ± 0.030	0.700 ± 0.066	0.637 ± 0.034	0.667 ± 0.047
MLCC	0.787 ± 0.027	0.658 ± 0.037	0.619 ± 0.030	0.637 ± 0.024
PLS	0.804 ± 0.024	0.630 ± 0.020	0.754 ± 0.038	0.686 ± 0.027
FDNN	0.798 ± 0.032	0.482 ± 0.251	0.622 ± 0.312	0.541 ± 0.274

Table 4.1: Metrics for all models obtained from nested cross-validation procedure

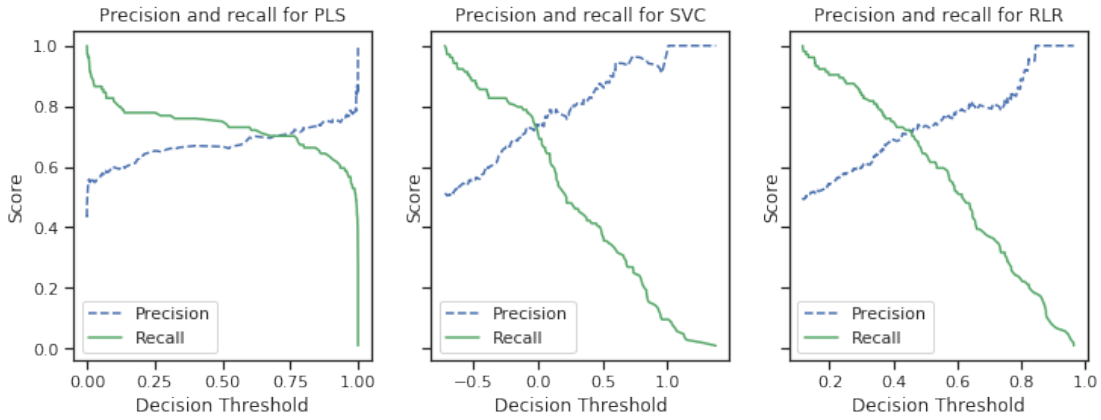

Figure 4.2: Precision and Recall vs decision threshold for PLS, SVC and RLR

The values on horizontal axis for SVC are not in range $[0, 1]$ as the classifier does not return probabilities but scores representing distances from margins. We can see that plots for RLR and SVC are similar but the one for PLS is notably different. For PLS recall decreases (precision increases) quite slowly in the interval $[0.15, 0.85]$ whereas for SVC and RLR both recall and precision seem to change linearly for the whole range of decision thresholds. As mentioned in the section 2.1, in case of diagnosis tool false negatives are much more costly than false positives. The most crucial questions remains: what will be the obtained precision if we would like to have classifier with recall at level r. For $r = 0.95$ the values of precision are 0.556, 0.516 and 0.527 for PLS, SVC and RLR respectively. For $r = 0.8$ these values change to 0.596, 0.689 and 0.641. This suggests that SVC and RLR offer a little better precision-recall tradeoff but for the standard reference value of recall 0.95 PLS is better. Although these values do not seem to be very promising it is worth comparing them with results from the literature. In [52] three methods are analyzed - the one introduced in the paper (PCA-AE-Ada), and two most popular models: classifier based on correlation with 70 selected gene signatures ([46]) and classifier based on relapse score using 76 selected gene signatures ([48]). PCA-AE-Ada outperforms the others but yields only 0.714 ROC-AUC score which is significantly lower than for the selected PLS model. What is more, in [12] the model using prior knowledge obtains accuracy 0.558 at fixed recall level 0.9 whereas PLS achieves 0.650 accuracy which is significantly higher. In all papers these results are deemed promising as such classifier would be able to help reducing number of patients unnecessarily receiving chemotherapy. Nonetheless, such comparison is not perfectly justified as different data sets were used, namely GSE2034 in [12], [48] and GSE2034, GSE4922, GSE6532, GSE7390, GSE11121 in [52]. However, considering that our data set contained GSE6532, GSE7390, GSE11121 we can still claim that our performance is comparable or even better than the one reported in state-of-the-art literature on breast cancer metastasis prediction.

4.2.2 Feature selection

In spite of decent performance of classifiers from the previous section we have to consider some limitations required for a model to be used as a part of medical diagnosis process. We already discussed precision-recall tradeoff and importance of high sensitivity. Furthermore, black box nonlinear models are not acceptable. Although, they are able to detect complicated nonlinear dependencies within feature space, the relation between input and the output of the model is not explicit. For example, for non linear dimensionality reduction algorithm this means being unable to connect directly a new set of variables with the input features. Doctors need to be able to justify their diagnosis and point the biomarkers influencing their decision, making the nonlinear methods undesirable in most applications. Although, the chosen algorithms from previous section do not follow such scheme the interpretation of the models is not that straightforward. For PLS the relation between

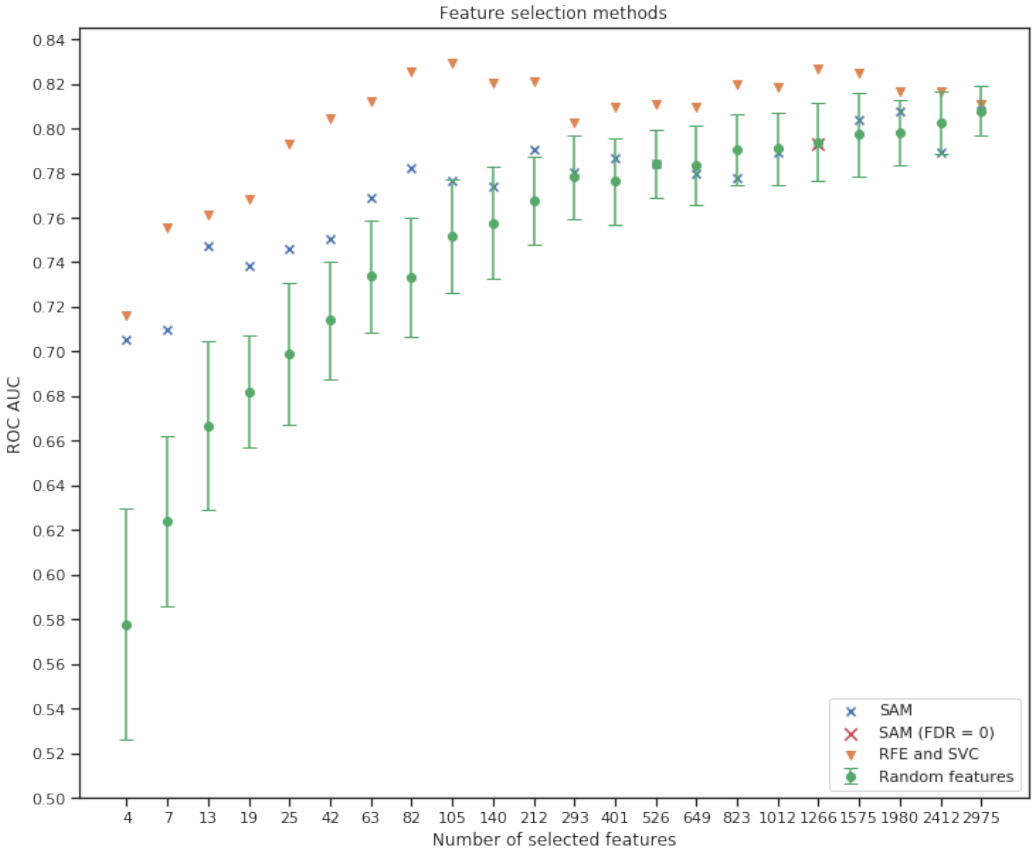

Figure 4.3: ROC AUC score for feature selection methods

input and output feature is quite complicated. For SVC and RLR one can look at the weights assigned to each variable to measure its importance, however having lots of non zero coefficients may affect the robustness of such interpretation. In order to tackle these problems we consider feature selection methods. In spite of some considerable drawbacks like discarding a number of features instead of coding them in the reduced form or usually yielding worse prediction performance than feature extraction, feature selection is often favourable. It is common that in medical applications finding a small subset of genes influencing the development of condition is much more valuable than creating best performing classifier, as discovering related biomarkers can be an important step in identifying the genesis of the disease.

In order to see how feature selection methods perform for different number of selected variables (corresponding to values of Δ in SAM) we fit a logistic regression model and measure ROC AUC score. Also, to have an idea how reliable are the selected predictors, for each number of features we randomly rank variables and use them to fit a classifier. To estimate the effect of randomization we repeat the process 30 times and report mean and standard deviation of our score. The results are presented in the figure 4.3.

In this figure we can see ROC AUC scores on test set for logistic regression

model fitted using different number of variables selected by SAM, RFE using SVC and random feature selection. The number of predictors is on the x axis. The red x (at number of features equal to 1266) corresponds to $\Delta = 2.4571$ - lowest value for which 90-th percentile of estimated FDR was 0, indicating that all selected genes are influencing the response. However, 1266 variables selected is not plausible result, as that many genes are not probable to impact the disease. Nonetheless, there are few interesting things to notice in the figure. First of all, we can see that RFE with SVC performed better for every number of features than SAM. Secondly, score for RFE increased with number of predictors until reaching maximum at 105 predictors, on contrary to SAM, suggesting better stability. It indicates that well performing, strongly regularized classifier allows to discover important variables better than a method based on statistical properties of high dimensional data. LR with 105 predictors selected using RFE achieved ROC AUC score of 0.829, which is almost as good as ROC AUC reported by fitted SVC classifier and better than the one for PLS from section 4.2.1. This indicates that not only feature selection obtains competitive results with feature extraction methods but also that only 105 out of 12179 genes are enough to get a very good classifier. What is more, as the number of variables increases the difference between random selection and other methods decreases. It suggests, that only a small subset of features carries the majority of predictive information and random sample of increasing size has a higher chance of containing it. We can also compare how many common elements have the lists of genes returned by RFE and SAM. For 13 selected predictors only 2 and for 105 only 14 were chosen by both algorithms. The other disadvantage of feature selection methods is that in order to obtain a stable set of variables one needs much more samples than we have in the training set ([16]). Therefore, although these 105 predictors resulting in best classifier seem to be the genes responsible for metastasis development, a professional would need to carefully investigate this claim, performing a biological examination of selected features. Nonetheless, as the performance of LR using these 105 variables is almost the same as the performance of regularized SVC from section 4.2.1 we would select as our final model the first one, due to better interpretability and simplicity.

Chapter 5

Conclusions

In this thesis we aimed at reviewing and applying data mining methods in the context of the prediction of breast cancer metastases. For this purpose, we used gene expression data, recognizing challenges corresponding to high dimensional data analysis. To tackle them we exploited a variety of regularized methods and dimensionality reduction algorithms. Out of the latter we explored separately two major types: feature extraction and feature selection. Advantages, disadvantages of each classifier were discussed and related tradeoffs were analyzed. We also used feature bagging technique to considerably improve the performance of logistic regression, obtaining a powerful Random Logistic Regression model. Furthermore, we thoroughly inspected model/algorithm selection tools like nested cross-validation to be able to pick the best ones with minimized risk of biased selection. Moreover, we incorporated limitations dictated by the field of disease development diagnosis, e.g. larger significance of recall optimization and the doctors' demand on having a profound justification of the examination in the form of interpretable models or even selected genes.

In scope of literature review we compared performance of chosen methods with results from the other papers. However, in the field of the prediction of breast cancer metastases this is not that trivial. The main issue is the wide variety of gene expression data sets used and lack of publicly available code that could be used to reproduce the results. It appears, that initiatives similar to ICLR Reproducibility Challenge are yet to come in this area. The next problem is that the majority of the papers incorporates only simple statistical methods like hierarchical clustering and correlation study, not exploiting the recent developments in the field of machine learning. Furthermore, even methodological errors might happen in papers with hundreds of citations, e.g. performing cross-validation only after dimensionality reduction. Some of them are highlighted in [2]. The mentioned issues usually appear, due to lack of researcher with statistical background in the group working on biomedical studies. In the future more effort should be put to connect professionals from both fields to reduce the risk of publishing biased or even unreliable outcomes.

The results presented in this thesis suggest that data mining methods can be

exploited in the field of prediction of breast cancer metastases. However, recognizing the obtained precision on the recall level set at 0.95, we must state the these models cannot be used as standalone decision makers but only as supplementary tools. Moreover, as the differences during the algorithm selection process were not large, we may assume that the gene expression data is not enough to be able to predict breast cancer metastases. Perhaps, other biological markers need to be incorporated, e.g age, sex, fragment of patient's medical history, environment, etc. Future work in this field may aim at including these factors in the analysis. Apart from that, it should address the reproducibility of other papers and a creation of common data set that could be used in all studies on the topic. This crucial advancement would allow building more complex models as the number of samples would be larger and also easier comparison between past and future result, which is an essential step for a measurable development in the field of the prediction of breast cancer metastases.

Bibliography

[1] Charu C. Aggarwal. *Data classification : algorithms and applications.* Chapman and Hall/CRC, 1st edition, 2014.

[2] Christophe Ambroise and Geoffrey J McLachlan. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the National Academy of Sciences of the United States of America*, 99(10):6562–6, may 2002.

[3] Sako Arts. *Dimensionality Reduction of Gene Expression Data.* Mater's thesis, Eindhoven Univeristy of Technology, 2018.

[4] Tanya Barrett, Stephen E. Wilhite, Pierre Ledoux, Carlos Evangelista, Irene F. Kim, Maxim Tomashevsky, Kimberly A. Marshall, Katherine H. Phillippy, Patti M. Sherman, Michelle Holko, Andrey Yefanov, Hyeseung Lee, Naigong Zhang, Cynthia L. Robertson, Nadezhda Serova, Sean Davis, and Alexandra Soboleva. NCBI GEO: archive for functional genomics data sets—update. *Nucleic Acids Research*, 41(D1):D991–D995, nov 2012.

[5] Michaela Bayerlová, Kerstin Menck, Florian Klemm, Alexander Wolff, Tobias Pukrop, Claudia Binder, Tim Beißbarth, and Annalen Bleckmann. Ror2 Signaling and Its Relevance in Breast Cancer Progression. *Frontiers in Oncology*, 7:135, jun 2017.

[6] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[7] B.M. M Bolstad, R.A A Irizarry, M. Astrand, and T.P. P Speed. A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Variance and Bias. *Bioinformatics*, 19(2):185–193, jan 2003.

[8] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[9] Terence A. Brown. *Gene cloning and DNA analysis : an introduction.* Wiley-Blackwell, 2016.

[10] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort,

Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. sep 2013.

[11] Gavin C. Cawley and Nicola L. C. Talbot. On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, 11(Jul):2079–2107, 2010.

[12] Han Yu Chuang, Eunjung Lee, Yu Tsueng Liu, Doheon Lee, and Trey Ideker. Network-based classification of breast cancer metastasis. *Molecular Systems Biology*, 3(140):1–10, 2007.

[13] Emily Clough and Tanya Barrett. The Gene Expression Omnibus Database. *Methods in molecular biology (Clifton, N.J.)*, 1418:93–110, 2016.

[14] Aaron Defazio, Francis R Bach, and Simon Lacoste-Julien. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. *CoRR*, abs/1407.0, jul 2014.

[15] Sandrine Dudoit, Jane Fridlyand, and Terence P Speed. Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. *Journal of the American Statistical Association*, 97(457):77–87, mar 2002.

[16] Liat Ein-Dor, Or Zuk, and Eytan Domany. Thousands of samples are needed to generate a robust gene list for predicting outcome in cancer. *Proceedings of the National Academy of Sciences of the United States of America*, 103(15):5923–8, apr 2006.

[17] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874, 2008.

[18] M. Gaasenbeek, E. S. Lander, D. K. Slonim, M. L. Loh, P. Tamayo, C. Huard, H. Coller, J. P. Mesirov, T. R. Golub, C. D. Bloomfield, J. R. Downing, and M. A. Caligiuri. Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring. *Science*, 286(5439):531–537, 1999.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene Selection for Cancer Classification using Support Vector Machines. *Machine Learning*, 46(1/3):389–422, 2002.

[21] Xiyi Hang and Fang-Xiang Wu. Sparse Representation for Classification of Tumors Using Gene Expression Data. *Journal of Biomedicine and Biotechnology*, 2009:1–6, 2009.

[22] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009.

[23] John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[24] R. A. Irizarry, Bridget Hobbs, Francois Collin, Yasmin D Beazer-Barclay, Kristen J Antonellis, Uwe Scherf, and Terence P Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, apr 2003.

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[26] I. T. Jolliffe. *Principal component analysis*. Springer, 2002.

[27] Yunchuan Kong and Tianwei Yu. A Deep Neural Network Model using Random Forest to Extract Feature Representation for Gene Expression Data Classification. *Scientific Reports*, 8(1):16477, dec 2018.

[28] Olivier Ledoit and Michael Wolf. Honey, I Shrunk the Sample Covariance Matrix. *The Journal of Portfolio Management*, 30(4):110–119, jul 2004.

[29] George Lee, Carlos Rodriguez, and Anant Madabhushi. An Empirical Comparison of Dimensionality Reduction Methods for Classifying Gene and Protein Expression Datasets. In *Bioinformatics Research and Applications*, pages 170–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[30] Ying Lu and Jiawei Han. Cancer classification using gene expression data. *Information Systems*, 28(4):243–268, jun 2003.

[31] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, 2012.

[32] National Human Genome Research Institute. Fact Sheets about Genomics. https://www.genome.gov/about-genomics/fact-sheets, 2015.

[33] D. V. Nguyen and D. M. Rocke. Tumor classification by partial least squares using microarray gene expression data. *Bioinformatics*, 18(1):39–50, jan 2002.

[34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS-W*, 2017.

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron

Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[36] Sebastian Raschka. Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. nov 2018.

[37] Mark Schmidt, Nicolas Le Roux, Francis Bach, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, sep 2013.

[38] Haipeng Shen and Jianhua Z. Huang. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of Multivariate Analysis*, 99(6):1015–1034, jul 2008.

[39] Piotr Sobczyk, Malgorzata Małgorzata Bogdan, and Julie Josse. Bayesian dimensionality reduction with PCA using penalized semi-integrated likelihood. *Journal of Computational and Graphical Statistics*, 26(4):826–839, jun 2017.

[40] Piotr Sobczyk, Stanislaw Wilczynski, Julie Josse, and Malgorzata Bogdan. *varclust: Variables Clustering*, 2019.

[41] C. Sotiriou, S.-Y. Neo, L. M. McShane, E. L. Korn, P. M. Long, A. Jazaeri, P. Martiat, S. B. Fox, A. L. Harris, and E. T. Liu. Breast cancer classification and prognosis based on gene expression profiles from a population-based study. *Proceedings of the National Academy of Sciences*, 100(18):10393–10398, 2003.

[42] R Tibshirani, Michael J Seo, G Chu, Balasubramanian Narasimhan, and Jun Li. *samr: SAM: Significance Analysis of Microarrays*, 2018.

[43] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays. *Statistical Science*, 18(1):104–117, feb 2003.

[44] Laura Toloşi and Thomas Lengauer. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*, 27(14):1986–1994, jul 2011.

[45] V G Tusher, R Tibshirani, and G Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences of the United States of America*, 98(9):5116–21, apr 2001.

[46] Laura J. Van't Veer, Hongyue Dai, Marc J. Van de Vijver, Yudong D. He, Augustinus A M Hart, Mao Mao, Hans L. Peterse, Karin Van Der Kooy, Matthew J. Marton, Anke T. Witteveen, George J. Schreiber, Ron M. Kerkhoven, Chris Roberts, Peter S. Linsley, René Bernards, and Stephen H. Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415(6871):530–536, 2002.

[47] Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC bioinformatics*, 7:91, feb 2006.

[48] Yixin Wang, Jan GM Klijn, Yi Zhang, Anieta M Sieuwerts, Maxime P Look, Fei Yang, Dmitri Talantov, Mieke Timmermans, Marion E Meijer-van Gelder, Jack Yu, Tim Jatkoe, Els MJJ Berns, David Atkins, and John A Foekens. Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. *The Lancet*, 365(9460):671–679, feb 2005.

[49] Britta Weigelt, Johannes L. Peterse, and Laura J. Van't Veer. Breast cancer metastasis: Markers and models. *Nature Reviews Cancer*, 5(8):591–602, 2005.

[50] M. West, C. Blanchette, H. Dressman, E. Huang, S. Ishida, R. Spang, H. Zuzan, J. A. Olson, J. R. Marks, and J. R. Nevins. Predicting the clinical status of human breast cancer by using gene expression profiles. *Proceedings of the National Academy of Sciences*, 98(20):11462–11467, sep 2001.

[51] Stanisław Wilczyński. *Reduction of dimensionality by sparse subspace clustering.* Bachelor's thesis, University of Wrocław, 2017.

[52] Dejun Zhang, Lu Zou, Xionghui Zhou, and Fazhi He. Integrating Feature Selection and Feature Extraction Methods with Deep Learning to Predict Clinical Outcome of Breast Cancer. *IEEE Access*, 6:28936–28944, may 2018.

[53] Shunpu Zhang. A comprehensive evaluation of SAM, the SAM R-package and a simple modification to improve its performance. *BMC bioinformatics*, 8:230, jun 2007.

[54] J. Zhu and Trevor Hastie. Classification of gene microarrays by penalized logistic regression. *Biostatistics*, 5(3):427–443, jul 2004.

[55] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, jun 2006.