Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

Data Mining and Computational Intelligence in Bioprocessing

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Biomedical Engineering

by

Marcus Eger
Registration Number 1229249

to the Faculty of Technical Chemistry at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Christoph Herwig

Vienna, 21st December, 2017

Marcus Eger	Christoph Herwig

Erklärung zur Verfassung der Arbeit

Marcus Eger
Kanitzgasse 13-19/1/2, 1230 Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. Dezember 2017

Marcus Eger

Acknowledgements

I would like to express my gratitude to Prof. Christoph Herwig for overseeing my work and giving me the opportunity to carry out this thesis.

I deeply thank my parents for their unconditional trust, timely encouragement, and endless patience. Finally, I would like to express my deepest gratitude to Corina, my fiancé and partner, for her support and guidance throughout the years.

Abstract

Biotechnology production plants offer a wealth of information in recorded bioprocessing data through modern data logging and archiving systems. This rich amount of data allows to optimize single bioprocesses, conduct quality control, and characterize processes. Due to the variety of different data logging systems, the resulting recorded bioprocess data is stored in different document formats, storage and database systems, and largely unstructured shapes. State of the art methods of bioprocess data alignment include the development of specific extraction scripts and manual data alignment. A large part of the analytical process is thus put into time-consuming manual data alignment processes. Machine Learning (ML) techniques can uncover hidden patterns in these seemingly unstructured shapes and thus allow for the automatic alignment of bioprocess data, independent of different storage formats and shapes. It is thus potentially advantageous to apply ML techniques in order to ease the time-consuming effort put into the data alignment process.

The aim of this thesis is to develop a bioprocess data extraction and alignment framework based on different Machine Learning techniques for data preprocessing and classification. The data set used in the scope of this thesis consists of different online- and offline-recorded batch processes collected from six different companies in the pharmaceutical, bioreactor, and biosensor industry (see appendix). All data sources were anonymized and recorded values replaced by random values prior to any processing steps. Each recorded batch is partitioned into a vectorized grid, where every data entry is represented by a single cell on the grid. Features for the classification process are built from cell properties and their surrounding cell neighborhood information for a given radius. Due to the high-dimensionality of the resulting feature space and to ensure maximum variability, the input dimension is reduced using Principal Component Analysis (PCA). The processed feature set is tested on two different classifiers, Stochastic Gradient Descent (SGD) with L2 regularization (Support Vector Machine) and Gradient tree Boosting. Tests were run using different neighborhood distances, different training/testing ratios, and different sets of hyperparameters for the chosen classifiers.

The variation in training/testing ratios showed that the variance between the highest and lowest test result steadily decreased for an increasing number of training samples. Based on the resulting averaged classifier F-scores for different neighborhood radii tests, a neighborhood distance radius of 2 demonstrated a good agreement between specificity

and sensitivity for the model learning curve without overfitting the prediction models. Between the two classifiers, the Gradient tree Boosting method achieved an overall higher prediction accuracy than the Support Vector Machine. The results of the feature importance tests for the cell features showed that the features data type and string similarity contributed the most during the training phase. The results for the feature importances for different neighborhood directions indicated a strong bias towards the cell information below each cell. The final prediction model achieved an average F-score of 86.27 with a low standard deviation of only 0.0477.

A machine-learning based extraction method for bioprocess data proved to be succesful, but with limitations. Improvements in the parsing process of the bioprocess data, the addition of new cell features, and a higher information content in the cell neighborhood features would greatly refine the accuracy of the prediction models. Future opportunities to expand the work done in this thesis would be an extension of the hyperparameter optimization for the Gradient Boosting classifier, improved sampling methods to reduce the class label imbalance in the training set, and a comparison of different nearest neighbor metrics in relation to the respective feature importance.

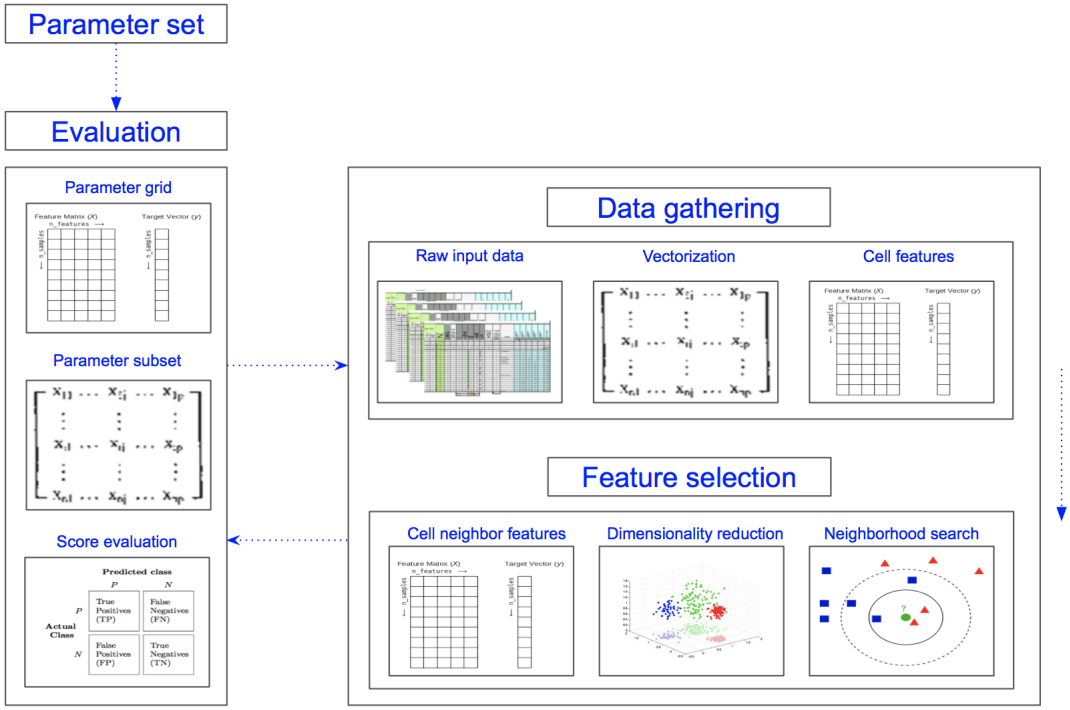


Figure 1: Bioprocess data extraction and prediction framework with the three building blocks data gathering, feature selection, and evaluation.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Modern biotechnology production plants utilize data recording and archiving systems to log every change in system parameters, sensor data, or environmental factors over time [9]. This wealth of information in recorded bioprocess data is used in the analysis of process outcome, process prediction, and error tracking [10]. Time series of bioprocess are stored into standard tabular file formats that contain a time axis, process variable columns, parameter settings, and recorded environment variables. Performing process analysis on the resulting recorded tabular data sheets requires careful extraction of the data and domain-based knowledge of the underlying recorded processes and their respective structure. Standard methods of data extraction for bioprocess sheets often require manual conversion, data interpretation by engineers and bioinformaticians or further process knowledge. These techniques are time consuming and prone to human error. Specific extraction algorithms may be applied to interpret the data sheet structure, but often fail to capture variations in the recording process or tabular structure.

The transformation of bioprocess sheets into multi-dimensional arrays represents the vectorized basis for the application of Machine Learning (ML) algorithms to the extraction problem. Machine Learning algorithms can infer the complex tabular structures in different bioprocess sheets by learning patterns on the transformed, multi-dimensional arrays in a supervised manner. Features can be built using a mixture of domain-specific knowledge, e.g common terms for bioprocess variables and parameters, and exploration of local and global neighborhood relations on the array structure. Discriminative models like Support Vector Machines (SVM) and Gradient Boosting can estimate predictions for new and unseen structures based on the learned feature structure. Given a sufficiently large amount of diverse sheet structure, a data extraction framework can be trained to ease the time-consuming effort and knowledge put into traditional extraction methods.

The development of an automatic method for the extraction of bioprocess data would thus be highly benefitial for bioprocess engineers, bioinformaticians, and researchers.

1.2 Aim of the Thesis

The aim of this thesis is to develop an automatized framework for data mining of complex bioprocess data structures collected from different systems through the use of Machine Learning (ML) techniques. Principal Componenten Analysis (PCA) is used to reduce the high-dimensional input space. Support Vector Machines (SVM) and Gradient Boosting are used as classifiers to establish a self-learning framework in order to learn typical structures of bioprocess data sheets. The resulting structure is then analyzed and statistically evaluated. As a result, new bioprocess data consisting of various structures can be automatically labeled.

1.3 Methodological Approach

The bioprocess data for this study is acquired from six different bioprocess reactor types. The implementation of the thesis is performed in the Python environment to analyze the provided data and create evaluation plots and results. All data sheets for training and testing are manually labeled by an experienced bioprocess engineer. A pipeline object is used to stack the preprocessing and prediction models into a single chain and ease the control of hyperparameters. The pipeline object is then placed into a grid object that allows for training and evaluation using different sets of hyper-parameter ranges. Every subset of hyper-parameters that is drawn from a given parameter range distribution is evaluated using 10-fold cross validation to prevent overfitting. All predicted labels are compared and evaluated against the manually labeled ground truth data. Preprocessing models, predictive models, the pipeline framework, and the grid search object are built using the python scikit learn library [40].

1.4 Structure of the Thesis

Chapter 1: Introduction
The first chapter of this thesis introduces the topics of Bioprocessing and the use of Machine Learning as a method of exploring and learning the input structure of bioprocess data. The motivation for the work and the methodology are briefly explained.

Chapter 2: Background
The background chapter presents the results of the literature research conducted during the course of this thesis. The principles of Machine Learning, preprocessing methods, supervised classification, predictive models, and evaluation measures are introduced. Furthermore, the structure of bioprocess data sheets is presented and explained in detail.

Chapter 3: Methods
The methods chapter describes in detail how different Machine Learning techniques are applied in a full framework and evaluated using different scoring metrics on the given test and training data.

Chapter 4: Results The results of the evaluation analysis are presented in chapter four. A comparison between different radii in feature building, classifiers, sets of hyperparameters, and preprocessing methods is visualized in error plots with different metrics, confusion matrices, and tables.

Chapter 5: Discussion
In the discussion chapter, the results in chapter four are analyzed and interpreted. Additionally, limitations of different methods and statistical significance are further discussed.

Chapter 6: Conclusion
The conclusion chapter provides a review of the used approach and outcome of this work. A brief examination of the achieved results and encountered challenges are further analyzed. Finally, future improvements and suggestions are given.

CHAPTER 2

Background

This chapter provides a basic overview on the underlying Machine Learning basics, dimensionality reduction, kd-trees, string similarity metrics, and classifiers in the context of this thesis. Due to decades of research in the field of Machine Learning, this chapter is limited to the necessary basics that are used in the final framework. The first part presents a gentle introduction to general learning approaches, under -and over-fitting, the bias-variance-tradeoff, and the usage of model parameter and hyper-parameters in section 2.1. Section 2.2 introduces the concept of the Principal Component Analysis (PCA) for dimensionality reduction. Section 2.3 describes the theory of kd-trees and the concept of nearest neighbor search. Section 2.4 introduces the Jaro-Winkler distance for string similarity estimation. Section 2.5 and 2.6 introduce the two predictive models that are used throughout the context of this thesis, the Gradient Boosting classifier and the Support Vector Machine.

2.1 Machine Learning in General

The goal of Machine Learning [35] can be understood as the process of gaining and using experience from data to improve computational methods without being explicitly programmed. Given a data set X of n observations and a set of corresponding labels Y, one might seek to learn a mapping from input to output in order to predict newly observed data. The sample set X can have many shapes, for instance images, videos, voice recordings or text data. In general, Machine Learning algorithms can be classified by the level of human supervision that the respective model requires [31].

Types of algorithms:
Supervised learning features a set of training observations x_i with labels y_i that is trained on a learning algorithm $h(x)$ to map the input space $\mathcal{X}$ to the output space $\mathcal{Y}$ by learning a function $h : \mathcal{X} \to \mathcal{Y}$. Supervised learning tasks can be further categorized into

continuous (regression) and discrete (classification) problems. For instance, a typical classification task is the training of a spam filter on a set of emails and their corresponding text content together with the respective labels, e.g 'spam' or 'no spam'. Based on the trained classifier, new emails can be categorized by using a probabilistic approach or simple distance metrics. Unlike discrete predictions in classification, regression models do not quantize the data into different bins, but predict continuous values. Popular supervised learning approaches include k-Nearest Neighbors (kNN), Linear and Logistic Regression, Support Vector Machines (SVM), Neural networks (NN), and Decision Trees/Random Forests. The supervised learning approaches used in this thesis are the Support Vector Machine (SVM) and Gradient Boosting (Ensemble Learning).

In **unsupervised learning** the learning algorithm h receives an unlabeled set of training observations x to make predictions on unlabeled sets of data by modeling the underlying structure of the data. A prominent unsupervised learning task is clustering. For instance, one seeks to group visitors of a website into different clusters to draw sufficient inference for future content. Given a set of features, e.g age or location, the input data can be clustered into different visitor groups or visualized into hierarchies of groups.
Another common task in unsupervised learning is dimensionality reduction as described in the following section. The primary goal of dimensionality reduction is to find a more compact representation of the input data while preserving enough information to draw statistical inference from the data. This can be further grouped into feature selection and feature extraction. In feature selection the original feature space is reduced to return a subset of the original data. This allows for simplification of the model interpretation, shorten the training time, and reduce the risk of overfitting. Feature extractions seeks to rebuild the original feature space by using subsets of old features to construct a new feature set.

The category of algorithms that processes partially labeled training sets is referred to as **semi-supervised learning**. A prominent example are deep belief networks (DBN), a mixture of supervised and unsupervised learning algorithms. A DBN is a multi-layer network that consists of several layers of Restricted Boltzmann Machines (RBM), a type of stochastic feature extractor. In a first step, these feature extractors are trained by reconstructing the original input given a set of probabilistic constraints. By training the whole network from the bottom to the top, complex features are built upon simple shapes. These feature extractors are trained in an unsupervised fashion by minimizing the reconstruction error between the original input and the reconstructed feature. The whole network is then trained again in a supervised manner by fine-tuning each layer based on the previous layer using labeled training data.

Reinforcement Learning is based on a so-called learning agent. The agent interacts in a trial and error manner with its environment where suitable actions are rewarded and negative actions penalized. The assignment of rewards and penalties is drawn from a policy configuration, comparably to a set of rules in law. A strategy is then learned

by maximizing the amount of rewards over time until the agent acts according to the given policy. A typical application of reinforcement learning is the area of robotics, where robots are trained to walk using reinforcement learning algorithms [45].
An overview of different Machine Learning methods and their respective input/output structure is shown in Figure 2.1. Learning algorithms can also be categorized by the

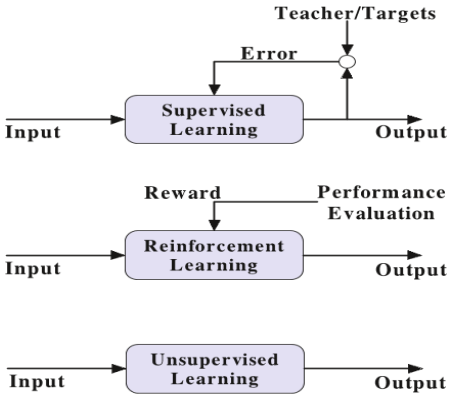


Figure 2.1: Basic learning structures of the three machine learning branches: supervised learning, reinforcement learning, and unsupervised learning [46].

way that data is fed into a learning system. A learning problem is defined by a given space $\mathcal{X}$ and a set of labels $\mathcal{Y}$ with the goal of assigning a label in $\mathcal{Y}$ to every instance of the given space $\mathcal{X}$. A statistical assumption that can be made is that there exists a probability distribution over this product space of $\mathcal{X} \times \mathcal{Y}$ and that there exists a training set that is i.i.d from this distribution. A function that maps the instances of $\mathcal{X}$ to target labels in $\mathcal{Y}$ is called a hypothesis h. In **Batch learning** the training set that is i.i.d drawn from a certain distribution is used to generate a hypothesis h [5]. This hypothesis is expected to generalize well on unseen instances sampled from the distribution. If no statistical assumptions about the underlying distribution of the data is made, instances of data are sequentially processed by feeding chunks of examples to the learning algorithm. After every iteration, the algorithm corrects a previously learned hypothesis by adjusting weights based on new instances. This is referred to as **Online Learning** [5].

Model parameters & Hyperparameters
When modeling data using a learning algorithm, a model utilizes a set of internal variables to construct a certain hypothesis. A learning algorithm is referred to as non-parametric if the complexity of functions that it is capable of learning is increasing with the amount of processed training data. Examples of nonparametric models include k-nearest neighbors, decision trees, and Support Vector Machines (SVM). If the complexity of functions is restricted to a certain set and thus independent of the amount of training observations, the model is referred to as a parametric model. Examples of parametric models include Logistic Regression, Linear Discriminant Analysis (LDA), and Naive Bayes classifiers. When there is insufficient prior knowledge about the target function it is thus advanta-

geous to apply non-parametric learners on a set of observations to adjust the hypothesis to the complexity of the data.
Unlike model parameters, **hyperparameters** are external configuration parameters of a model that are not inferred from the data set itself. These values are set prior to the training process and govern the way that the algorithm will behave when trained on different types of data. For instance, in neural networks hyperparameters can control the number of layers, input neurons, output neurons, and further regularization settings.
When trained in a supervised manner, input data in neural networks is forwarded and processed through the network to predict values. These predicted values are then compared with the the desired output and the error is back-propagated through the model to adjust the weights of the network.
Figure 2.2 shows a so-called autoencoder structure, a type of neural network that tries to learn an approximation of the identity function and thus reconstruct the input signal. This is achieved by applying certain constraints on the network structure, e.g limiting the number of hidden nodes in the hidden layer to force a compressed representation of the original input. Similarly to a Principal Component Analysis, an autoencoder can therefore output a lower-dimensional reconstruction of the input data if the hyperparameters are set in a way that enforce a compressed processing of data. Instead of reconstructing the input data, the neural network structure in Figure 2.3 seeks to predict a target vector y from input vectors x by constructing a hypothesis $h(x)$ based on the processed input data through different layers. This demonstrates well, how different hyperparameter settings that control layers and nodes, can influence the way data is processed and output. The goal of finding an optimal combination of hyperparameters for a certain problem

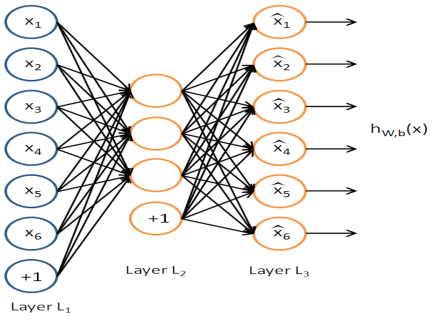


Figure 2.2: Autoencoder with six input neurons, three hidden nodes, and six output neurons [39].

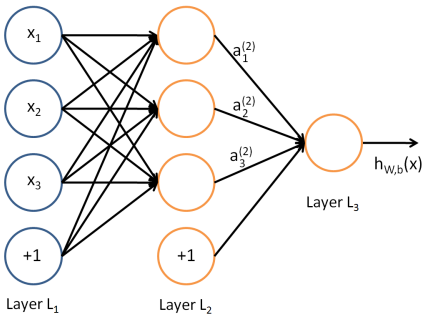


Figure 2.3: Neural Network model with three input neurons, three hidden nodes, and one output node in the output layer [39].

is referred to as hyperparameter optimization. In order to compare different sets of hyperparameters, a suitable metric has to be chosen first to rank different combinations based on their ability to maximize the chosen matric. Given a model $\mathcal{M}$, a loss function $\mathcal{L}$, a set of test data X_{test}, a set of training data X_{train}, and a set of hyperparameters λ

we can formulate the goal of finding the optimal set of hyperparameters λ^* as

$$\lambda^* = \arg\min_{\lambda} \mathcal{L}(X_{test}; \mathcal{A}(X_{train}; \lambda)) \tag{2.1}$$

where the model $\mathcal{M}$ is constructed by a learning algorithm $\mathcal{A}$. The search for the optimal set of hyperparameters is often strongly linked to the level of model complexity. If the model is too complex, it will fit well on the training data, but fail to generalize on the unseen test data. If in turn the model complexity is too low, the model will fail in learning to capture the structure of the training data. This is referred to as the **bias-variance-tradeoff**.
Given a target variable Y, the input data X, and a model estimation $f(\hat{X})$, a simple squared error prediction can be defined as

$$Err = E[(Y - f(\hat{x}))^2] \tag{2.2}$$

which can be further decomposed into

$$Err = \Big(E[\hat{f}(x)] - f(x)\Big)^2 + E\left[\Big(\hat{f}(x) - E[\hat{f}(x)]\Big)^2\right] + \sigma_e^2 \tag{2.3}$$

where the first term is the squared bias, the second term represents the variance of the model, and the third term is an irreducible noise error. The error due to **bias** is the difference between the correct (target) value of our data and the predicted values of the model based on the training data. Assuming that the model is trained repeatedly using different hyperparameters, shuffled training data or given general randomness in the data set itself, the bias is a measure of how wrong in general the algorithms predictions are compared to the true target values of the data set.
The error due to **variance** is the variability of the models prediction for a given observation. Thus, variance is a measure of how much the predictions of a model vary for a given observation or the algorithm's tendency to capture random information while disregarding the real, underlying structure of the data.
A high bias in a predictive model is a sign of underfitting the training data and thus having a low model complexity. A high variance indicates overfitting, thus the model's complexity is too high and fits the model too closely to randomness and noise in the training set. Figure 2.4 shows a simple two class separation tasks and the possible model complexities underfitting, overfitting, and a bias-variance tradeoff between high and low complexity.

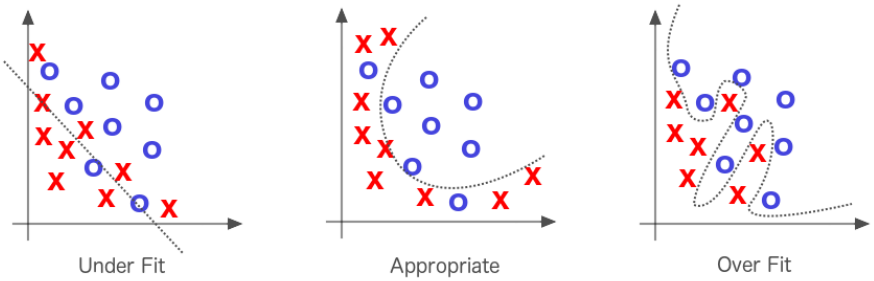


Figure 2.4: Different model complexities on a two-class separation problem [22].

2.2 PCA

The aim of the Principal Component Analysis (PCA) is to apply an orthogonal transformation to a possibly correlated set of data, so that the principal component of the transformed system carries the largest possible variance [34]. Thus, each component of the transformed system carries the highest possible variance given its orthogonal position to the prior component. A high-dimensional, noisy data set can therefore be represented using a lower-dimensional subspace transformation by computing a meaningful basis to re-express the noisy data.
Given two $m \times n$ data matrices X and Y and their linear transformation P, where X is the original data set and Y corresponds to the transformed presentation of X, so that

$$PX = Y \tag{2.4}$$

we try to find a suitable transformation P to represent X using the principal components $p_1, ..., p_m$. Finding a suitable basis P is coherently linked to the type of features that are needed in Y, that is low noise and less redundancy. The noise ratio can be expressed by the signal-to-noise ratio (SNR)/variance ratio

$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}} \tag{2.5}$$

where σ^2_{signal} and σ^2_{noise} are the respective variances of the data set. SNR values $\gg 1$ express low-noise data, while values $\ll 1$ are linked to noisy data. A simple way to quantify the redundancy between the variables in the data set is to express the correlation between all possible combinations of pairs. By computing the covariances of all possible variable combinations, we obtain the relationships between all the measurements in the given data set. The covariance matrix for a given $m \times n$ matrix X is defined as

$$S_X = \frac{1}{n-1} X X^T \tag{2.6}$$

where $\frac{1}{n-1}$ is the unbiased estimator of X. A covariance value of zero corresponds to an uncorrelated pair of variables, while a covariance $\neq$ zero expresses the magnitude of

correlation. Rewriting the covariance matrix in terms of the orthonormal matrix P gives

$$S_Y = \frac{1}{n-1} PAP^T \tag{2.7}$$

with $A = XX^T$. Given a diagonal matrix D and an eigenvector matrix E of A, a symmetric matrix A can be expressed as $A = EDE^T$. Using the eigenvectors of XX^T to fill the rows of the matrix P, A can be defined as $A = P^T DP$ with $P = E^T$. Using equation 2.7 and the new definition for the matrix A it can be seen that $S_Y = \frac{1}{n-1} P(P^T DP)P^T$ simplifies to

$$S_Y = \frac{1}{n-1} D \tag{2.8}$$

proving that P diagonalizes S_Y. The eigenvectors of XX^T are the principal components of X and the variance of X along vector p_i is the i-th diagonal value of S_Y. Figure 2.5 shows the first two principal components of the iris data set, a collection of 150 observations with four different features from three different types of Irises, after the transformation. The compressed representation using the first two principal components captures 95 % of the variance of the original data set.

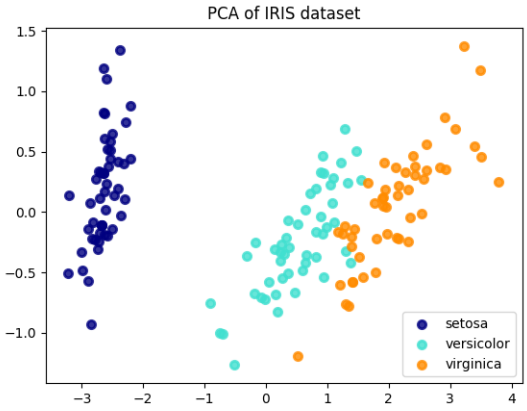


Figure 2.5: Principal Component Analysis (PCA) on the iris dataset. [40]

2.3 Nearest neighbor search and kd-trees

The nearest neighbor structure is defined as follows: Given a set of P points in an n-dimensional space $\mathcal{R}^n$, we seek a structure that can be queried for any point q on the space to find the nearest point P^* to q in the set P [3]. The nearest neighbor search requires a distance metric that computes the vectorial distance between a query point q and possible nearest neighbors in P. Common distance metrics are the l_s norms between

two points q and p as $||p - q||_s$, so that

$$||x|| = (\sum_{i}^{d} |x_i|)^{\frac{1}{s}} \tag{2.9}$$

The distance metric must be symmetric, non-negative, and fulfill the triangle inequality. A tree structure is constructed by first assigning all available points to the so-called root node and then recursively partitioning the points into child nodes. Depending on the chosen tree structure, a termination criteria is defined, so that the partitioning process stops once the criteria is met. Common criteria include the maximum leaf size, defining the maximum amount of points per leaf, and the maximum leaf radius, depending on a given radius within the hyperball of the leaf.
A tree search or query is defined by the underlying condition that is applied to the search:

- **ϵ-close neighbor search** (range search): Return the set of points $P_c \subset P$, so that $p_i \in P_c$ and $d(p_i, q) \leq \epsilon$
- **$k-$nearest neighbor search** (k-NN): Return the set of k points $P_c \subset P$, so that $\forall p_i \in P_c$ and $\forall p_j \notin P_c \rightarrow d(p_i, q) \leq d(p_j, q)$

Different types of queries using a kd-tree structure can be seen in Figure 2.6. A ϵ-NN search is performed by traversing the tree from the root and recursively exploring the child nodes in the intersection the hyperball created from a given radius ϵ around the query point q. If a leaf node is found, each data point of the respective node must be checked for the given radius criterion. This is necessary to avoid leaf nodes that are fulfilling the triangle equality but might contain points that do not fulfill the radius criterion. A k-NN search is performed by first locating the leaf node that contains the query point and then filter all adjacent leaf nodes until k points are found. Once k points have been found, all surrounding points outside of the maximum radius of k points are excluded and sorted again by decreasing distances until the k-nearest points are found.
A simple and fast tree construction is the so-called **kd-tree** structure. Data points are recursively partitioned for every node, so that two sets are created by splitting along a single dimension of the input data. The construction algorithm runs until the termination criteria are met. The performance of kd-tree queries depends on the splitting criteria.
In contrast to kd-trees, **ball trees** structure the data points based on a pre-defined distance metric. Each point of a sample node is assigned to the closest center of the node's children, which have the maximum possible distance between them. In a first step, the center of the data points is calculated and the center point is chosen as the point with the maximum distance to the first node child. The center of the second node child is then chosen to be the point with the maximum distance to the first center point. This construction allows for significantly faster queries if the underlying distribution of the points is captured by the node construction.
A set of different variations for the shape parameters radius r and Minkowski distance p can be seen in Figure 2.7.

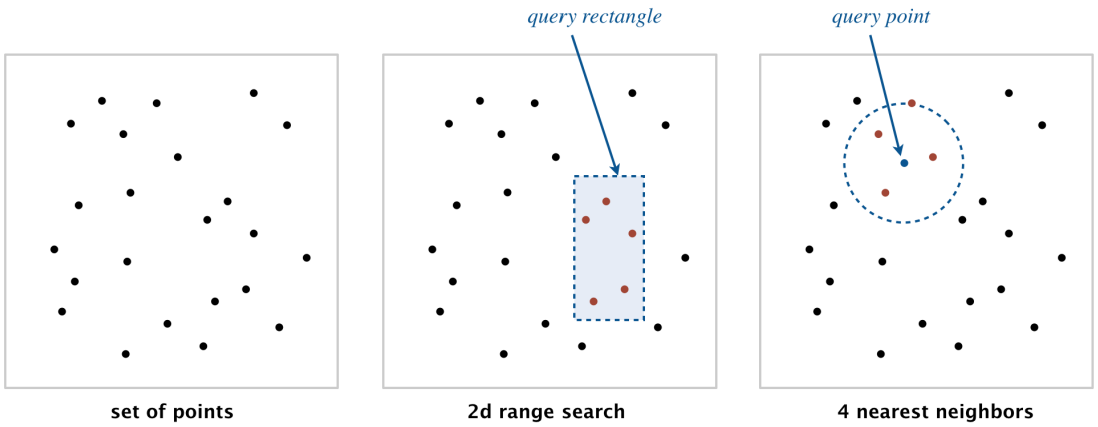


Figure 2.6: Rectangle and ball point query on a set of points using a kd-tree structure [43].

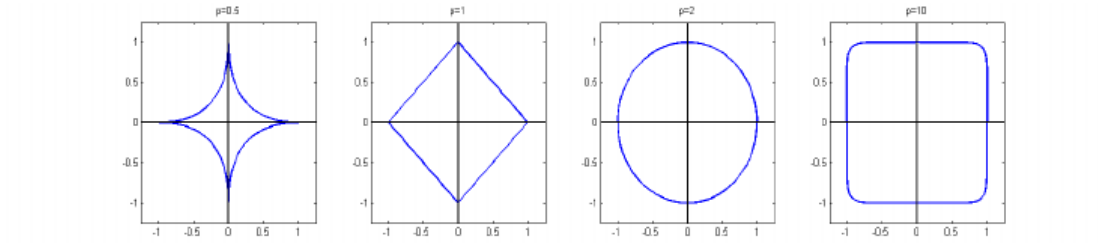


Figure 2.7: Unit circle for different Minkowski distances p_i [29].

Given a partitioned ball tree structure, a Minkowski distance of 2, and a radius r, the resulting number of possible neighbor cells $N(r)$ around a certain query point q for $r > 0$ inside of a ball query can be determined by the Gauss circle problem

$$N(r) = \{(x, y) \in \mathbb{Z}^2 | x^2 + y^2 \leq r^2\} \tag{2.10}$$

with $N(r) = 1, 5, 13, 29, 49, 81, 113, 149, 197, 253, 317$ for $r = 0, ..., 10$. The exact solution for the gauss circle problem of any radius r and infinitely large values of i is given by the series

$$N(r) = 1 + 4 * \sum_{i=0}^{\infty} (|\frac{r^2}{4i+1}| - |\frac{r^2}{4i+3}|) \tag{2.11}$$

where i is the number of iterations that controls the approximation

2.4 Jaro-Winkler distance

Given two strings s_1 and s_2 and their respective string lengths $|s_1|$ and $|s_2|$, the character window size W is given by

$$W_{i,j} = \frac{\min(|s_1|, |s_2|)}{2} \tag{2.12}$$

This defines the maximum allowed distance for two characters c_i and c_j to be considered as matching characters. The Jaro distance d_j between the two strings is given by

$$d_j(s_1, s_2) = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{if } m \neq 0 \end{cases} \tag{2.13}$$

where t is half of the number of transpositions, and m is the number of matched characters. Given a character c_1 of s_1 and a character c_2 of s_2 and their maximum distance given by equation 2.12, the characters are considered as matching if they are identical and have a distance that is equal or below the character window size of the respective strings. The number of transpositions is given by the number of matching characters with different sequence order divided by 2.
To add an extra weight to matching prefixes, the Jaro-Winkler distance metric utilizes a prefix factor p. This allows for higher similarity scores to strings that match in substring regions of a given length l_{ij}. Given two strings s_1 and s_2, the Jaro-Winkler distance metric is defined as

$$JW(s_1, s_2) = d_j(s_1, s_2) + pl_{i,j}(1 - d_j(s_1, s_2)) \tag{2.14}$$

where $d_j(s_1, s_2)$ is the Jaro distance between the two strings, p is a predefined prefix factor, and $l_{i,j}$ is the number of matching prefix characters in s_1 that match the prefix characters in s_2 [13]. A similarity score of 1 indicates an exact similarity, and a score of 0 indicates no similarity between two strings. The Jaro-Winkler algorithm procedure for an array c of string tuples can be seen in algorithm 1.

Algorithm 1 Jaro-Winkler string similarity algorithm

Initialization:
Input: text array c with n pairs of strings

1: **procedure**
2: **for** each string pair s_i, s_j in c **do**
3: Compute Jaro distance estimation $d_j(s_1, s_2)$
4: Compute the Jaro-Winkler distance estimation given by $JW(s_1, s2) =$
5: $\begin{cases} d_j(s_1, s_2) + pl_{i,j}(1 - d_j(s_1, s_2)) & \text{if tuple contains no empty string(s)} \\ 0 & \text{if tuple contains empty string(s)} \end{cases}$
6: **end for**
7: **end procedure**

2.5 Gradient Boosting for classification tasks

Domain-specific problems often require additional expert-level knowledge to capture the underlying relationship of the observed data set and adjustment of the applied predictive

model. Non-parametric models, like support vector machines (SVM) and neural networks pose a way to adjust the model to the training data in a supervised fashion without explicitly constraining to a certain probability density. Instead of training a single predictive model on the data set, a series of weak predictive models can be chosen to average different votes on the training sample. This is referred to as Ensemble Learning [14]. In the scope of this thesis, the following subsections are reduced to boosting and bagging techniques.

Ensemble Learning

Ensemble Learning aims to find the combined predictive solution of a set of different or identical learners, e.g classifiers or experts. The primary intention behind the usage of multiple learners is to average multiple predictions and thus draw a collective decision. This has several statistical, computational, and representational advantages over using a single classifier.
Finding the ideal solution for a given classification task can be challenging in terms of choosing the appropriate classifiers and finding an adequate set of hyper-parameters for the given data set (**model selection**). Utilizing ensemble methods is not guaranteed to deliver a better classification result, but reduces the risk of choosing a poor estimator by averaging the ensemble results. A fundamental requirement for successful model selection using ensemble learning is a sufficient level of diversity for the chosen learners to ensure variability in prediction errors. Ensembles can either be constructed by using a series of different classifiers or a series of similar classifiers with different sets of hyper-parameters. This allows for different error estimations that can be combined and averaged, so that the overall ensemble error can be determined as

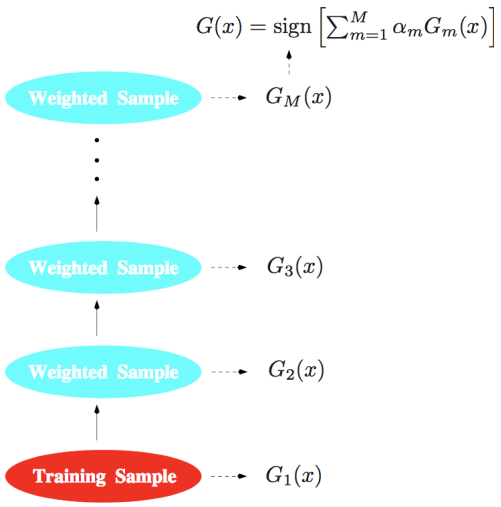
$$\hat{E} = \bar{E} - D \tag{2.15}$$

where $\bar{E}$ is the average error of the single classifiers and D is a ratio expressing the level of diversity among the single components. An ensemble of single learners trained on weighted subsets of a dataset can be seen in Figure 2.8. At each step of the algorithm, a series of weights $w_1, ..., w_i$ is applied to the training data (x_i, y_i) and readjusted after every iteration. Depending on the results of each learner $G_i(x)$ of the ensemble, weights are then increases or decreased. Thus emphasis is put on difficult observations that induce misclassification, while easier observations are decreased. The final predictor $G(x)$ is the weighted sum of the single predictors, so that

$$G(x) = \sigma\Big(\sum_{m=1}^{M} \alpha_m G_m(x)\Big) \tag{2.16}$$

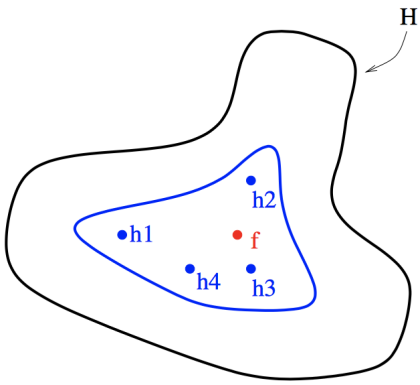
where σ is the sign function, α_m the weights assigned by the Boosting algorithm to adjust the power of different learners, and G_m the series of weak classifiers in the ensemble. Final predictions are then processed by a majority voting of the single classifiers.
Majority voting using multiple learners solves a common problem in the application of single prediction models. Given a small data set and a comparitively large hypothesis

Figure 2.8: Schematic flow of Adaptive Boosting (AdaBoost) (from elements of statistical learning) [18].



space $\mathcal{H}$, ensemble methods can successfully cast votes that are sufficiently distributed over the solution space. An averaged hypothesis is thus a good approximation to the underlying true function f [15]. A sample sketch of a solution space $\mathcal{H}$ with different hypotheses h_i by ensemble learners and the true function f can be seen in Figure 2.9.

Figure 2.9: Multiple hypotheses by different learners on the hypothesis space $\mathcal{H}$ [15].



Similarly, single learning algorithms may get stuck in local optima due to a limited local search scope.

Bagging

Bagging, an abbreviation for bootstrap aggregating, is a type of ensemble learning method that aims to improve unstable estimations [36]. Bagging can be seen as a

variance reduction method for a set of given base estimators, e.g decision trees for classification and regression. It can also be seen as a method to increase and stabilize the predictive power of classifiers and regression trees. Bagging consists of multiple uniform sampling steps with replacement from a given training set. A series of base classifiers is then fit iteratively on the subsets of the training set. The final prediction decision is a so-called majority-voting. Figure 2.10 shows a series of bagging predictors on a set of

Algorithm 2 Bootstrapping algorithm

Initialization:
Input data set (X, Y) with $(x, y)_i^N$ observations
Ensemble $\mathcal{D} = \{\}$
Number of chosen base-learners $L = (M_i, ..., M_k)$

1: **procedure** BOOTSTRAPPING MAJORITY PREDICTION
2: **for** $k = 1$ to L **do**
3: Draw a bootstrap sample S_k from X
4: Fit a classifier/base learner M_k with training data S_k
5: Add the classifier to the initialized ensemble, so that $\mathcal{D} = \mathcal{D} \cup D_k$
6: **end for**
7: **for** $i = 1$ to D **do**
8: Predict the target observation using D_i
9: **end for**
10: Chose the prediction using a weighted majority-vote over all trained classifiers
11: **end procedure**

ozone data. The prediction result of the final estimator is the weighted average of the single bagging predictors (red).

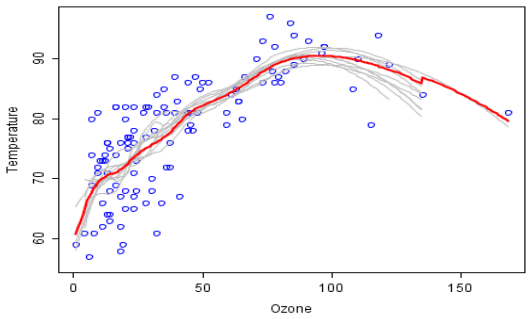


Figure 2.10: Different bagging estimator predictions (gray) on ozone data. The final averaged, weighted predictor is marked in red [40].

Gradient Boosting

Problem statement

We consider another supervised classification problem with a dataset $(x, y)_i^N$, where $x_1, ..., x_i$ is the set of input variables and $y_1, ..., y_i$ is the set of corresponding target variables. The aim is to reconstruct a function $f(x)$ that holds the dependence $x \to y$ by approximating a function $\hat{f(x)}$, so that a loss function $L(y, f)$ is minimized

$$\hat{f}(x) = \arg\min_{f(x)} L(y, f(x)) \tag{2.17}$$

The approximated function can also be expressed in terms of the conditioned expected loss function, so that

$$\hat{f}(x) = \arg\min_{f(x)} E_x[E_y(L(y, f(x)))|x] \tag{2.18}$$

where $E_y(L(f, f(x)))$ is the expected target variable loss and $E_x[.]$ is the expectation over the entire data set. The loss function can take different forms depending on the type of classification tasks, the nature of the data set, and the type of features. For a simple binary classification task, the loss function can be binomial. For continuous distributions, the L_2 squared or Huber loss function can be chosen. The different types of loss functions are later on described in detail.
In order to provide a computationally tractable explanation of the Gradient Boosting algorithm, the search space of the target function is parameterized, so that

$$\hat{f}(x) = f(x, \theta) \tag{2.19}$$

with the parametric function family $f(x, \theta)$ and thus

$$\hat{\theta} = \arg\min_{\theta} E_x[E_y(L(y, f(x, \theta)))|x] \tag{2.20}$$

where $\hat{\theta}$ is the set of approximated parameter estimations for the model. The parameter estimations are normally obtained by iterative numerical optimizations, where a chosen loss function is minimized on a given grid space. Numerical optimization of such loss functions is performed iteratively by

$$\hat{\theta} = \sum_{i}^{N} \Psi(\hat{\theta}_i) \tag{2.21}$$

where N is the maximum number of iterations and ψ an iterative minimization method.

Gradient Descent

A common iterative minimization method in machine learning is **the steepest gradient descent** algorithm. The Gradient Descent (GD) algorithm is an optimization algorithm that aims to find the local minimum of a given function space. Given a certain loss function L, the goal is to find the point of minimum error as it can be seen in Figure

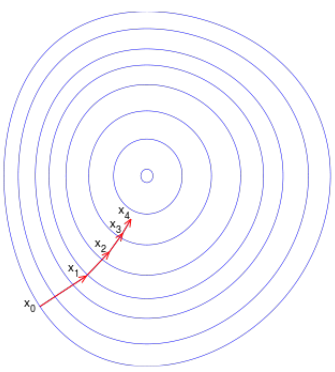


Figure 2.11: Steepest Gradient Descent over function space [47].

2.11. For instance, we would like to fit a linear function $f(x) = mx + b$ through a set of points (x_i, y_i) and thus find the best coefficients m and b. In order to find the best set of coefficients, an error function has to be defined in order to provide a function space to be minimized. For the given problem, the average sum of squared differences (SSD) is chosen as the error function. The SSD function is defined as

$$E(m, b) = \frac{1}{N} \sum_{i}^{N} (y_i - f(x_i))^2 \tag{2.22}$$

where y_i is the set of target variables and $\hat{f}(x)$ is the approximated function. Starting on a random point on the function surface area, the GD algorithm seeks to find the global minimum point $(b*, m*)$. The gradient at a certain point is defined as the partial derivative and its direction represents the largest rate of increase of the function. Since we seek to find the minimum of the error function, we would like to move in the negative direction of the gradient. The gradient descent algorithm can be summarized in the following three steps

1. Error calculation $E(m_i, b_i)$ by estimation of parameters (m_i, b_i)
2. Calculation of the partial derivatives $\frac{\partial E}{m_i}$ and $\frac{\partial E}{b_i}$.
3. Adjustment of estimations $m_{i+1} = m_i - \gamma \frac{\partial E}{m_i}$ and $b_{i+1} = b_i - \gamma \frac{\partial E}{b_i}$

where γ is the learning rate that controls the speed of convergence. Given the error function in equation 2.20, the partial derivatives w.r.t the SSD error function are

$$\begin{aligned} \frac{\partial E}{m_i} &= -\frac{2}{N} \sum^{N} x_i (y_i - f(x_i)) \\ \frac{\partial E}{b_i} &= -\frac{2}{N} \sum^{N} (y_i - f(x_i)) \end{aligned} \tag{2.23}$$

When an appropriate learning rate is chosen and a convex error function is used, the algorithm will converge to the local minimum.

Gradient Boosting algorithm

To fully comprehend the gradient boosting algorithm, a base learner $h(x, \theta)$ has to be defined first. Base learners can be drawn from different families, e.g decision trees or splines [19]. Given a base learner $h(x, \theta)$, a function estimate $\hat{f}(x)$, and step size (learning rate) γ, a function optimization step can be defined as

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \gamma h(x, \theta_t) \tag{2.24}$$

where γ_t is the variable learning rate at step t, θ_t the set of parameters at step t, and $\hat{f}_{t-1}$ the function estimate at the previous step. The minimization objective can be formulated as

$$(\gamma_t, \theta_t) = \arg\min_{\theta,\gamma} \sum^{N} L(y_i, \hat{f}_{t-1}) + \gamma h(x_i, \theta) \tag{2.25}$$

where $L(.)$ is the chosen loss function, and (x_i, y_i) the i-th observation of the training set. The prediction results of a gradient boosting regression algorithm of multiple observations of the function $x \sin(x)$ can be seen in Figure 2.12.

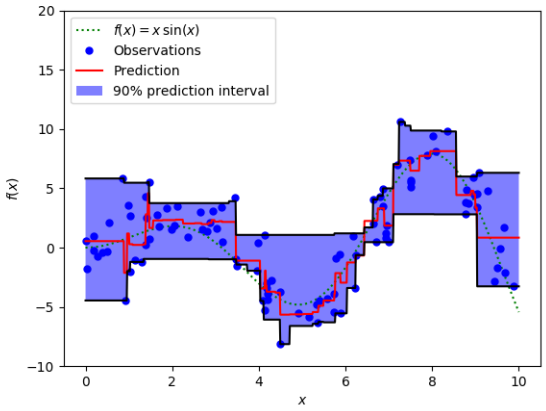


Figure 2.12: Gradient boosting regression prediction of noisy observations of the function $x \sin(x)$ [40].

Given a loss function $L(y, f)$ and a base learner $h(x, \theta)$, the model parameter estimation can be simplified by choosing a new base learner $h(x, \theta_t)$ that is parallel to the negative gradient on the training set. The negative gradient can be obtained by

$$g_t(x) = E_y\left[\frac{\partial L(y, f(x))}{\partial f(x)} | x\right]_{f(x)=\hat{f}^{t-1}(x)} \tag{2.26}$$

where $\frac{\partial L(y,f(x))}{\partial f(x)}$ is the partial gradient of the loss function L w.r.t to the previous approximation $\hat{f}^{t-1}$ of the target function $f(x)$. The complete Gradient boosting algorithm

can be seen below in Algorithm 3.

Algorithm 3 Gradient Boosting algorithm [7]

Initialization:
Input data set (X, Y) with $(x, y)_i^N$ observations
Chosen loss function $L(y, f)$
Chosen base-learner $h(x, \theta)$
Maximum number of iterations M or manual stopping criterion

1: **procedure** INITIALIZATION OF FUNCTION APPROXIMATION($\hat{f}_0$)
2: **for** $t = 1$ to M **do**
3: Computation of negative gradient $g_t(x)$
4: Fitting of new base-learner $h(x, \theta_t$ based on gradient
5: Estimate gradient step size: $\gamma_t = \arg\min_\gamma \sum_i^N L(y_i, \hat{f}_t - 1(x_i) + \gamma h(x_i, \theta_t))$
6: Update of function approximation: $\hat{f}_t \leftarrow \hat{f}_{t-1} + \gamma_t h(x, \theta_t)$
7: **end for**
8: **end procedure**

Loss functions
The selection of a particular loss function $L(y, f)$ is often influenced by the desired properties of the conditional distribution, e.g outlier robustness. Loss functions can be categorized by the underlying type of target variable y of the respective classification problem. For continuous target variables, one seeks to solve a regression problem. One of the most common continuous loss functions in regression tasks is the squared L_2-loss, which is defined as

$$L(y, f) = \frac{1}{2}(y - f)^2 \tag{2.27}$$

with the derivate $y - f$. The L_2 loss function penalizes large deviations between prediction and target variable, but disregards small deviations.
The Laplacian loss function (L_1)-loss is the median of the conditional distribution, so that

$$L(y, f) = |y - f| \tag{2.28}$$

The L_1 loss function minimizes the absolute differences between predicted values and target variable. As it can be seen, the L_2 error is much larger for outliers compared to the L_1 loss. A combination of the two previously described loss functions is the so-called huber loss function. The Huber loss function is a parameterized loss function that is defined as

$$L(y, f) = \begin{cases} \frac{1}{2}(y - f)^2, & \text{if } |y - f| \leq \delta. \\ \delta(|y - f| - \delta/2), & \text{if } |y - f| > \delta. \end{cases} \tag{2.29}$$

where δ is a robustness factor that controls the outlier sensitivity. The huber function thus incorporates L_1 and L_2 characteristics. The three described continuous loss functions can be seen in Figure 2.13.

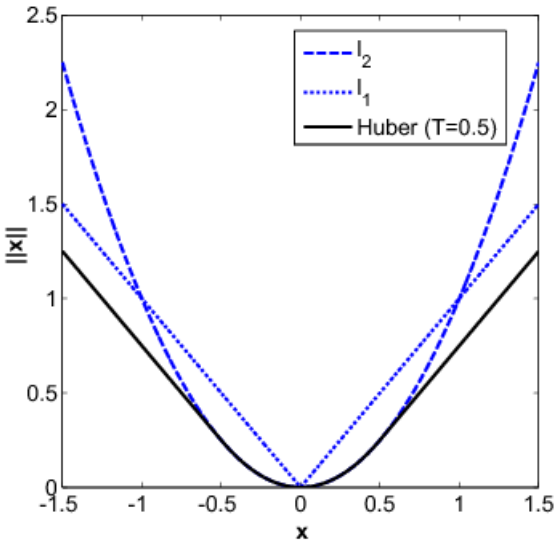


Figure 2.13: L_1, L_2, and Huber loss functions [17].

2.6 Support Vector Machines for classification tasks

Support Vector Machines (SVM) are a class of non-probabilistic supervised learning algorithms that were first introduced in 1964 by Vapnik and Chervonenkis for classification and regression tasks. The SVM is a so-called maximal margin classifier. Every object in the input space is represented by a vector in a vector space. This vector space can be effectively divided by constructing a hyperplane around the respective class objects. By maximizing the distance of the separating hyperplane to the training data, only the nearest vector points are chosen as the support vectors. This signifies that the mathematical description of the hyperplane only requires the surrounding support vectors. Thus, training data with a large distance to the hyperplane does not influence the shape of the separator.

Linear SVM

Given a set of n training observations x_i with $x \in \mathcal{R}^d$ and $i = 1, .., n$ and their corresponding class labels $y_i \in \{+1, -1\}$, we seek to learn a classifier $f(x)$ for a binary classification task, so that

$$f(x_i) = \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases} \tag{2.30}$$

A linear classifier can be described in the general form by

$$f(x) = w^T x + b \tag{2.31}$$

where w is a normalized weight vector and b the bias term. The linear classifier f forms the basis for the SVM algorithm and the separating hyperplane spanned by the SVM can thus by described by $H(x) = w^T x + b$. The output of the linear SVM is given by

$$g(w^T x + b) \tag{2.32}$$

where $g(.)$ is a threshold function that regulates the class membership of each observation x_i. The threshold function for the linear SVM has the following properties:

$$g(z) = \begin{cases} z \geq 0 & y = 1 \\ z < 0 & y = -1 \end{cases} \tag{2.33}$$

The threshold function properties fulfill the desired characteristics of the binary classification task in equation 2.28. The threshold function of the SVM is also referred to as the step function. Figure 2.14 shows a linear SVM with maximum-margin hyperplane for a binary classification task.

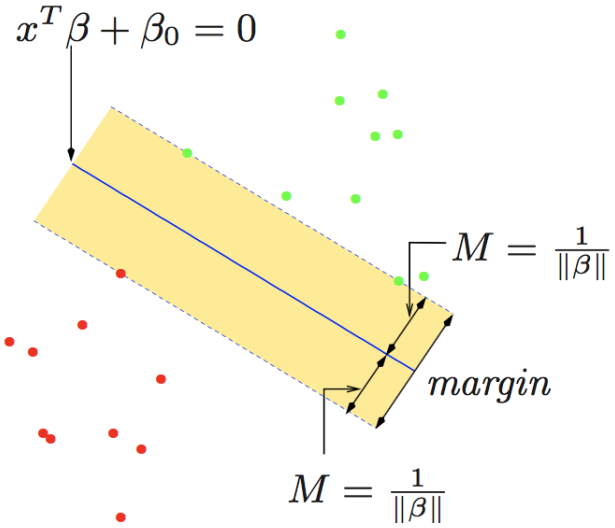


Figure 2.14: Linear SVM with maximum-margin hyperplane for a two class separation problem [18].

Margin estimation

The distance between the hyperplane and any observation x_i in the vector space can be inferred from the weight vector w. This is achieved by normalizing the weight vector $\frac{w}{||w||} = 1$, so that for any observation x_i, there is a point F given by

$$x_i - d_i \frac{w}{||w||} \tag{2.34}$$

where d_i is the distance of observation x_i to the hyperplane h. Since the weight vector is perpendicular to the decision boundary, F is on the decision boundary and fulfills $w^T x + b = 0$, so that

$$\begin{aligned} w^T(x_i - d_i \frac{w}{||w||}) &= 0 \\ d_i &= \frac{w^T x_i + b}{||w||} \\ &= \frac{1}{||w||}(w^T x_i + b) \end{aligned} \tag{2.35}$$

computes the distance to the hyperplane h given any observation x_i. Recalling equation 2.28, the class label y_i can be used to extend equation 2.33, so that

$$d_i = y_i \frac{1}{||w||}(w^T x_i + b) \tag{2.36}$$

where $d = \arg\min_{1,..,n} d_i$ is the margin that spans the hyperplane. The hyperplane is therefore the center of the minimum distances of each class. Observations that fulfill the minimum distance and are on the margin are so-called support vectors as it can be seen in Figure 2.15.

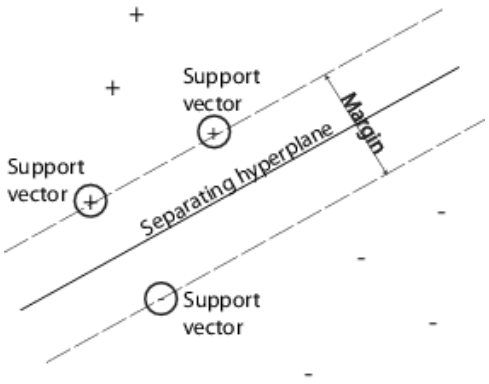


Figure 2.15: Maximum margin linear SVM on a binary classification problem with two support vectors on the + class and one support vector on the − class side of the hyperplane h.

Optimal hyperplane & dual form

Among all possible hyperplanes, we seek the hyperplane with the maximum distance d to any class y, so that:

$$\arg\max_{w,b} \Big(\frac{1}{||w||} \min_{i=1,...,n} (y_i(w^T x_i + b)) \Big) \tag{2.37}$$

Since the problem formulation is non-convex and the minimization of $\frac{1}{||w||}$ is equal to the maximization of $\frac{1}{2}||w||^2$ w.r.t w and b the optimization problem can be reformulated as

$$\arg\min_{w,b} \frac{1}{2}||w||^2 \text{ with } y_i(w^T x_i + b) \geq 1 \tag{2.38}$$

The optimization problem is now strictly convex using only linear constraints and can thus be solved using a simple quadratic solver. Instead of applying a quadratic solver to equation 2.36, the optimization problem can be solved in a way that allows the usage of kernel functions (see subsection xxx). By applying the rule of Lagrange multipliers, the constraints in equation 2.36 can be included into the optimization term. Given an optimization problem

$$\min_w f(w) \tag{2.39}$$

with the linear constraints $g_i(w) \leq 0$ for $i = 1, ..., k$ and $h_i(w) = 0$ for $i = 1, ..., l$ we can rewrite equation 2.37, so that

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_i^k \alpha_i g_i(w) + \sum_i^l \beta_i h_i(w) \tag{2.40}$$

and the optimization problem becomes $\min_w \max_{\alpha,\beta} \mathcal{L}(w, \alpha, \beta)$. This is referred to as the **primal problem**. The primal problem can be reformulated as the dual problem by switching the constraints, so that $\max_{\alpha,\beta} \min_w \mathcal{L}(w, \alpha, \beta)$. Applying the Lagrange multipliers in equation 2.38 on the SVM problem in equation 2.35, the Lagrange equation yields

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_i^n \alpha_i(y_i(w^T x_i + b))) \tag{2.41}$$

where $\mathcal{L}$ needs to be minimized w.r.t w and maximized w.r.t α. The corresponding dual problem is

$$\sum_i^n \alpha_i - \frac{1}{2}\alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{2.42}$$

where $\langle xx \rangle$ is the scalar product $x^T x$. The scalar product is later replaced by the kernel function K.
The decision boundary for an unknown observation x can now be computed by

$$y = \sigma\Big(\sum_i^n \alpha_i y_i \langle x, x_i \rangle + b\Big) \tag{2.43}$$

where $\sigma(x)$ is the sign function and y the predicted class label of observation x.

Nonlinear SVM & Kernel methods

Figure 2.14 and 2.15 show linearly separable classification cases. In the case of linearly inseparable data the Support Vector Machine fails to find an optimal solution given the linear constraints in the previous subsection, that do not allow any observations inside the margin region. This is referred to as the hard **margin** form of the SVM.
In order to soften the harsh constraints of the hard margin form, a so-called slack variable $\zeta_i \geq 0$ for the linear constraints is introduced. Correctly classified observations thus fulfill $\zeta_i = 0$, observations inside the margin $\zeta > 0$, and observations that are on the wrong side of the hyperplane $\zeta \geq 1$. The constraint are thus changed to

$$y_i(\langle w, x_i \rangle + b) \geq 1 - \zeta_i \tag{2.44}$$

where ζ_i is the newly introduced slack variable. This is referred to as the **soft-margin** form of the SVM. Misclassifications are now allowed, but punished by an error term $C\zeta_i$. The error weight C adjusts the relation of a low error rate and a maximum margin classification. Figure 2.16 shows the soft-margin form of the SVM on a linearly inseparable classification task for different values of the error rate term C.

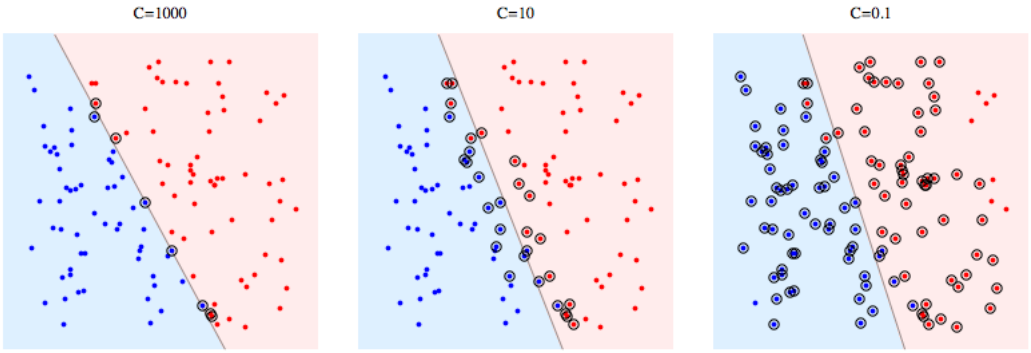


Figure 2.16: Soft-margin SVM on linearly inseparable classification task for different values of C [18].

Instead of balancing the relation between misclassification and maximum margin in the same feature space as the data set, observations can be mapped into a new set of quantities. These quantities, the so-called input features, are obtained by a feature mapping $\phi(x)$. Using the inner product rule of equation 2.40 and 2.41, these inner products can be replaced with the Kernel function K if we have data $x, z \in \mathcal{X}$ and a mapping table $\phi(x, z) \to \mathcal{R}^N$:

$$k(x, z) = \langle \phi(x), \phi(z) \rangle \tag{2.45}$$

where $K(x, z)$ is the Kernel mapping of observation x and z. Kernels can map the input data into a possibly higher-dimensional feature space to discover new linear relations between the observations in the given data set. These linear relations can then be divided

by a linear algorithm. The data set does not have to explicitly embedded into the new feature space, because the Kernel function solely relies on the inner product of the two vectors. This can be shown on a simple two-dimensional input space example $X \subseteq \mathbb{R}^2$ with the feature mapping

$$\phi : x = (x_1, x_2) \rightarrow \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1, x_2) \in F \tag{2.46}$$

where F is the new feature space with $F \in \mathbb{R}^3$. Given the inner product of x and z

$$\begin{aligned} \langle \phi(x), \phi(z) \rangle &= \langle (x_1^2, x_2^2, \sqrt{2}x_1 x_2), (z_1^2, ...) \\ &\vdots \\ &= \langle x, z \rangle^2 \end{aligned} \tag{2.47}$$

it can be seen that $k(x, z) = \langle x, z \rangle^2$. Figure 2.17 shows a kernel mapping scheme for Support Vector Machines for a linearly inseparable classification task using a polynomial kernel of degree 3 and a radial basis function kernel (RBF).

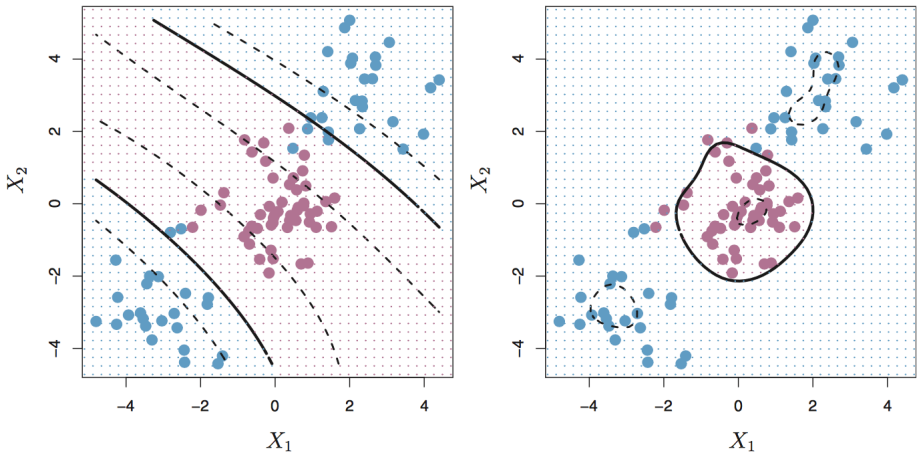


Figure 2.17: Left: An SVM with a polynomial kernel of degree 3 is applied to a non-linear data set. Right: An SVM with a radial kernel is applied [18].

The polynomial kernel used on the right-hand site in Figure 2.17 is defined as

$$K(x, y) = (x^T y + c)^d \tag{2.48}$$

where x and y are input vectors, d is the dimension of the target mapping, and c is a free parameter that controls the balance between high and low order polynomial terms in the Kernel mapping.
The RBF kernel used on the left-hand site of Figure 2.17 is defined by

$$K(x, y) = \exp\Big(- \frac{||x - y||^2}{2\sigma^2} \Big) \tag{2.49}$$

where $||x - y||^2$ is the squared Euclidian distance between the feature vectors x and y, and σ is a free parameter with the relation $\gamma = \frac{1}{2\sigma^2}$, so that

$$K(x, y) = \exp(-\gamma||x - y||^2) \tag{2.50}$$

As it can be seen from equation 2.49, the Kernel value decreases with increasing distance between the feature vectors with values between zero and one. The parameter γ controls the misclassification ratio of the predictions [27]. As γ increases, the model tries to avoid misclassifications by defining increasingly tighter hyperplanes around the different observations. While this effectively reduces the number of misclassifications, the model fails to generalize well and overfits the training data as it can be seen on the right hand site of Figure 2.18.

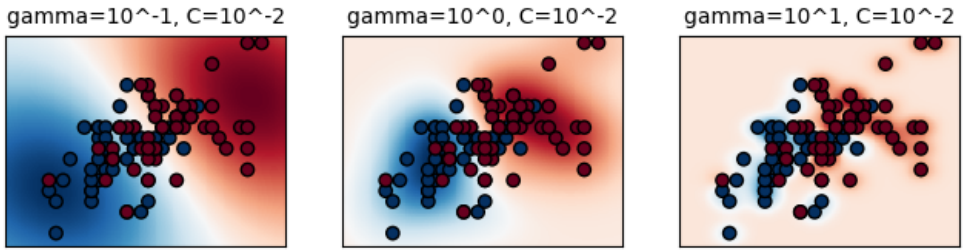


Figure 2.18: SVM classification with an RBF kernel on a two class problem for different settings of the control parameters γ (Left: $\gamma = 0.1$, Middle: $\gamma = 1$, Right: $\gamma = 10$) [40].

The scalar product in the dual form of the SVM in equation 2.40 can now be replaced by the Kernel function, so that

$$w(x) = \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j K(x_i, y_i) \tag{2.51}$$

where $K(x_i, x_k)$ is the Kernel mapping for input vectors x_i and x_j.
For classification problems that involve more than two classes of labels, a one-versus-all classification is applied [16]. Given a classification problem with N different classes, N different SVMs are fit to the data set, where each SVM compares a single class to the leftover $N - 1$ classes. For a series of resulting parameters $\beta_{0n}, ..., \beta_{pn}$, where the $n - th$ class is compared to the leftover $n - 1$ classes and $\beta_{0n} + \beta_{1n}x_1^* + ... + \beta_{pn}x_p^*$ is the series of SVM results for a specific class $n*$, the winning class is picked by choosing the largest series. This effectively equals to the highest confidence that a certain observation belongs to the $n - th$ class, given n trained SVM models.

CHAPTER 3

Methods

This chapter provides an overview of the methodology used during the scope of this thesis. The structure of the bioprocess data used for training and testing, the concept of local and global neighborhoods on the sheet structure, the model architecture and pipeline concept, and different statistical validation measures are described in this chapter. Figure 2.6 shows an overview of the processing pipeline.

3.1 Data set structure

The training and testing data set consists of six different, labeled bioprocess data sheets. The sheets were gathered from different bioreactor types to ensure a high diversity in the training and testing process. Each cell on the training and testing sheet has been manually labeled by expert bioprocess engineers before further computational processing was applied. The respective labels are categorized as following and can be seen in a sample structure in Figure:

- **Numerical variables**: Measured values, control values, and offset values
- **Time variables**: Time indices, process starts and stops, and reference times
- **Empty cells**: Cells that do not contain valuable information for the classification process or indicate a boundary on the sheet
- **Variable headers**: Cells indicating the start of a measurement process
- **Time headers**: Cells indicating the time index of a measurement
- **Parameter headers**: Cells indicating a header for an offset or control value
- **Units:** All cells that contain a unit of measurement

3.2 Feature-set structure

Cell-based features: For each cell on the grid structure, a local feature set is built from a set of simple data properties. The algorithm used to compute the string similarity measures is described in detail in section 2.4. The set consists of the following features:

- **Data type**: Vectorized feature for the respective data type, e.g (string, int, float, bool). To ensure the data type, each cell is casted to the respective data types to avoid data type mismatches, e.g float values in string format.
- **String header similarity**: averaged estimated similarity between the respective cell and a dictionary containing common bioprocess variables names
- **Unit similarity**: averaged estimated similarity between the respective cell and a dictionary containing common unit structures
- **Cell length**: Number of characters in the respective cell content
- **Digit ratio:** The ratio of digits found in the cell content given the overall length of the cell content.
- **Center of mass distance to numerical values**: Manhattan distance between the respective cell coordinate and the center of mass of numerical values on the respective sheet structure

As it can be seen from the list of cell-based features, the local feature set allows the classifier to differentiate between string based and empty, numerical or time based content, but is still limited in its predictive power for different header classes (see section 3.1). To increase the predictive power for each cell on the grid, the final feature set includes the local cell-based feature sets of the surrounding neighbor cells, given a certain radius r as described in section 2.3.

3.3 Model Architecture

The model architecture is divided into three main blocks, a data gathering block, a feature building/selection block, and the classification and evaluation block as it can be seen in Figure 3.1. The data gathering and feature building block are iteratively called and executed by the classification and evaluation block, given a set of different parameters to test. Given a subset of the parameters, the bioprocess sheets are first parsed, vectorized, and then concatenated into a single array. During the feature phase, a local feature set is first built on each cell on the array grid and extended in a second step by the feature sets of n neighboring cells depending on the size of the circle radius and the chosen minkowski metric in the parameter set (see section 2.3). After applying the dimensionality reduction on the resulting feature set, the data set is then split into a test and training fraction according to the split ration in the parameter subset. Finally,

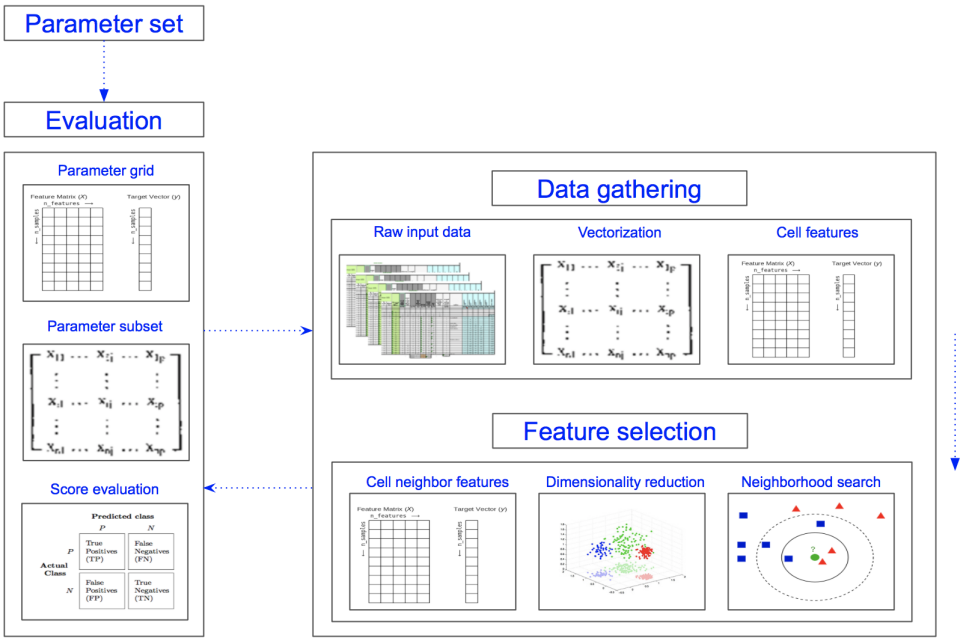


Figure 3.1: Model architecture with the three building blocks, data gathering, feature selection, and evaluation.

the test set is evaluated on the trained classifier on a chosen metric, e.g micro F1-scores. This process is then iteratively repeated with different subsets of the parameter grid.

CHAPTER 4

Results

This chapter presents the feature importance results for cell-based and global features, predictions results of training/test split variations, radius variations, and the classifier comparison. The feature importance results show the average feature importance value, the respective standard deviation, minimum and maximum value, and the relative standard deviation over 100 runs for each feature.
The training/test split comparison shows the classification results and confusion matrix using a split ratio of $0.1, 0.3, 0.5$ for each classifier respectively.
The radius variation test results are obtained by alternating the neighborhood circle radius for the input data between 1 and 3 and show the learning curve and macro/micro-averaged ROC results for each classifier.
The final classifier comparison results are obtained by 100 different test runs, each using a 10-fold cross validation using the macro-averaged F score. The final averaged training and testing results are shown row by row for each respective parameter subset.

4.1 Bioprocess data set statistics

The data set is comprised of six different bioprocess sheets, annotated by an expert bioprocess engineer. The overall cell content of the combined data set is 2334. The cell count is divided into seven different class labels and contains

- 1037 cells (44.4 %) for class **numerical variables**
- 83 cells (3.5 %) for class **time variables**
- 924 cells (39.6 %) for class **empty cells**
- 125 cells (5.4 %) for class **variable headers**
- 12 cells (0.514 %) for class **time headers**

- 25 cells (1.07 %) for class **parameter cells**
- 128 cells (5.48 %) for class **units**

4.2 Feature importance

	digit ratio	num. center	dtype	d_{header}	cell length	d_{unit}
avg.	0.027	0.015	0.337	0.301	0.131	0.188
σ	0.006	0.006	0.013	0.025	0.024	0.023
min	0.017	0.003	0.312	0.251	0.093	0.142
max	0.041	0.029	0.373	0.366	0.183	0.233
$\sigma_{rel}(\%)$	23.527	39.417	3.996	8.333	18.644	12.171

Figure 4.1: Averaged feature importance statistics of the Gradient Boosting classifier after 100 runs using a 10-fold split ratio for all cell-based features. Indices on the very left indicate the respective average, standard deviation, minimum value of all runs, maximum value of all runs, and the relative standard deviation in %. Columns from left to right indicate the feature digit ratio, numerical center, data type, string similarity for headers, cell length, and string similarity for units.

	right	above	left	below
avg.	0.253	0.065	0.230	0.453
σ	0.016	0.008	0.030	0.027
min	0.218	0.052	0.169	0.403
max	0.283	0.085	0.290	0.510
$\sigma_{rel}(\%)$	6.194	12.424	12.867	6.000

Figure 4.2: Averaged feature importance statistics of the Gradient Boosting classifier after 100 runs using a 10-fold split ratio for the four directions of the concatenated global feature sets. Indices on the very left indicate the respective average, standard deviation, minimum value of all runs, maximum value of all runs, and the relative standard deviation in %. Columns from left to right indicate the neighborhood features to the right, above, left, and below the target cell.

4.3 Prediction results for different train/test splits

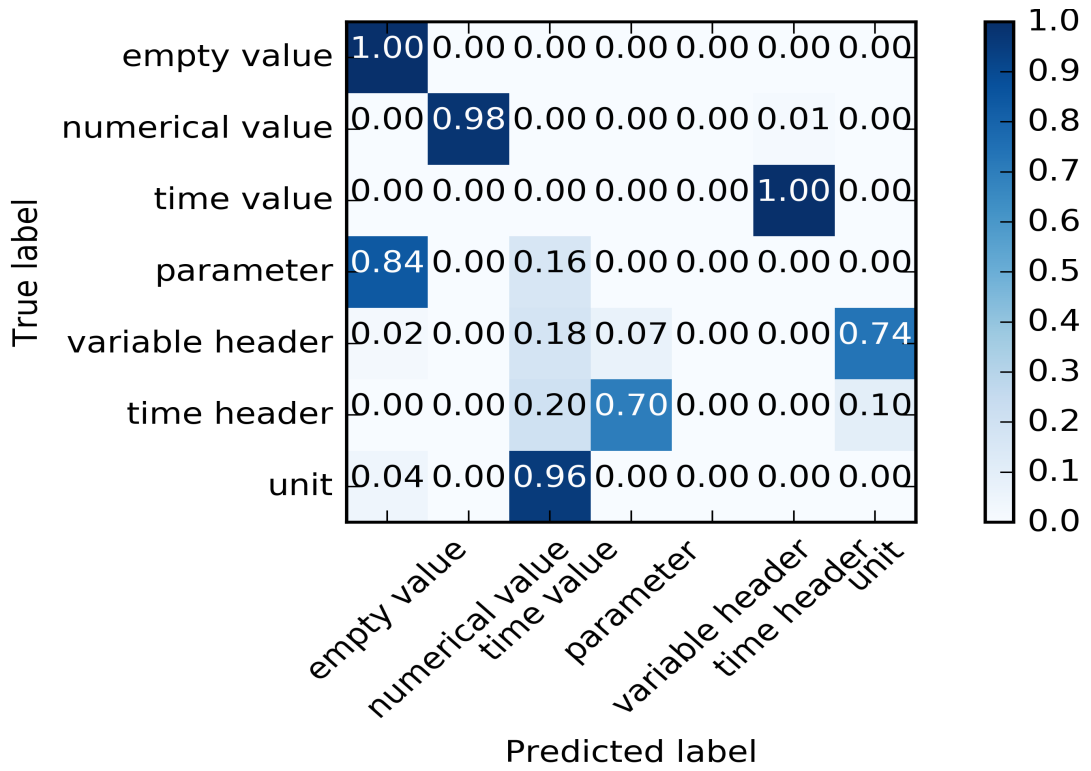


Figure 4.3: Normalized confusion matrix of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 10 %.

class	f_score	precision	recall	support
empty value	0.98	0.96	1.00	842
numerical value	0.99	1.00	0.98	932
time value	0.00	0.00	0.00	78
parameter	0.00	0.00	0.00	23
variable header	0.00	0.00	0.00	108
time header	0.00	0.00	0.00	10
unit	0.00	0.00	0.00	108

Figure 4.4: Classification report of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 10 %.

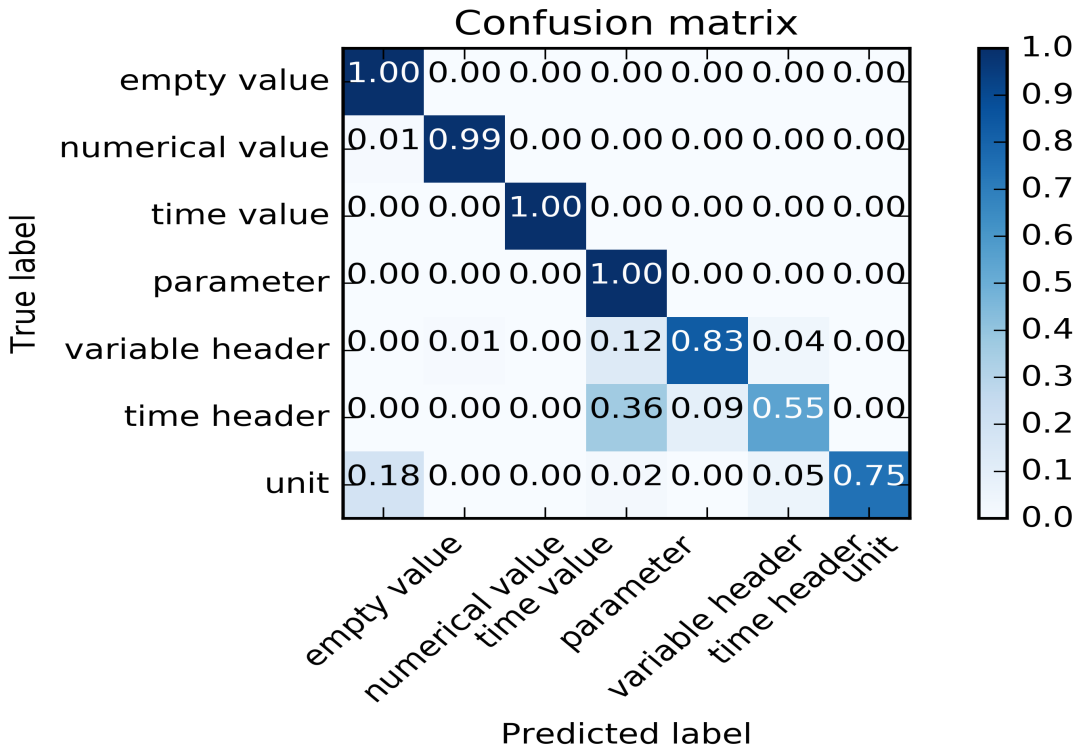


Figure 4.5: Normalized confusion matrix of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 10 %.

class	f_score	precision	recall	support
empty value	0.98	0.96	1.00	842
numerical value	0.99	1.00	0.99	932
time value	1.00	1.00	1.00	78
parameter	0.69	0.52	1.00	23
variable header	0.90	0.99	0.83	108
time header	0.41	0.33	0.55	10
unit	0.85	0.99	0.75	108

Figure 4.6: Classification report of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 10 %.

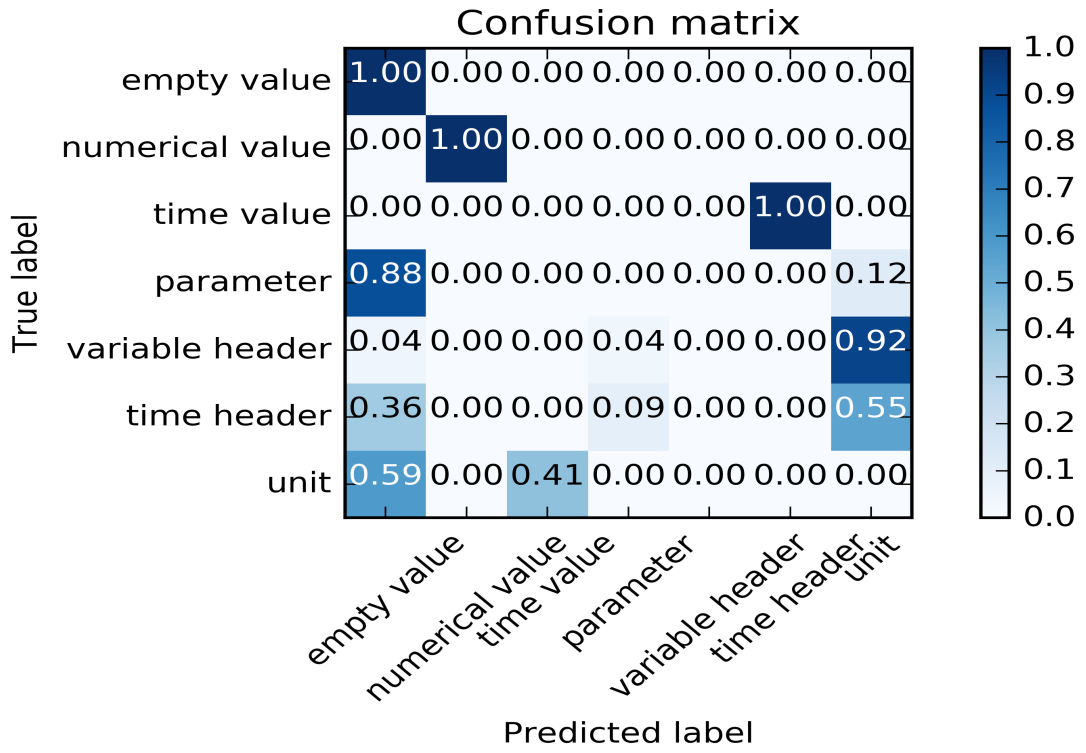


Figure 4.7: Normalized confusion matrix of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 10 %.

class	f_score	precision	recall	support
empty value	0.94	0.89	1.0	842
numerical value	1.00	1.00	1.0	932
time value	0.00	0.00	0.0	78
parameter	0.00	0.00	0.0	23
variable header	0.00	0.00	0.0	108
time header	0.00	0.00	0.0	10
unit	0.00	0.00	0.0	108

Figure 4.8: Classification report of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 10 %.

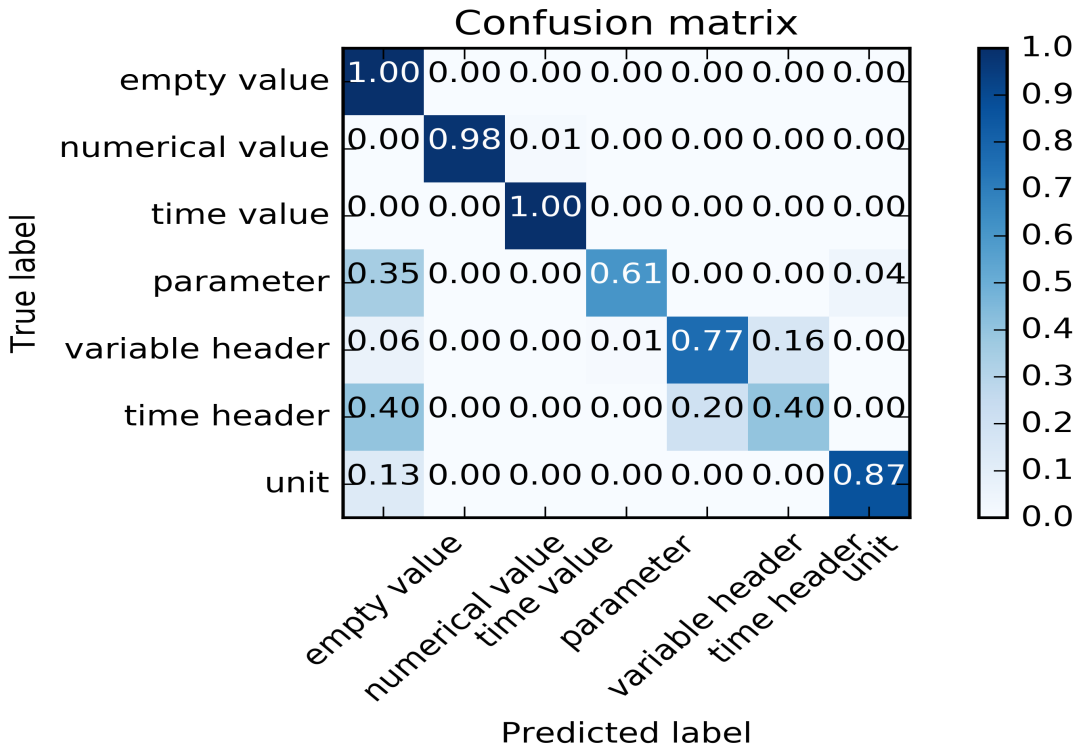


Figure 4.9: Normalized confusion matrix of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 10 %.

class	f_score	precision	recall	support
empty value	0.98	0.96	1.00	842
numerical value	0.99	1.00	0.98	932
time value	0.92	0.86	1.00	78
parameter	0.74	0.93	0.61	23
variable header	0.86	0.98	0.77	108
time header	0.26	0.19	0.40	10
unit	0.93	0.99	0.87	108

Figure 4.10: Classification report of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 10 %.

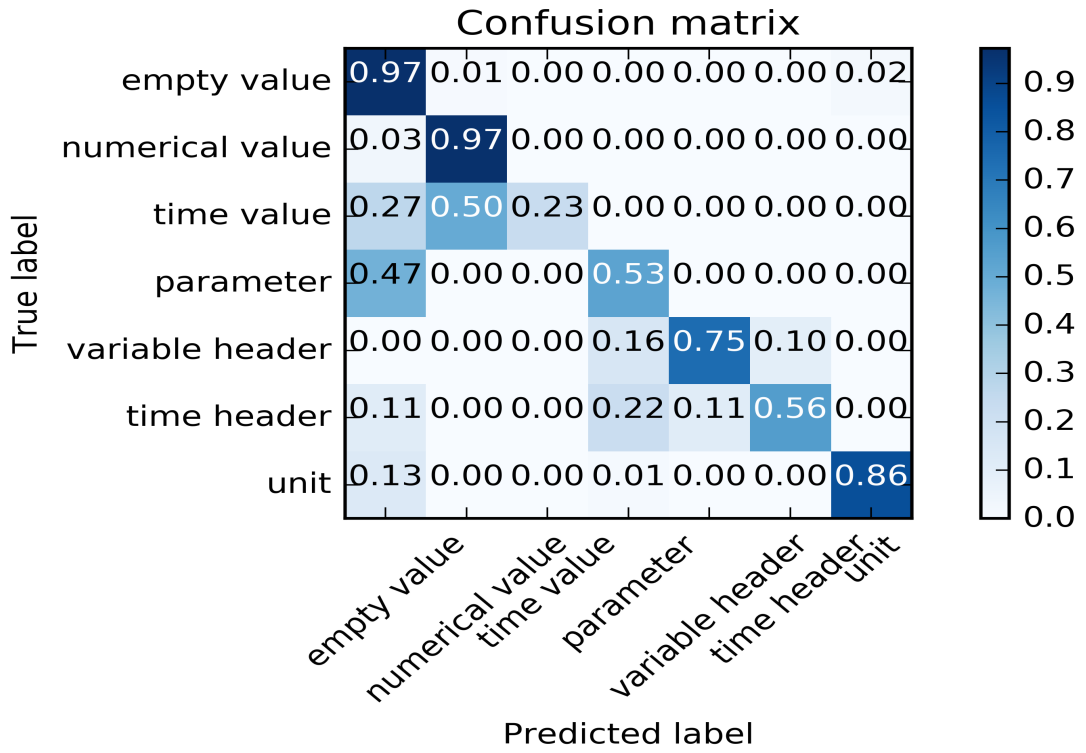


Figure 4.11: Normalized confusion matrix of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 30 %.

class	f_score	precision	recall	support
empty value	0.94	0.92	0.97	652
numerical value	0.96	0.95	0.97	726
time value	0.38	1.00	0.23	60
parameter	0.43	0.36	0.53	17
variable header	0.85	0.98	0.75	88
time header	0.45	0.38	0.56	9
unit	0.86	0.87	0.86	82

Figure 4.12: Classification report of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 30 %.

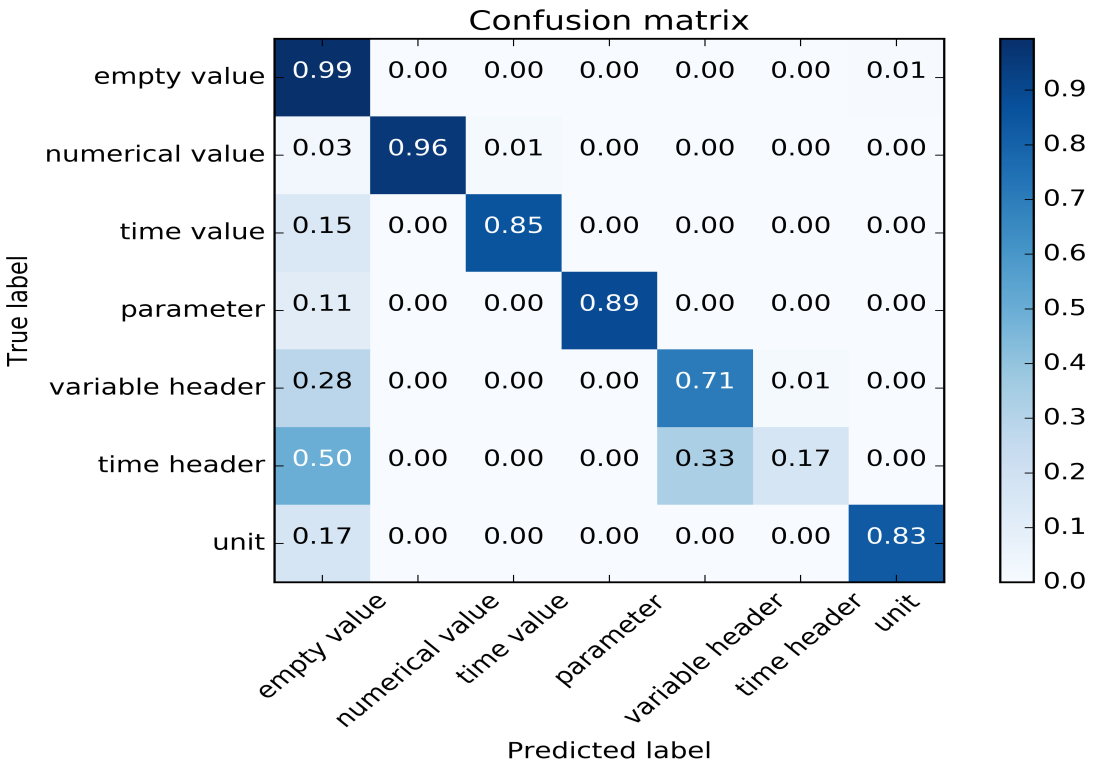


Figure 4.13: Normalized confusion matrix of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 30 %.

class	f_score	precision	recall	support
empty value	0.94	0.90	0.99	652
numerical value	0.98	1.00	0.96	726
time value	0.85	0.84	0.85	60
parameter	0.94	1.00	0.89	17
variable header	0.82	0.97	0.71	88
time header	0.25	0.50	0.17	9
unit	0.89	0.95	0.83	82

Figure 4.14: Classification report of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 30 %.

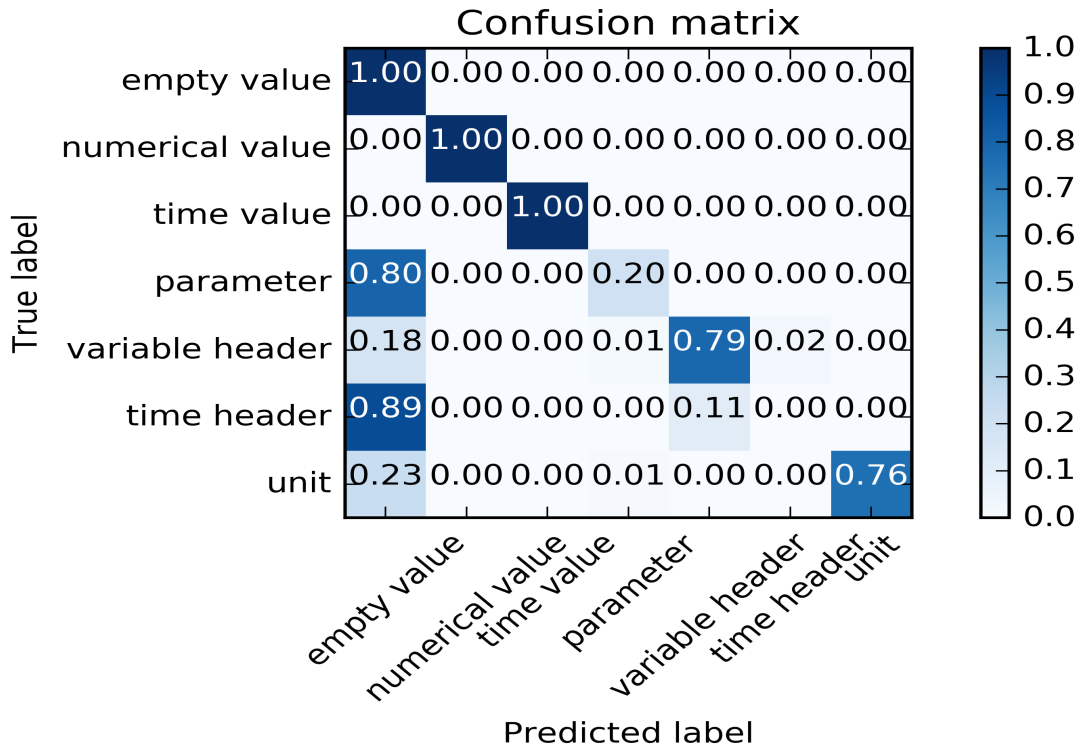


Figure 4.15: Normalized confusion matrix of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 30 %.

class	f_score	precision	recall	support
empty value	0.96	0.92	1.00	652
numerical value	1.00	1.00	1.00	726
time value	1.00	1.00	1.00	60
parameter	0.30	0.60	0.20	17
variable header	0.88	0.99	0.79	88
time header	0.00	0.00	0.00	9
unit	0.86	1.00	0.76	82

Figure 4.16: Classification report of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 30 %.

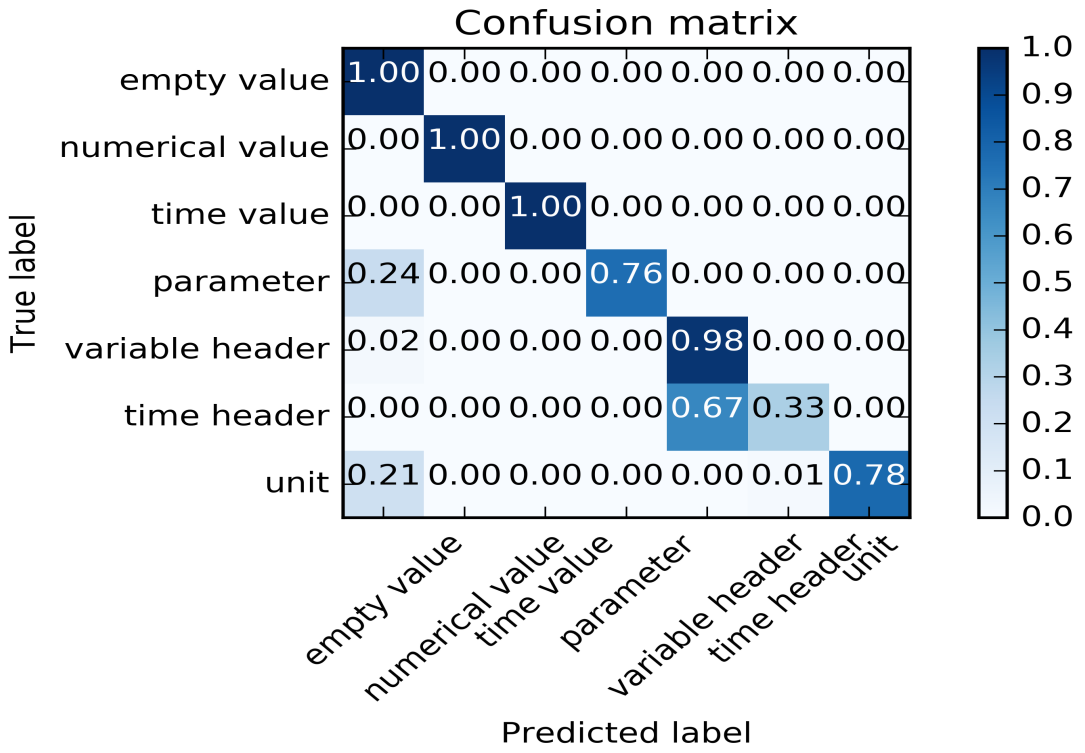


Figure 4.17: Normalized confusion matrix of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 30 %.

class	f_score	precision	recall	support
empty value	0.98	0.96	1.00	652
numerical value	1.00	1.00	1.00	726
time value	0.99	0.98	1.00	60
parameter	0.87	1.00	0.76	17
variable header	0.96	0.93	0.98	88
time header	0.46	0.75	0.33	9
unit	0.88	1.00	0.78	82

Figure 4.18: Classification report of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 30 %.

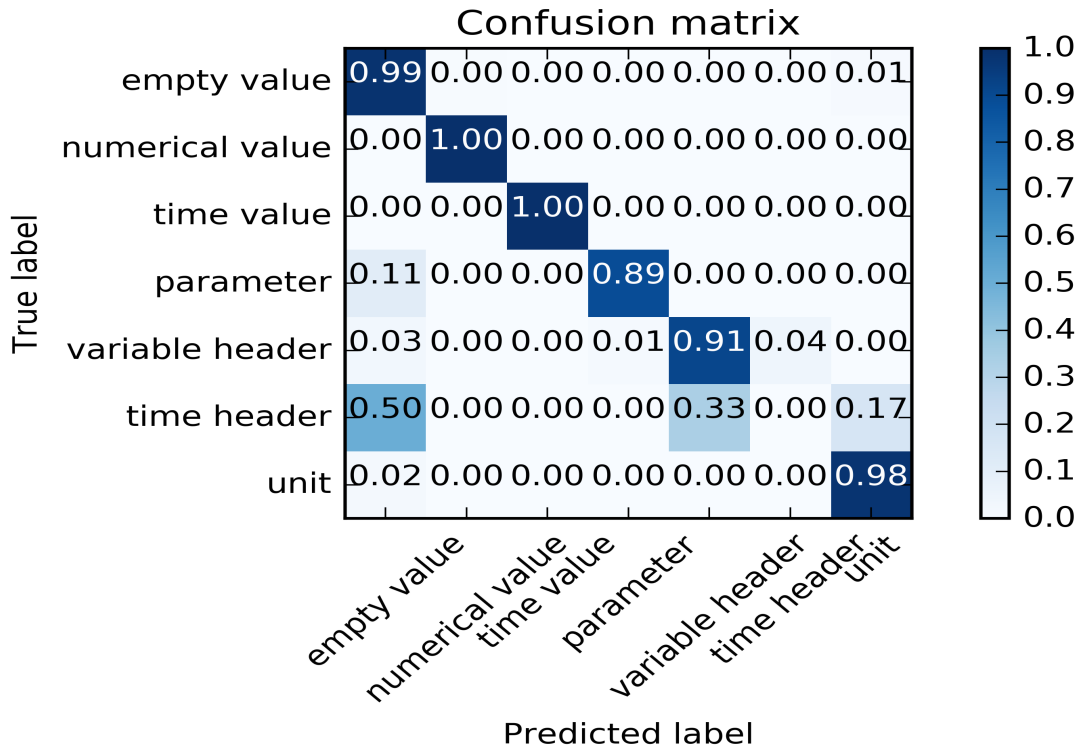


Figure 4.19: Normalized confusion matrix of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 50 %.

class	f_score	precision	recall	support
empty value	0.98	0.98	0.99	470
numerical value	1.00	1.00	1.00	518
time value	1.00	1.00	1.00	39
parameter	0.89	0.89	0.89	13
variable header	0.94	0.97	0.91	62
time header	0.00	0.00	0.00	4
unit	0.95	0.91	0.98	61

Figure 4.20: Classification report of the lowest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 50 %.

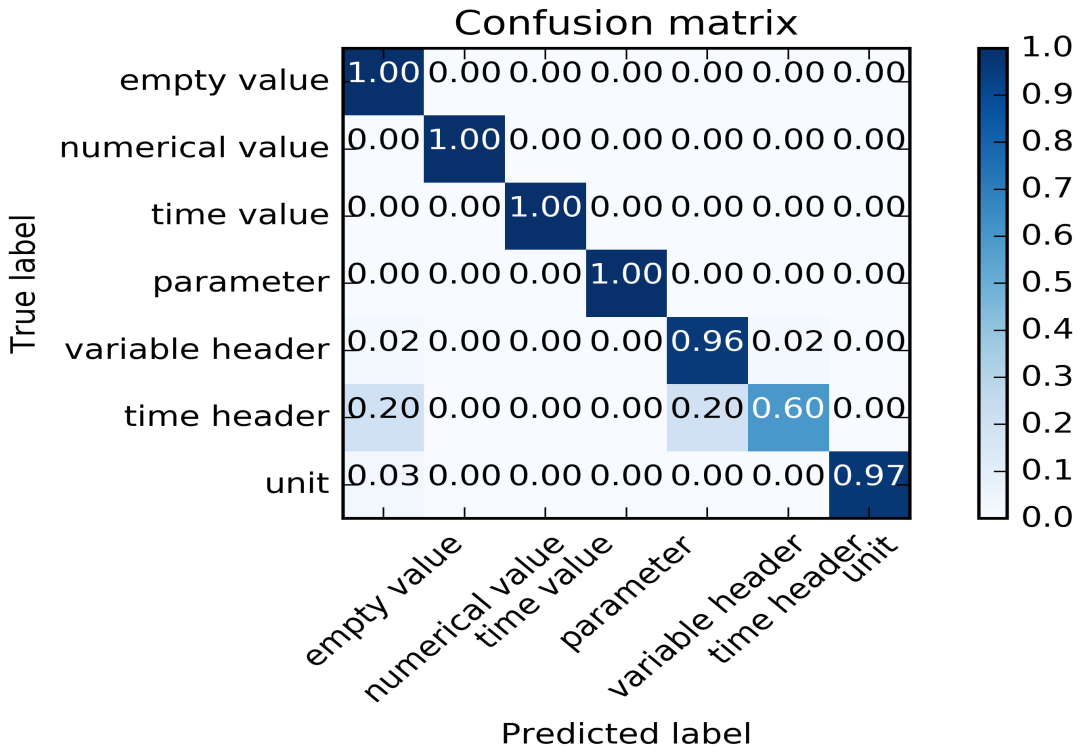


Figure 4.21: Normalized confusion matrix of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 50 %.

class	f_score	precision	recall	support
empty value	0.99	0.99	1.00	470
numerical value	1.00	1.00	1.00	518
time value	0.99	0.98	1.00	39
parameter	1.00	1.00	1.00	13
variable header	0.97	0.98	0.96	62
time header	0.67	0.75	0.60	4
unit	0.98	0.99	0.97	61

Figure 4.22: Classification report of the highest prediction result for the Support Vector Machine after 100 different runs using a training/testing data split ratio of 50 %.

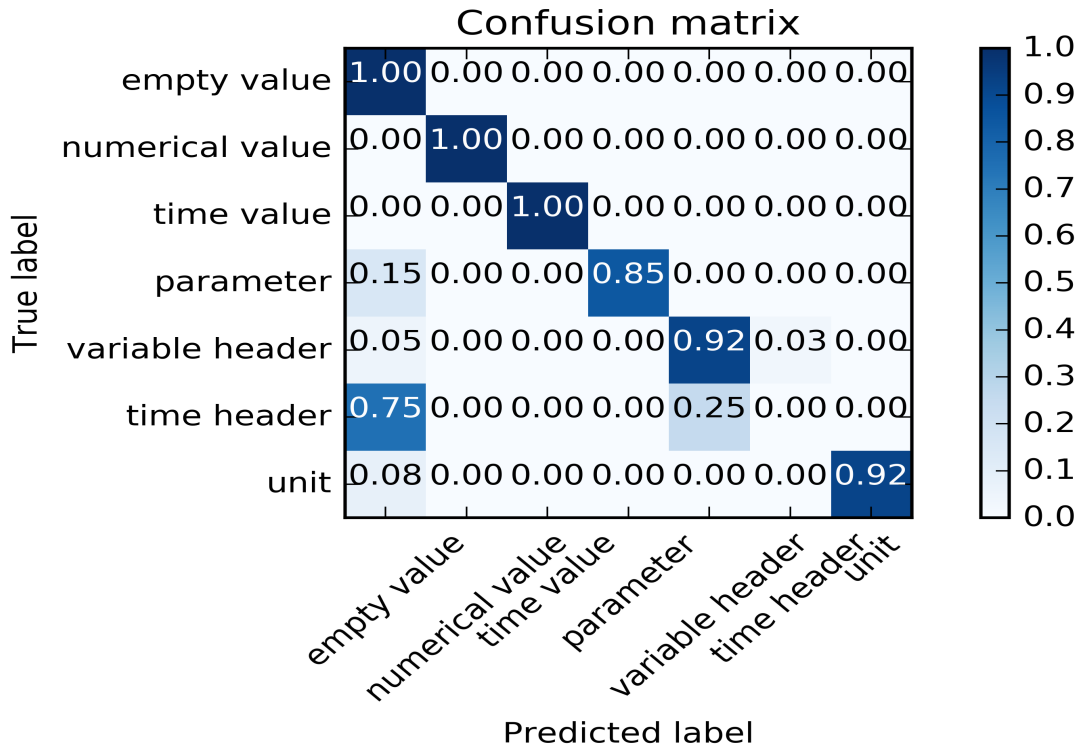


Figure 4.23: Normalized confusion matrix of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 50 %.

class	f_score	precision	recall	support
empty value	0.99	0.97	1.00	470
numerical value	1.00	1.00	1.00	518
time value	1.00	1.00	1.00	39
parameter	0.92	1.00	0.85	13
variable header	0.95	0.98	0.92	62
time header	0.00	0.00	0.00	4
unit	0.96	1.00	0.92	61

Figure 4.24: Classification report of the lowest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 50 %.

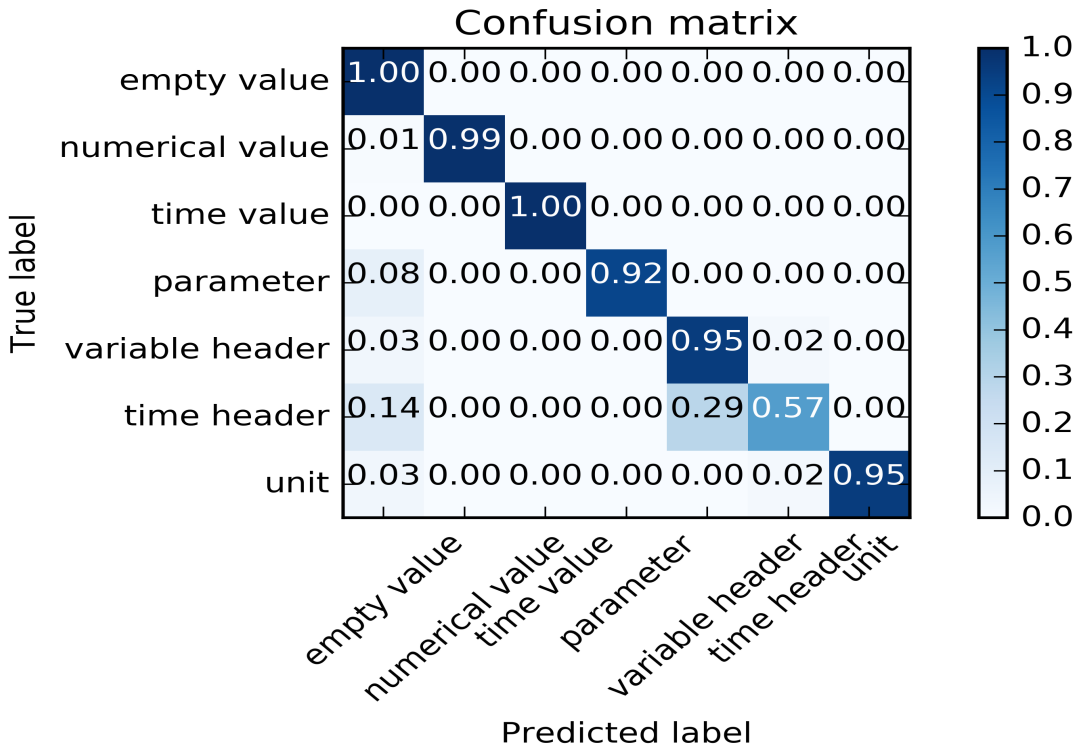


Figure 4.25: Normalized confusion matrix of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 50 %.

class	f_score	precision	recall	support
empty value	0.99	0.98	1.00	447
numerical value	1.00	1.00	0.99	536
time value	1.00	1.00	1.00	43
parameter	0.96	1.00	0.92	12
variable header	0.96	0.97	0.95	61
time header	0.62	0.67	0.57	7
unit	0.97	0.98	0.95	61

Figure 4.26: Classification report of the highest prediction result for the Gradient Boosting classifier after 100 different runs using a training/testing data split ratio of 50 %.

4.4 Prediction results for different neighborhood radii

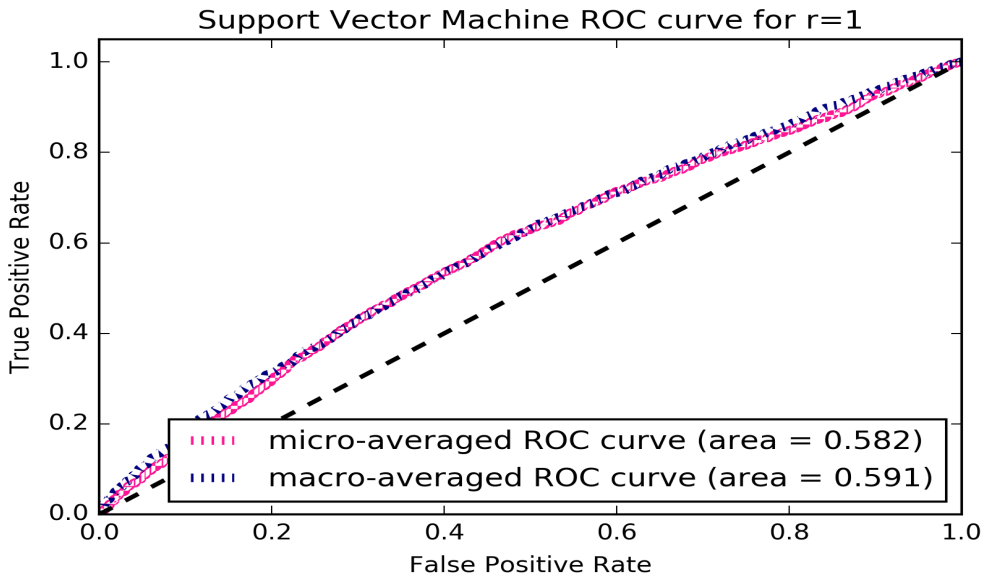


Figure 4.27: Averaged ROC curves for the Support Vector Machine using a radius of 1 and a split ratio of 0.25 for the prediction of the data set.

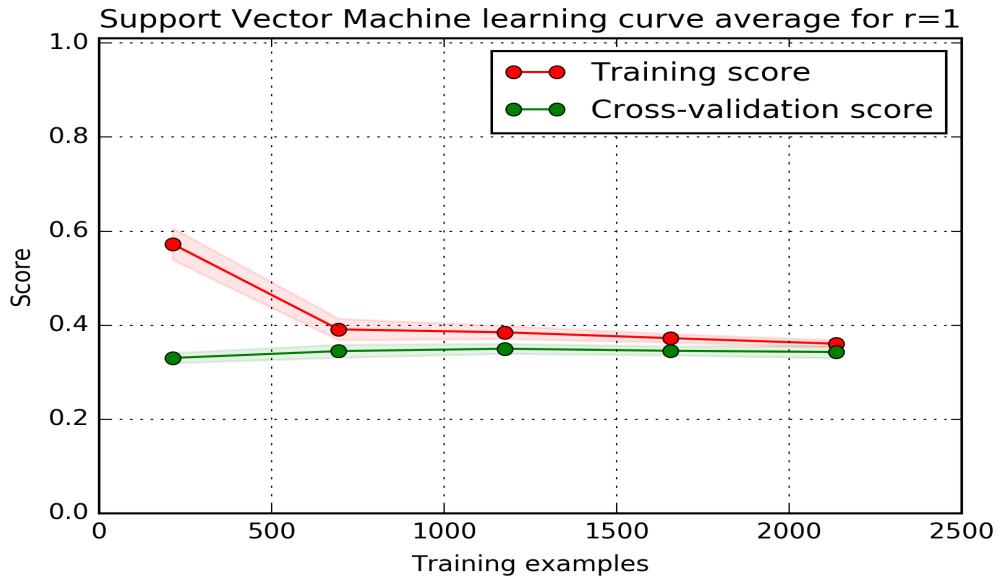


Figure 4.28: Averaged training and cross-validated prediction learning scores for the Support Vector Machine using a radius of 1 and a split ratio of 0.25.

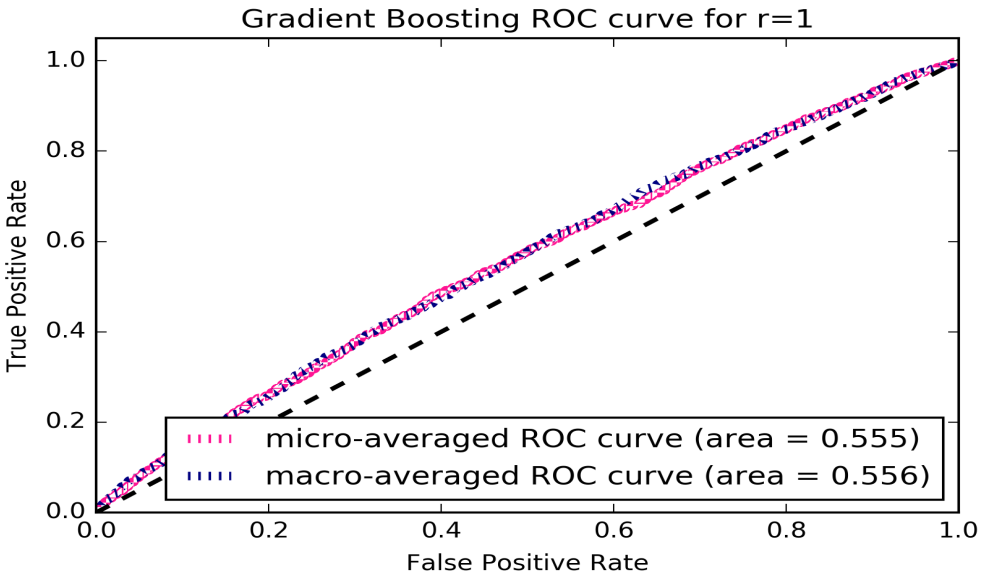


Figure 4.29: Averaged ROC curves for the Gradient Boosting classifier using a radius of 1 and a split ratio of 0.25 for the prediction of the data set.

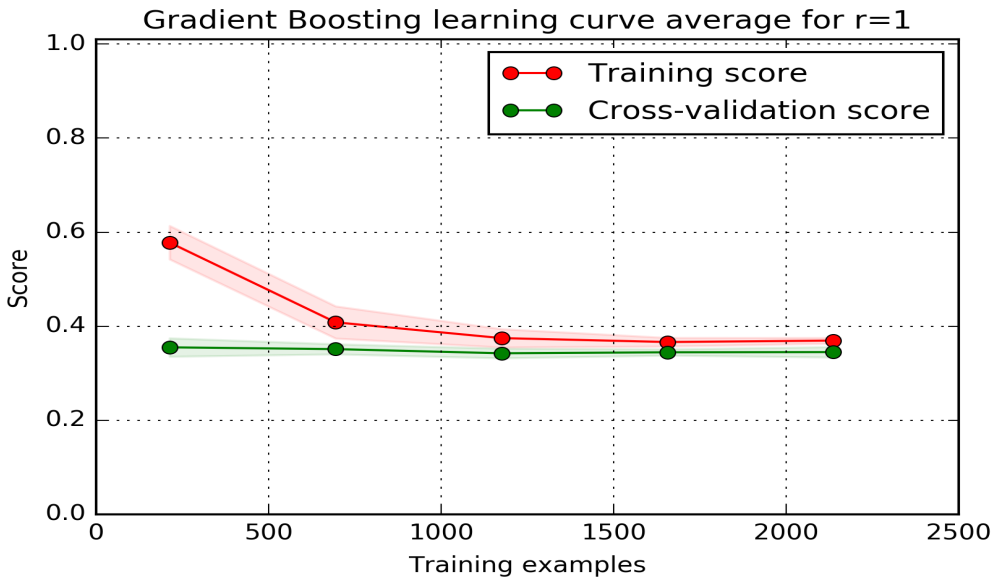


Figure 4.30: Averaged training and cross-validated prediction learning scores for the Gradient Boostint classifier using a radius of 1 and a split ratio of 0.25.

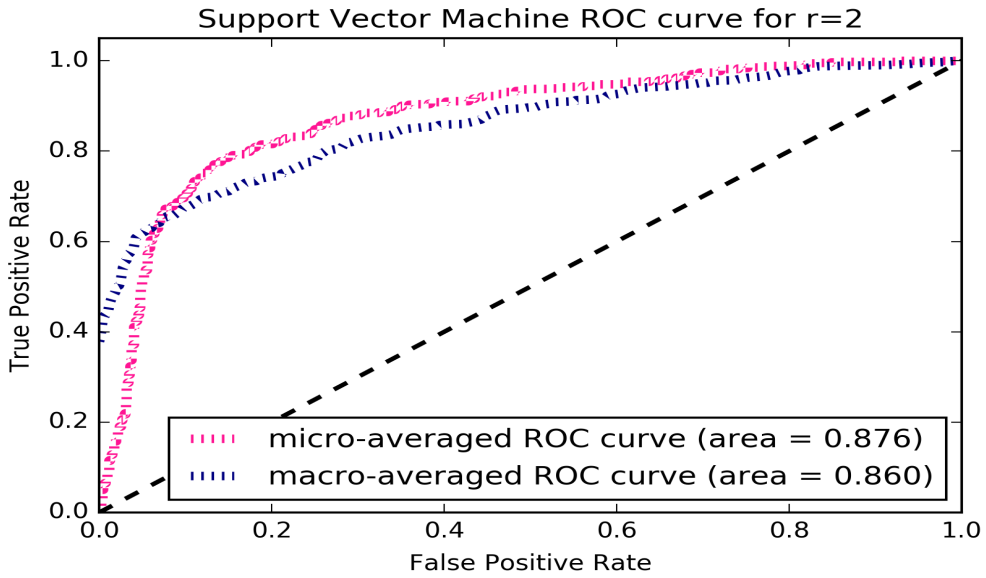


Figure 4.31: Averaged ROC curves for the Support Vector Machine using a radius of 2 and a split ratio of 0.25 for the prediction of the data set.

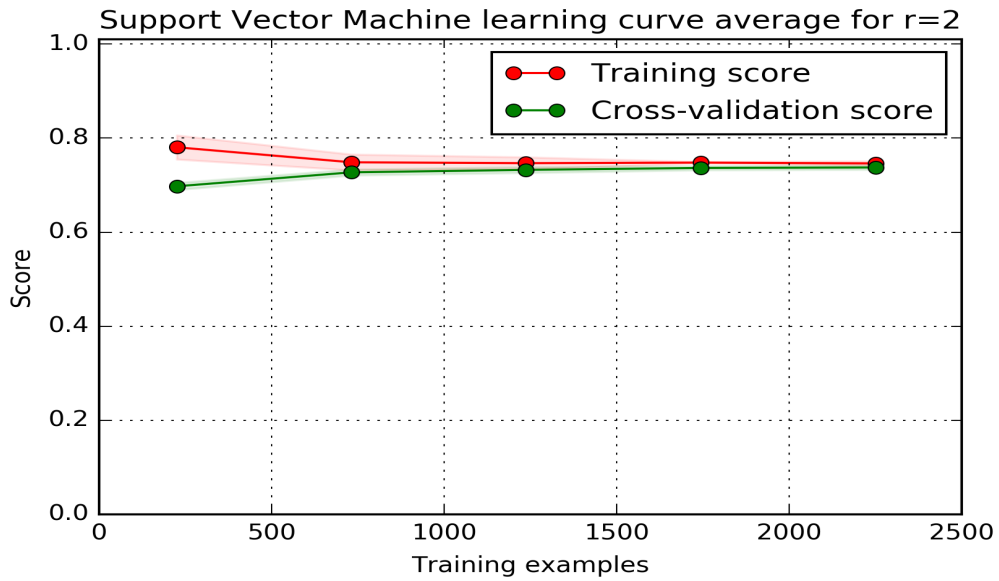


Figure 4.32: Averaged training and cross-validated prediction learning scores using a radius of 2 and a split ratio of 0.25.

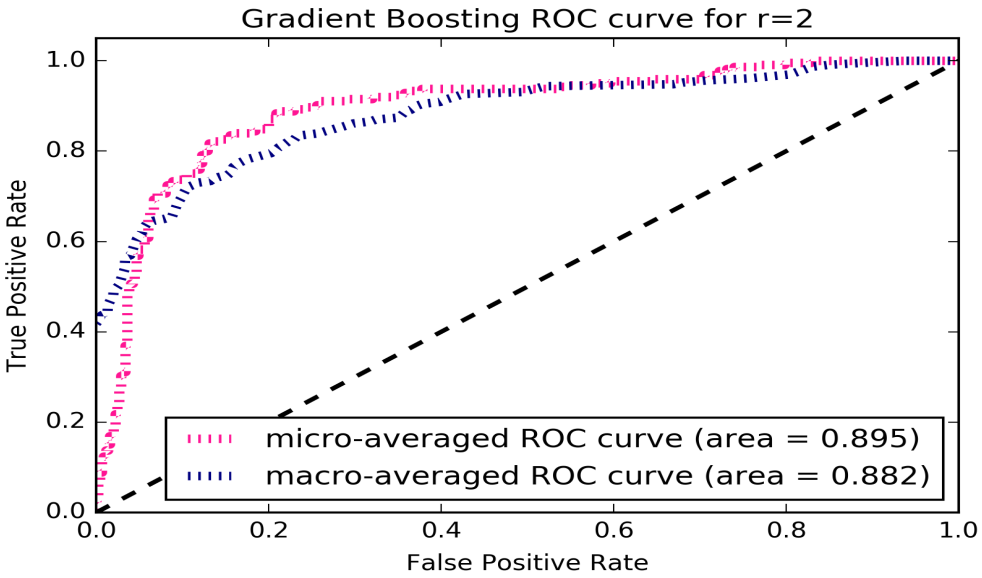


Figure 4.33: Averaged ROC curves for the Gradient Boosting classifier using a radius of 2 and a split ratio of 0.25 for the prediction of the data set.

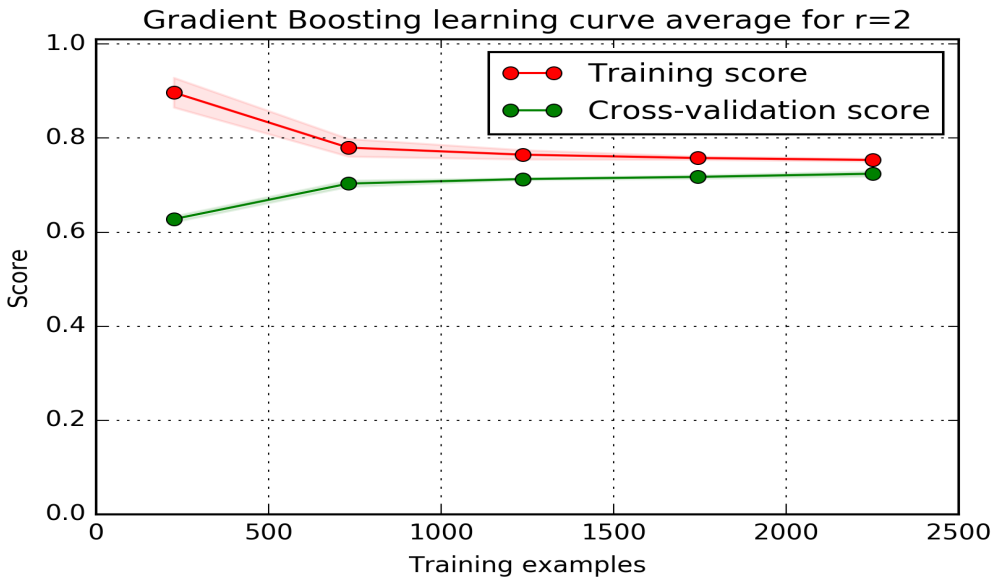


Figure 4.34: Averaged training and cross-validated prediction learning scores for the Gradient Boosting classifier using a radius of 2 and a split ratio of 0.25.

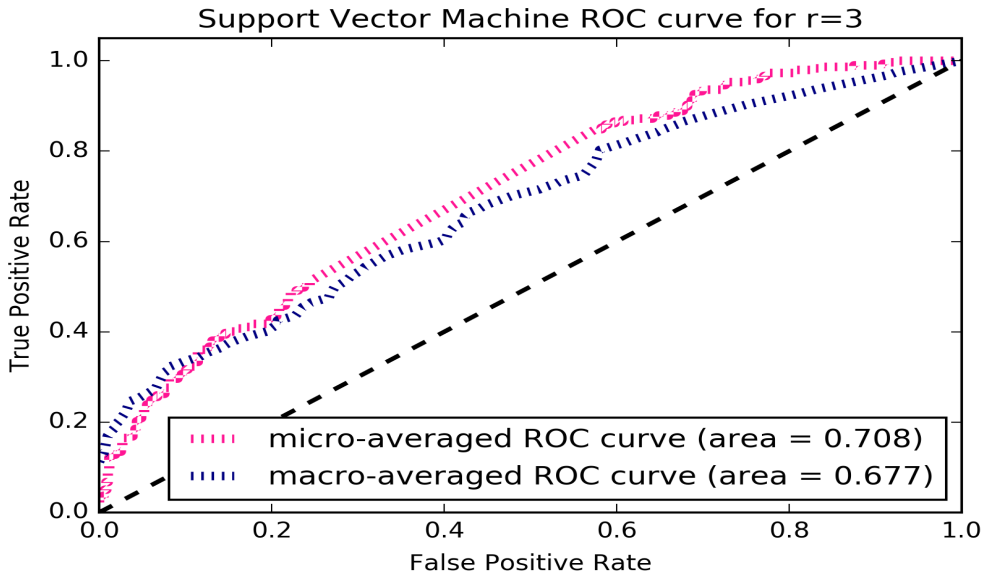


Figure 4.35: Averaged ROC curves for the Support Vector Machine using a radius of 3 and a split ratio of 0.25 for the prediction of the data set.

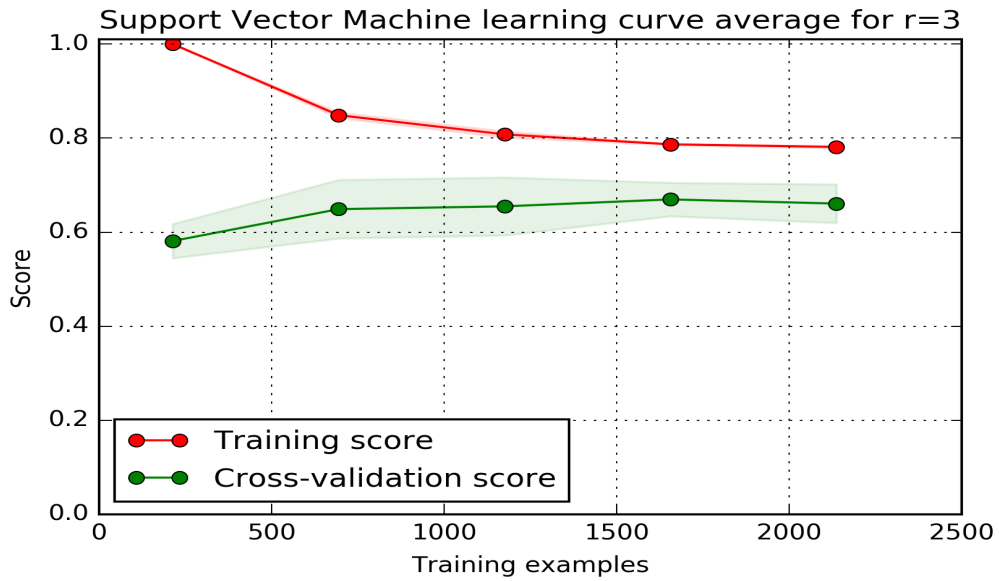


Figure 4.36: Averaged training and cross-validated prediction learning scores using a radius of 3 and a split ratio of 0.25.

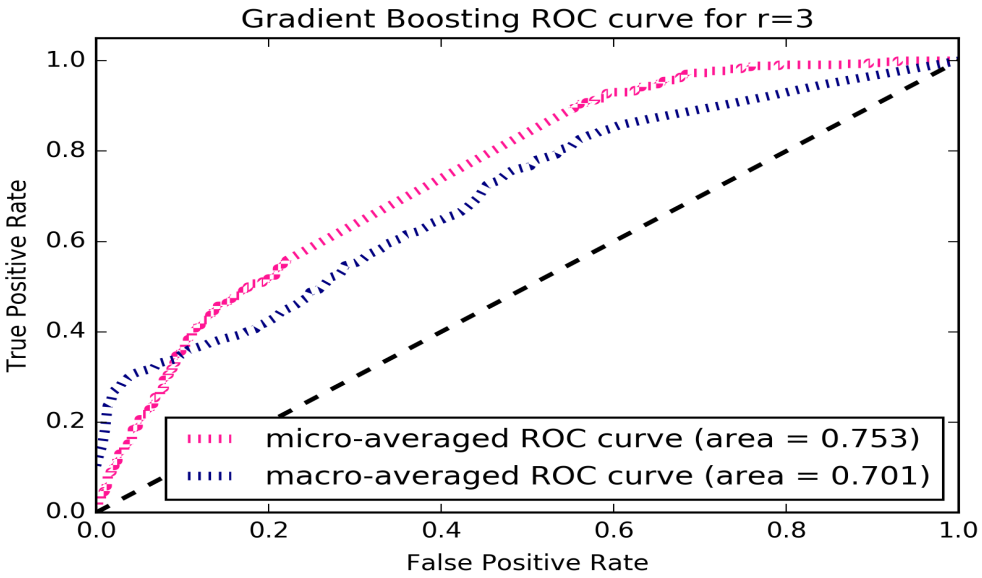


Figure 4.37: Averaged ROC curves for the Gradient Boosting classifier using a radius of 3 and a split ratio of 0.25 for the prediction of the data set.

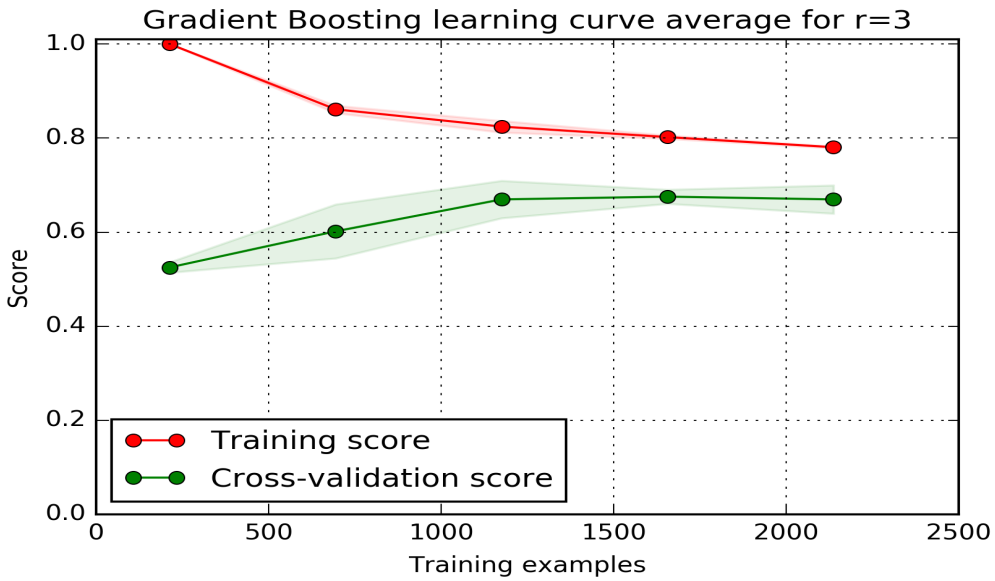


Figure 4.38: Averaged training and cross-validated prediction learning scores for the Gradient Boosting classifier using a radius of 3 and a split ratio of 0.25.

4.5 Classifier comparison

η	depth	estimators	$F_{test,avg}$	$F_{train,avg}$	$F_{train,std}$	$F_{test,std}$	ranking
0.1	1	10	0.728877	0.739840	0.005306	0.044683	27
0.1	1	50	0.821078	0.827516	0.002472	0.033478	8
0.1	1	100	0.830091	0.846798	0.005213	0.038917	2
0.1	2	10	0.805951	0.816956	0.004575	0.028032	22
0.1	2	50	0.821028	0.872385	0.007328	0.042269	9
0.1	2	100	0.825910	0.915540	0.004764	0.044989	4
0.1	3	10	0.825404	0.842885	0.004832	0.037655	5
0.1	3	50	0.827876	0.920707	0.005289	0.031066	3
0.1	3	100	0.816168	0.974813	0.005809	0.042574	13
0.2	1	10	0.778612	0.783473	0.002767	0.047110	26
0.2	1	50	0.832119	0.847677	0.005464	0.042176	1
0.2	1	100	0.817433	0.876813	0.005012	0.044340	11
0.2	2	10	0.812676	0.824155	0.003010	0.021854	17
0.2	2	50	0.822501	0.920173	0.007662	0.042477	7
0.2	2	100	0.811013	0.970183	0.003654	0.045663	19
0.2	3	10	0.820218	0.860378	0.005740	0.034861	10
0.2	3	50	0.815976	0.978333	0.004689	0.035189	14
0.2	3	100	0.814464	1.000000	0.000000	0.037569	16
0.3	1	10	0.803676	0.805208	0.007804	0.033552	25
0.3	1	50	0.823957	0.866244	0.004700	0.044302	6
0.3	1	100	0.808879	0.890701	0.005846	0.032456	20
0.3	2	10	0.817078	0.842035	0.001999	0.034965	12
0.3	2	50	0.804294	0.950916	0.007094	0.048924	24
0.3	2	100	0.804329	0.990185	0.004227	0.045144	23
0.3	3	10	0.815028	0.886372	0.009846	0.052584	15
0.3	3	50	0.812484	0.994815	0.001995	0.038288	18
0.3	3	100	0.807526	1.000000	0.000000	0.035470	21

Figure 4.39: Grid search results for the Gradient Boosting classifier using 10-fold cross validation and the averaged macro F-score metric for evaluation on the test predictions. Each row represents a subset of the parameters learning rate η, maximum leaf depth, and number of estimators for 10 different splits. For each subset of parameters, the macro-averaged training and testing F scores and the respective standard deviations are shown. The right column shows the final ranking of the parameter subsets according to the respective test scores.

η	depth	estimators	$F_{test,avg}$	$F_{train,avg}$	$F_{train,std}$	$F_{test,std}$	ranking
0.01	1	100	0.726397	0.743705	0.005224	0.041209	20
0.01	2	100	0.807748	0.813666	0.003191	0.030916	18
0.01	3	100	0.825021	0.844302	0.006925	0.040798	14
0.01	4	100	0.822545	0.896986	0.004977	0.060533	15
0.01	5	100	0.854355	0.929146	0.005036	0.058463	4
0.03	1	100	0.785174	0.798331	0.006944	0.037260	19
0.03	2	100	0.812037	0.841453	0.007277	0.032017	17
0.03	3	100	0.828470	0.894862	0.008846	0.049284	12
0.03	4	100	0.846232	0.946434	0.007196	0.054294	5
0.03	5	100	0.859284	0.977772	0.006147	0.051861	3
0.05	1	100	0.820955	0.829317	0.005213	0.036730	16
0.05	2	100	0.842658	0.890628	0.004255	0.043813	8
0.05	3	100	0.840853	0.937545	0.007336	0.047594	10
0.05	4	100	0.846020	0.972024	0.006063	0.060839	6
0.05	5	100	0.859434	0.997407	0.002062	0.045412	2
0.07	1	100	0.827907	0.839814	0.005448	0.034610	13
0.07	2	100	0.841186	0.910325	0.006696	0.040484	9
0.07	3	100	0.839156	0.959237	0.004471	0.048677	11
0.07	4	100	0.844290	0.990555	0.003925	0.051956	7
0.07	5	100	0.862734	1.000000	0.000000	0.047708	1

Figure 4.40: Grid search results for the Gradient Boosting classifier using 10-fold cross validation and the averaged macro F-score metric for evaluation on the test predictions. Each row represents a subset of the parameters learning rate η, maximum leaf depth, and number of estimators for 10 different splits. For each subset of parameters, the macro-averaged training and testing F scores and the respective standard deviations are shown. The right column shows the final ranking of the parameter subsets according to the respective test scores.

C	degree	kernel	$F_{test,avg}$	$F_{train,avg}$	$F_{train,std}$	$F_{test,std}$	ranking
0.35	2	linear	0.788309	0.818807	0.004946	0.044012	7
0.35	2	rbf	0.740166	0.872734	0.008754	0.070338	19
0.35	2	poly	0.768830	0.861666	0.007619	0.049686	10
0.35	4	linear	0.788309	0.818807	0.004946	0.044012	7
0.35	4	rbf	0.740166	0.872734	0.008754	0.070338	19
0.35	4	poly	0.737990	0.993518	0.002652	0.034766	22
0.35	6	linear	0.788309	0.818807	0.004946	0.044012	7
0.35	6	rbf	0.740166	0.872734	0.008754	0.070338	19
0.35	6	poly	0.719359	1.000000	0.000000	0.045828	25
0.70	2	linear	0.790192	0.818311	0.006743	0.042389	4
0.70	2	rbf	0.750339	0.906553	0.006202	0.051456	13
0.70	2	poly	0.761812	0.880111	0.005741	0.056906	11
0.70	4	linear	0.790192	0.818311	0.006743	0.042389	4
0.70	4	rbf	0.750339	0.906553	0.006202	0.051456	13
0.70	4	poly	0.733834	0.998333	0.000556	0.035838	23
0.70	6	linear	0.790192	0.818311	0.006743	0.042389	4
0.70	6	rbf	0.750339	0.906553	0.006202	0.051456	13
0.70	6	poly	0.719359	1.000000	0.000000	0.045828	25
1.00	2	linear	0.793316	0.820292	0.006887	0.046530	1
1.00	2	rbf	0.741039	0.924691	0.004206	0.057536	16
1.00	2	poly	0.761063	0.887378	0.004415	0.070433	12
1.00	4	linear	0.793316	0.820292	0.006887	0.046530	1
1.00	4	rbf	0.741039	0.924691	0.004206	0.057536	16
1.00	4	poly	0.723245	1.000000	0.000000	0.039594	24
1.00	6	linear	0.793316	0.820292	0.006887	0.046530	1
1.00	6	rbf	0.741039	0.924691	0.004206	0.057536	16
1.00	6	poly	0.719359	1.000000	0.000000	0.045828	25

Figure 4.41: Extended parameter grid search results for the Support Vector Machine using 10-fold cross validation and the averaged macro F-score metric for evaluation on the test predictions. Each row represents a subset of the parameters C, kernel type, and kernel degree for 10 different splits. For each subset of parameters, the macro-averaged training and testing F-scores and the respective standard deviations are shown. The right column shows the final ranking of the parameter subsets according to the respective test scores.

C	kernel	class weight	$F_{test,avg}$	$F_{train,avg}$	$F_{train,std}$	$F_{test,std}$	ranking
1	linear	balanced	0.814688	0.840291	0.004703	0.038386	1
10	linear	balanced	0.806111	0.839749	0.005764	0.041497	2
100	linear	balanced	0.804468	0.839378	0.005989	0.041943	4
1000	linear	balanced	0.804482	0.840118	0.005174	0.037526	3

Figure 4.42: Extended parameter grid search results for the Support Vector Machine using 10-fold cross validation and the averaged macro F-score metric for evaluation on the test predictions. Each row represents a subset of the parameters C, kernel type, and kernel degree for 10 different splits. For each subset of parameters, the macro-averaged training and testing F-scores and the respective standard deviations are shown. The right column shows the final ranking of the parameter subsets according to the respective test scores.

CHAPTER 5

Discussion

This chapter gives a detailed interpretation and analysis of the results shown in the previous chapter and considers the drawbacks of utilizing a machine learning based method for the automatic extraction of bioprocess data.
The feature engineering section 5.1 analyzes the effect of the chosen cell-based features on the results of the classification task and discusses the influence of the global features that are drawn from surrounding cells. Different settings for the radius distance parameter and their effect on possible overfitting and underfitting results are discussed in section 5.3. The influence of different training/testing split ratios and their respective prediction results are discussed in section 5.2. In Section 5.4 the results of the chosen classifiers and their respective sets of hyperparameters is discussed. The last two sections discuss the overall model architecture and possible applications to bioprocess data.

5.1 Feature engineering

The feature set used for the resulting predictions is built in a hierarchical manner by first constructing a set of local cell-based features and then gathering the neighboring cells to stack all surrounding information into a single feature set. Thus, in order to analyze the influence of cell-based and neighborhood features separately, the feature test results are split into two different experiments.
The first part of section 4.2 shows the averaged feature importances for the cell-based features, their respective standard deviations, and the lowest and highest obtained valued for 100 different test runs and both classifiers. The second part of section 4.2 shows the feature importances of the combined neighborhood cell-based feature sets in relation to their location relative to the center cell for both classifiers.
Table 4.1 shows the resulting feature importance statistics for the Gradient Boosting classifier over 100 different prediction runs using a training/testing split ratio of 10 %. The mean value for the feature importance of feature **data type** is 0.337 with a standard

deviation of 0.013 a lowest score of 0.312 and a highest score of 0.373. In comparison with the averaged feature importances of the remaining cell-based features, the data type feature shows the lowest relative standard deviation of 3.996%. This shows that the feature adds a consistent value over different data set splits and runs, independent of the cell type distribution of the respective split. A fraction of the standard deviation was found to be caused by wrongly parsed cell types while building up the data set.
The averaged feature importance for the feature **string header similarity** is 0.301 with a standard deviation of 0.025, a lowest score of 0.251 and a highest score of 0.3666. The highest score of the 100 runs show that the feature importance contributes almost as much as the data type feature for splits with a high proportion of header cells. The addition of domain-specific string greatly aids in the prediction process as reported by Zelikovitz et. al. in [48] and Jiang et. al. in [28]
The feature **unit similarity** has an averaged feature importance of 0.188 with a standard deviation of 0.023, a lowest score of 0.142, and a highest score of 0.233. The result for the mean feature importance is only 62 % of the header similarity and shows a relative standard deviation of 12.171%. The lower score is partially caused by special characters appearing in the header variables that are common in unit cells, a lower overall appearance of the unit label in the dataset, and parsing errors during the setup of the feature matrix.
The feature **cell length** shows an average feature importance of 0.131 with a standard deviation of 0.024, a lowest score of 0.093, and a highest score of 0.183. For cell type distributions with large numerical values the feature becomes almost redundant because of the interfering lengths of numerical values and strings, which is also shown in the high relative standard deviation of 18.644%. The feature contributes for cell type distributions with numerical values that have a digit length below the length of the string cells and thus shows a good average maximal value of 0.183.
The feature importance results for the feature **digit ratio** showed the second highest relative standard deviation of 23.527%, a lowest score of 0.017, and a highest score of 0.041. The average score of 0.027 with a standard deviation of 0.006. Due to the high imbalance between class labels, the feature becomes redundant for testing splits where the string similarities and the data type cover most of the discriminative information for the classification.
The feature importance results for the feature **center of numerical values** showed the lowest average value of 0.015 with a standard deviation of 0.006, a lowest score of 0.003, and a highest score of 0.0029. The feature calculates the normalized Manhattan distance of a cell w.r.t the coordinates of the center of numerical values. This easily results in misclassification when cells are wrongly parsed as string cells and thus receive a low distance value compared to high distance values of most string cells on a bioprocess sheet. In turn, the feature contributes information if numerical cells are parsed correctly. The feature replaced an earlier feature that calculated the normalized cell type value for each row, describing the amount of strings, bools, floats, and integers in each row by normalized type count. As it can be seen in the resulting average and standard deviation values, the feature delivered a higher overall feature importance but resulted in overfitting for cell type distributions where class labels were unevenly split.

Table 4.2 shows the feature importances for the concatenated neighborhood features in relation to the respective direction on the grid structure. The highest directional feature importance is achieved by the neighborhood features that are located below the target cell. The majority of content on the bioprocess sheet is located under the header, time, and unit cells, thus explaining the high average feature importance of 0.453 and the lowest relative standard deviation of 6%.
The average feature importance results for the left and right neighborhood direction differ only by 10% and thus contributing almost equally to the prediction results. The left neighborhood features show a much larger relative standard deviation because of the tighter boundary structure on the left side of the sheets in contrast to the right side that often has a layer of empty cells between the sheet boundary and the last data entries.
The feature importance results for the neighborhood features that are located above the target cells show the smallest contribution with an average score of 0.065 and a large relative standard deviation of 12.4%. The variable header, time, and unit cells can be detected by their location relative to the numerical and time content on the sheet structure and are thus not relying on the cell neighborhood on top of the sheet.

5.2 Training/Testing size variation

The classification reports in Figure 4.4 and Figure 4.6 show that a training/testing split ratio of 10% delivered a high variance in prediction results due to the variability of the features in different splits and runs. As it can be seen, the prediction of the class labels **empty value** and **numerical value** are independent of the random splits during the testing phase and deliver an average f-score of > 0.98 over all runs. As shown in the confusion matrices in Figure 4.3 and 4.5, the high variation in f-scores for other class labels and their lowest respective scores of 0 are due to misclassification errors between class labels with similar feature sets. For instance time and header cells, parameter and empty cells, and unit and time value cells for splits where either class is dominating during the training phase and a correct prediction cannot be inferred due to overlapping string similarities.
The corresponding classification report results in Figure 4.8 and Figure 4.10 for the Gradient Boosting classifier show a lower average f-score for the class label empty value in the lowest run and lower scores for time, variable header, and time header labels in the highest run. Since both classifiers were applied with the standard hyper-parameters, the Support Vector Machine showed a better out-of-the-box performance for a low training/test ratio of 10% as it can be seen in Figure 4.7 and 4.9. This contradicts the results in [6] where the Gradient Boosting classifier and the Support Vector Machine show an inverse behavior for data sets with imbalanced class labels. A possible reason for this behavior can be the difference in training data set during the 10-fold cross-evaluation.
As it can be seen in Figure 4.11 and 4.12, the Support Vector Machine shows much a better performance for the split ratio using 30% in the lowest run for all class labels, but overfits the data towards the majority class **empty value**. This results in a lower

recall for the class label time value and parameter. The results of the highest run in Figure 4.13 and Figure 4.14 still show a misclassification tendency towards the empty value label which is limited to the minority class **time header**, but accurate results for the leftover class labels. This is due to the fact that the standard settings of the SVM classifier internally shuffle the data at the beginning of every run and thus create a large variance in prediction results for imbalanced data sets.
The prediction scores of the lowest run for the Gradient Boosting classifier and a split ratio of 30% in Figure 4.15 and 4.16 show that the class labels with the fewest samples are completely dismissed in favor of the majority classes, due to the low amount of samples in the training/testing splits. The prediction scores of the highest run for the Gradient Boosting classifier and a split ratio of 30% in Figure 4.17 and 4.18 show highly accurate predictions for all class labels with a lowest recall value of 0.33 for the class label time header due to the small number of samples in the training/test splits.
In a final run with an even split ratio of 50% for the training and testing sets, the Support Vector Machine shows highly accurate results for the lowest run, but also completely dismisses the time header class label as it can be seen in Figure 4.19 and Figure 4.20. The highest run in Figure 4.21 and Figure 4.22 shows accurate predictions for all class labels. The split ratio of 50% shows that the variance between the lowest run and the highest run is only minimal and class imbalances can be compensated by applying resampling techniques as described in [20].
Figure 4.23 and Figure 4.24 show that the lowest run for the Gradient Boosting classifier obtains similar results, dismissing the minority clsss **unit**. The scores of the highest run in Figure 4.25 and Figure 4.26 show that the difference to the lowest run is only minimal for a split ratio of 50%.

As it can be seen from the prediction results for different splits, the variance between the lowest and highest scores constantly reduces for higher split ratios. The difference in prediction results between the two classifier is only minimal using the standard hyper-parameter settings as described in section 4.1. For the subsequent experiments and the results sections, the class imbalance is reduced by resampling the minority classes for both classifiers.

5.3 Radius variation

Figure 4.27 shows that the micro-averaged and macro-averaged ROC curve for a neighborhood radius of 1 are close to random guesses. This is the result of the limited neighborhood information on the data structure and thus resulting in a high bias. The corresponding learning curve in Figure 4.28 shows that the model underfits the data due to the lack of sufficient feature information in the training phase. The averaged ROC curve results for the Gradient Boosting classifier in Figure 4.29 shows similar results due to the underfitting of the model as it can be seen in Figure 4.30.
After increasing the gauss circle radius to 2, both models show a high specificity and sensitivity for the micro and macro-averaged ROC curves in Figure 4.31 and 4.33 due to

the increased information content of the feature set. The respective training curve validate the previous statement and show a good agreement between training and cross-validation progress as it can be seen in Figure 4.32 and 4.34.
A last increase in radius showed that the feature content overfit both models as it can be seen in the learning curve progress in Figure 4.36 for the Support Vector Machine and Figure 4.38 for the Gradient Boosting classifier. For each case, the learning curves diverge with a score difference between 0.5 and 1.5 over the validation progress, showing the resulting high variance of the trained model. The corresponding micro and macro-averaged ROC curves in Figure 4.35 and 4.37 support the high variance result of the trained model by showing decreased area results.
As a result of the ROC and learning curve analysis, a radius of 2 proved to be an accurate fit between a high bias and a high variance model for the last set of experiments using different subsets of hyper-parameters. This is in good agreement with the findings of Singh et. al. in [44], where large window sizes for the extraction of neighborhood content overfit the model with high false positive ratios and lower F1 scores.

5.4 Classifier comparison

Figure 4.39 shows that the Gradient Boosting classifier achieves a top score of 0.83 with a training rate of 0.2, 50 estimators, and a maximum estimator depth of 1. A lower learning rate results in a smaller number of estimator corrections for each additional tree in the model, but consequently also requires more trees to be added during the training process. Since we seek to achieve a high macro F-score, the performance decrease due to the increased number of trees is neglectable. From the results it can be seen that the high training score of 1 indicates an overfitted model once too many estimators are used. The number of sequential trees should thus be kept at a 100 estimators. From the results it can be seen that we can still improve on the learning rate and estimator depth of the classifier, without overfitting the model. Figure 4.40 shows a detailed grid search for the specific estimator setting of 100 and a variation of the learning rate and the maximum estimator depth. As it can be seen the model achieves even higher micro F score using a lower learning rate and a higher maximum depth for each tree. The highest ranked model run indicates an overfitted model with zero variance in the training scores. The optimal hyperparameters for the Gradient Boosting classifier are thus chosen from model rank 4, which delivers a high macro-averaged F score of 0.854 while showing a close average training score of 0.92 with a small learning rate of 0.01 and a maximum tree depth of 5. The combination of a high tree depth and a relatively small learning rate eliminate the risk of overfitting in the model, since the choice of a low learning rate allows the model to generalize well on unseen data and provides robustness. The results are consistent with the findings of Ganjisaffar et. al. in [21], Zhang et. al. in [49], and Rosset et. al. in [42] for Gradient Boosting classifiers, where the model generalizes well for decreasing learning rates and increasing number of samples.

As it can be seen in the macro-averaged F1 score results for the grid search using the

Support Vector Machine in Figure 4.41, the data set does not require an additional kernel mapping on the bioprocess data set. Since the parameter **degree** has no influence on the linear kernel mapping, the same score results for the linear kernel are shown multiple times for different settings of the parameter **degree**. As a result, the linear kernel in combination with an error weight of 1 shows the highest macro-averaged score of 0.79. Since the corresponding training score of 0.82 is still relatively low compared to the test score, we can still tweak the parameters in a more specific grid search. Furthermore, the grid search results show that the error weight parameter (see section 2.6 for a detailed description of the error weight in SVM) can still be increased without overfitting the data and thus further reducing the error rate. The results of the specific kernel mappings for the rbf and polynomial kernel are not further tested due to the high gap between the training and testing scores for the respective parameter subsets and possible overfitting. Figure 4.42 shows the specific grid search using four different values for the error weight and a balanced class weight. This results in a balanced usage of the error weight by multiplying the error term for each class label with the number of samples for the respective class and thus reducing the influence of the class imbalance in the data set. As it can be seen, the gap between the training and test score is minimized with increasing values of C, resulting in a more stable model. The final hyperparameters for the SVM are thus set to an error weight of 10, a balanced class weight, and the linear kernel. The combination of a relatively high error weight and a weighted scaling of the error term for each class ensures a reduction of the data set imbalance.
The choice of the linear kernel prevents additional overfitting of the model and further increase in model complexity for the given data set. The usage of kernel mappings results in infinite-dimensional feature spaces and thus in an equally infinite Vapnik-Chervonenkis dimension. This infinite VC-dimension does not always guarantee generalization of the learned model. Similar behavior is seen in [25, 8] where the resulting kernel mapping worsens the prediction results.
The opposite behavior can be seen in [33, 26, 23] where the number of samples for each class is much larger than the number of features and thus showing a significant improvement for kernel-based SVM results in comparison with the linear SVM.

5.5 Model architecture/novelty

State of the art model architectures and methods in mining and classification of tabular data include conditional random fields [1, 12, 41], hierarchical trees [30], and cell similarity measures [11]. These methods work well on datasets that require little to no a priori domain-knowledge and do not require additional, local neighborhood information for the classification of single cells. However, these state of the art methods fail to capture the global structure of the underlying data set due to the lack of domain-specific a priori knowledge and local neighborhood information. Nearest-neighbor feature extraction methods have been successfully applied to classification tasks across different research domains on images [37, 50], EEG signals [32], and text documents [24, 4].

A novel model architecture based on the combination of simple cell-based features, nearest neighbor methods for feature extraction, and domain-specific information in the document classification process is therefore proposed in the context of this thesis. The feature set was extracted in a hierarchical manner, where simple cell-based features like data type and string length represent the smallest building blocks as described in section 3.2. Additionally, a-priori domain knowledge is added by computing string similarities to a collection of bioprocess specific terms (see subsection 2.4). Building blocks were then concatenated into a single numerical feature. To capture the relational neighborhood information of each cell, concatenated building clocks were then extended using the kd-tree nearest neighbor structure as described in section 2.3.
As it can be seen from the results in Figure 4.27 and 4.32 for the Support Vector Machine and Figure 4.30 and 4.34 for the Gradient Boosting classifier, the additional neighborhood information from the nearest neighbor algorithm greatly boosts the prediction results until the model overfits the data. Without this information, the classifier fails to learn a meaningful structure on the given bioprocess data set and fails to predict unseen data due to poor generalization power. To the authors knowledge, this model architecture has not yet been used in the context of document/tabular data classification.

CHAPTER 6

Conclusion

In conclusion, the development of a machine-learning based extraction method for bioprocess data sheets proved to be succesful, but requires a careful choice of model hyper-parameters and feature extraction settings. Modifications in the parsing of features would greatly improve the prediction results and thus lower the amount of misclassifications due to falsely parsed cells in the training data set. These improvements include a detection algorithm for time formats, string cleaning methods to remove redundant characters for the header cells, and a refactoring of the domain-specific keyword dictionaries to avoid overlapping string matching results. These changes would be essential for the training process of the predictive model in order to learn a meaningful structure given the correct feature sets for the respective cells.

The selection of cell features showed that the features digit ratio and numerical center distance were almost completely dismissed during the training phase. A possible improvement would thus be to test the permutations of existing features and their respective feature importances over multiple runs to detect possible improvements in the classification process as described in [2]. Additionally, existing features that show no significant contribution to the classifier training after running the feature importance permutation tests can be replaced by new features. The highly unbalanced feature importances for the different directions neighborhood features indicate that cell neighborhood features strongly depend on the structure of the data set. It would thus be of interest how information content of diagonal cells can contribute to the training process.

The evaluation of different training/testing split ratios showed expected prediction results and proved to converge to a minimum variance between the lowest and the highest test run scores for sufficiently large amounts of training data. Interestingly, the Gradient Boosting classifier showed a lower performance for test cases with highly imbalanced class labels contrary to the results in [6]. Further investigation with different

sets of hyper-parameters for different training/test split rations would be of interest to analyze the class imbalance behavior for the two classifiers. The prediction results for the neighborhood radius variation indicate that the training of the classifier might profit from different neighborhood shapes during the construction of the feature set. These shapes can influence the information content for each cell while preserving the radius and reduce the risk of overfitting the model. An investigation of different neighborhood shapes and their impact on the resulting feature importance and classification result would be a logical next step to this thesis.

The grid search results for the respective classifiers showed expected scores, but still leave room for improvements of the hyper-parameter settings. Since the hyper-parameter optimization was only carried out on the most important settings due to the sheer complexity of the testing space, it would be of interest to conduct further parameter optimization tests. Due to the knowledge that Gradient Boosting machines are highly customizable [38] prediction models, it would be valuable to further explore the hyper-parameter space of the model, for instance by introducing new loss functions and customized base learners.

Appendix

Table 6.1: Batch source table by industry type for the training and testing data set.

company type	**batch**
pharmaceutical	online & offline data
pharmaceutical	online data
pharmaceutical	offline & offline data
bioreactors	online data
gas sensors	online data
biosciences	offline data

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] M. D. Adelfio and H. Samet. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment*, 6(6):421–432, 2013.

[2] A. Altmann, L. Toloşi, O. Sander, and T. Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[4] T. Basu and C. Murthy. Towards enriching the quality of k-nearest neighbor rule for document classification. *International Journal of Machine Learning and Cybernetics*, 5(6):897–905, 2014.

[5] L. Bottou and Y. L. Cun. Large scale online learning. In *Advances in neural information processing systems*, pages 217–224, 2004.

[6] I. Brown and C. Mues. An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3):3446–3453, 2012.

[7] P. Bühlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, pages 477–505, 2007.

[8] Y.-W. Chang and C.-J. Lin. Feature ranking using linear svm. In *Causation and Prediction Challenge*, pages 53–64, 2008.

[9] S. Charaniya, W.-S. Hu, and G. Karypis. Mining bioprocess data: opportunities and challenges. *Trends in biotechnology*, 26:690–699, Dec. 2008.

[10] S. Charaniya, H. Le, H. Rangwala, K. Mills, K. Johnson, G. Karypis, and W.-S. Hu. Mining manufacturing data for discovery of high productivity process characteristics. *Journal of biotechnology*, 147(3):186–197, 2010.

[11] H.-H. Chen, S.-C. Tsai, and J.-H. Tsai. Mining tables from large scale html texts. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 166–172. Association for Computational Linguistics, 2000.

[12] Z. Chen and M. Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, page 1. ACM, 2013.

[13] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Kdd workshop on data cleaning and object consolidation*, volume 3, pages 73–78, 2003.

[14] T. G. Dietterich. Ensemble learning. *The handbook of brain theory and neural networks*, 2:110–125, 2002.

[15] T. G. Dietterich et al. Ensemble methods in machine learning. *Multiple classifier systems*, 1857:1–15, 2000.

[16] K. Duan and S. S. Keerthi. Which is the best multiclass svm method? an empirical study. *Multiple classifier systems*, 3541:278–285, 2005.

[17] A. M. Ebtehaj and E. Foufoula-Georgiou. On variational downscaling, fusion, and assimilation of hydrometeorological states: A unified framework via regularization. *Water Resources Research*, 49(9):5944–5963, 2013.

[18] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[19] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[20] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.

[21] Y. Ganjisaffar, R. Caruana, and C. V. Lopes. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 85–94. ACM, 2011.

[22] A. Gibson and J. Patterson. *Deep Learning: A Practitioner's Approach*. O'Reilly, 2017.

[23] Q. Gu and J. Han. Clustered support vector machines. In C. M. Carvalho and P. Ravikumar, editors, *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, volume 31 of *Proceedings of Machine Learning Research*, pages 307–315, Scottsdale, Arizona, USA, 29 Apr–01 May 2013. PMLR.

[24] E.-H. S. Han, G. Karypis, and V. Kumar. Text categorization using weight adjusted k-nearest neighbor classification. In *Pacific-asia conference on knowledge discovery and data mining*, pages 53–65. Springer, 2001.

[25] H. Han and X. Jiang. Overcome support vector machine diagnosis overfitting. *Cancer informatics*, 13(Suppl 1):145, 2014.

[26] C.-J. Hsieh, S. Si, and I. Dhillon. A divide-and-conquer solver for kernel support vector machines. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 566–574, Bejing, China, 22–24 Jun 2014. PMLR.

[27] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. 2003.

[28] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

[29] M. Kaden, M. Lange, D. Nebel, M. Riedel, T. Geweniger, and T. Villmann. Aspects in classification learning-review of recent developments in learning vector quantization. *Foundations of Computing and Decision Sciences*, 39(2):79–105, 2014.

[30] V. Le and S. Gulwani. Flashextract: A framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014.

[31] P. Lison. "an introduction to machine learning, 2015.

[32] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4(2):R1, 2007.

[33] S. Maji, A. C. Berg, and J. Malik. Efficient classification for additive kernel svms. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):66–77, 2013.

[34] A. M. Martínez and A. C. Kak. Pca versus lda. *IEEE transactions on pattern analysis and machine intelligence*, 23(2):228–233, 2001.

[35] C. McDonald. Machine learning: a survey of current techniques. *Artificial Intelligence Review*, 3(4):243–280, 1989.

[36] C. Z. Mooney and R. D. Duval. *Bootstrapping: A nonparametric approach to statistical inference*. Number 94-95. Sage, 1993.

[37] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[38] A. Natekin and A. Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7, 2013.

[39] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen. UFLDL Tutorial. *http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial*, 2010.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242. ACM, 2003.

[42] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 5(Aug):941–973, 2004.

[43] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 2011.

[44] R. Singh, B. Livshits, and B. Zorn. Using neural networks to find spreadsheet errors. *Microsoft Tech Report Number MSR-TR-2017-5*.

[45] R. Tedrake, T. W. Zhang, and H. S. Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2849–2854. IEEE, 2004.

[46] S. Wang, W. Chaovalitwongse, and R. Babuska. Machine learning algorithms in bipedal robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(5):728–743, 2012.

[47] Wikipedia, the free encyclopedia. Illustration of gradient descent on a series of level sets. `https://commons.wikimedia.org/wiki/File:Gradient_descent.svg`, 2012. [Online; accessed September 14, 2017].

[48] S. Zelikovitz and H. Hirsh. Improving short text classification using unlabeled background knowledge to assess document similarity. In *Proceedings of the seventeenth international conference on machine learning*, volume 2000, pages 1183–1190, 2000.

[49] Y. Zhang and A. Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015.

[50] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014.