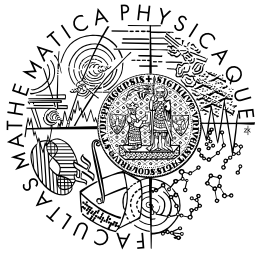

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Tomáš Krupka

SVM Classifiers and Heuristics for Feature Selection

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis: RNDr. Martin Holub, Ph.D.

Study programme: Mathematics

Study branch: General Mathematics

Prague 2016

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague 25. 7. 2016

Title: SVM Classifiers and Heuristics for Feature Selection

Author: Tomáš Krupka

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Martin Holub, Ph.D., Institute of Formal and Applied Linguistics

Abstract: In machine learning applications with a large number of computer-generated features, a selection of just a subset of features is often desirable. The Recursive Feature Elimination (SVM-RFE) algorithm proposed by Guyon et al. (2002) employs the mechanism of selecting the features based on their contribution to an SVM model decision rule, and has proven a state-of-the-art performance on the Gene Selection for Cancer Classification task (Tan et al. (2010)). This thesis expands on that work, and proposes a novel modification of the SVM-RFE feature selection method called Evaluation-Based RFE (EB-RFE). This heuristic significantly improves the performance of the SVM classifier in comparison to the original SVM-RFE on the studied machine learning task. In addition to the performance gain, the proposed algorithm has also, in experimental use, proven to have two other desirable properties. Firstly, EB-RFE produces much smaller feature subsets than SVM-RFE, which leads to more compact models. Secondly, unlike SVM-RFE, the EB-RFE heuristic is easily scalable with the computational time well beyond the possibilities of current high-end consumer CPUs.

Keywords: Machine Learning, Feature Selection, SVM, Recursive Feature Elimination

I thank my supervisor RNDr. Martin Holub, PhD. for introducing me to the interesting field of machine learning, for the inspiration during the experiments conducted for this work and for sharing his insight and intuition.

Contents

Introduction

Thanks to the availability of computational power, it is possible to train machine learning models with a large number of automatically generated features, in order to improve their performance. However, the performance of a model is usually greatly dependent on the particular feature subset that the model is trained on. Furthermore, a large number of features rules out the possibility of exhaustive enumeration search for the optimal feature subset, emphasizing the role of the feature selection heuristics.

This thesis studies the possibility of selecting the features using the machine learning algorithm known as the Support Vector Machines (SVM).

Chapter 1 provides a rigorous mathematical description of the SVM.

Chapter 2 analyses two feature selection algorithms, namely the Recursive Feature Elimination (SVM-RFE) published by Guyon et al. (2002) and the Bit-Flip/Attribute-Flip heuristics by Samb et al. (2012). Their inherent characteristics, limitations and performances are discussed and measured. The performance is measured using an experimental `R` implementation that includes the SVM kernel, parameters and class weights tuning, which are necessary for the SVM to perform well.

In Chapter 3, a novel modification of the SVM-RFE algorithm called Evaluation-Based SVM-RFE (EB-RFE) is proposed. Three of the main properties of EB-RFE are discussed in this chapter.

Firstly, this algorithm has experimentally proven to perform significantly better than the original SVM-RFE.

Secondly, it is demonstrated that the EB-RFE heuristic is able to produce much smaller feature subsets, which leads to a sparse representation of the resulting SVM classifier. This is a positive quality, since, for several reasons, sparsity with respect to input features is a desirable property of a classifier as well as its accuracy (Tan et al. (2010)).

Thirdly, unlike the SVM-RFE heuristic, EB-RFE can be tuned using a single parameter, which provides the possibility to scale the runtime according to the available computational power. This way, SVM models that are more sparse and yet perform better can be obtained in exchange for greater computational demands.

All results were obtained using the VPS-30-En dataset, which was used by Holub et al. (2012) for the task of lexical verb sense disambiguation. The results of that study are also used for performance comparison.

Particular care was taken to demonstrate the statistical significance of the results presented in this thesis.

Chapter 1

Support Vector Machines

The *Support Vector Machines* (SVM) are a well-performing, off-the-shelf method of supervised machine learning. The first idea was published by Boser et al. (1992). Today, the SVM is being widely used and researched. Its current iteration, the *soft margin* SVM, was first published by Cortes and Vapnik (1995).

This chapter is dedicated to the derivation of the SVM algorithm, to the justification of its correctness and existence, and to the discussion of its performance.

There are various ways of the algorithm derivation (e.g. introducing the duality either using the Lagrange theory or by exploiting the duality in the Rosenblatt's Perceptron). Similarly, there are various interpretations of the same terms, and this chapter refers to sources using different notation. One of the goals of this chapter is therefore to maintain consistency throughout the SVM theory explanation.

This rigorous description of the SVM will later be useful for the theory of the SVM-based feature selection.

1.1 Intuition about SVM

Definition 1. *Let n, $N \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathbb{R}$, $y \in \{-1, 1\}$. An* input example *is defined as an $(n+1)$-dimensional vector $\boldsymbol{x} = (x_1, \ldots, x_n, y)$. Its last component is called a* label. *A set of* N *input examples will be referred to as a* labelled input dataset *of length* N. *A dataset of examples without the label component is called* unlabelled. *The quantities $x_1, \ldots, x_n$ are referred to as* attributes.

Definition 2 (Linear machine)**.** *Let $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top$ be an unlabelled input dataset. Let $\boldsymbol{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$. A* linear learning machine *is defined as a real function*

$$(\boldsymbol{w}, b) \coloneqq f : \mathbb{R}^n \to \mathbb{R}$$
$$\boldsymbol{x} \mapsto \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = \sum_{i=1}^{n} w_i x_i + b,$$

where $\langle \cdot \rangle$ denotes an inner product of two vectors.

Note. Every fixed $c \in \mathbb{R}$ defines a hyperplane $\mathcal{H} = \{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) = c\}$. All such hyperplanes are mutually parallel.

Definition 3 (Linear classifier)**.** *Such linear machine f that satisfies $f(\mathbb{R}^n) = L$, where $L = \{l_1, \ldots, l_k\}$ is a finite set of real labels, is called a* linear classifier*.*

For simplicity purposes, only the binary classifiers*, with a label set $L = \{-1, 1\}$, are considered. By definition, the binary classifier f labels an unlabelled input example $\boldsymbol{x}$ with a label $y = \operatorname{sgn}(f(\boldsymbol{x}))$. By convention* $\operatorname{sgn}(0) = 1$*. The function $h(\boldsymbol{x}) = y$ is called a* hypothesis function*.*

Definition 4. *For a given labelled dataset and a linear classifier f, an input vector $(\boldsymbol{x}, y)$ is called* correctly classified*, if* $\operatorname{sgn}(h(\boldsymbol{x})) = y$*.*

Definition 5. *A hyperplane $\{\boldsymbol{x} : f(\boldsymbol{x}) = \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = 0\}$ is referred to as a* separating hyperplane $(\boldsymbol{w}, b)$*. A labelled training set $\boldsymbol{X}$ is said to be* linearly separable*, if there exists a separating hyperplane $(\boldsymbol{w}, b)$, such that the linear classifier $(\boldsymbol{w}, b)$ correctly classifies all the input examples.*

Note. The notation $(\boldsymbol{w}, b)$ can either refer to the vector itself, or both to a separating hyperplane induced by this vector, or to the linear classifier, which classifies using this hyperplane.

The SVM is a binary linear classifier operatig in a kernel-induced feature space. It develops around a few main ideas applied together:

- Finding a separating hyperplane as correct and as confident as possible. SVM is so called *maximal margin classifier*.
- Mapping the input space into a *feature space*, which is usualy high-dimensional and therefore suitable for linear separation.
- Transforming the parameters optimisation problem into its Lagrange dual, reducing significantly the computational complexity of the classification of new inputs.

1.2 Maximal Margin Classifier

The SVM is a linear classifier – that means it needs a rule for finding the $(\boldsymbol{w}, b)$ vector it uses for classification. The maximal margin classifier separates the input space with such separating hyperplane, that, firstly, is correct, and secondly, has the maximum possible distance to the nearest training examples on both sides.

The presumption of correctness is rather strong and later will be altered by adding slack variables, but for now, assume that $\boldsymbol{X} = \{(\boldsymbol{x}_1, y_1) \ldots (\boldsymbol{x}_N, y_N)\}$ is a separable training set.

The following definition enables the derivation of the optimisation problem that defines the maximum margin classifier:

Definition 6. *A* functional margin *of an input example $(\boldsymbol{x}_i, y_i) = (x_{i1}, \ldots, x_{in}, y_i)$ with respect to $(\boldsymbol{w}, b)$ is defined as*

$$\hat{\gamma}_i = y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b).$$

A functional margin of an example with respect to a normalised hyperplane $\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}, \frac{b}{\|\boldsymbol{w}\|}\right)$ *is then called a* geometric margin*:*

$$\gamma_i = y_i \left(\left\langle \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \cdot \boldsymbol{x}_i \right\rangle + \frac{b}{\|\boldsymbol{w}\|} \right).$$

Finally, we define the functional (geometric) margin *of a hyperplane* $(\boldsymbol{w}, b)$ *with respect to a dataset* $\boldsymbol{X}$ *as* $\hat{\gamma} = \min\{\hat{\gamma}_i : \boldsymbol{x}_i \in \boldsymbol{X}\}$ $(\gamma = \min\{\gamma_i : \boldsymbol{x}_i \in \boldsymbol{X}\})$.

Note. The functional margin of a training example $\boldsymbol{x}$ with respect to $(\boldsymbol{w}, b)$ being equal to 1 is equivalent to $|\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b| = 1$.

Under the presumption of the dataset $\boldsymbol{X}$ being linearly separable, the task to find the separating hyperplane that maximises the geometric margin of the dataset $\boldsymbol{X}$ is equivalent to solving the following optimisation problem:

$$\begin{aligned} &\text{maximise}_{\boldsymbol{w},b,\gamma} && \gamma \\ &\text{subject to} && y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq \gamma \\ & && i = 1, \ldots, N, \\ & && \|\boldsymbol{w}\| = 1. \end{aligned}$$

Note. The necessity of the norm of the weight vector beihg fixed is caused by the fact, that rescaling the separating hyperplane by an arbitrary constant λ does not change the geometric margin, i.e. the optimisation objective. However, note that this transformation results in the functional margin being multiplied by the same constant λ.

Equivalently, since $\gamma = \frac{\hat{\gamma}}{\|\boldsymbol{w}\|}$, the problem is feasible to a transformation that lacks the non-convex constraint $\|\boldsymbol{w}\| = 1$:

$$\begin{aligned} &\text{maximise}_{\boldsymbol{w},b,\gamma} && \frac{\hat{\gamma}}{\|\boldsymbol{w}\|} \\ &\text{subject to} && y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq \hat{\gamma} \\ & && i = 1, \ldots, N. \end{aligned}$$

Another desirable step would be transforming the objective function to a convex form. A way to achive this is to introduce a scaling condition that the functional margin be equal to 1:

$$\hat{\gamma} = 1.$$

Setting $\hat{\gamma} = 1$ is equivalent to multiplying $\boldsymbol{w}$ and b by $\frac{1}{\hat{\gamma}}$, which is a known constant for the dataset and the separating hyperplane. Also, maximizing $\frac{\hat{\gamma}}{\|\boldsymbol{w}\|} = \frac{1}{\|w\|}$ is equivalent to minimizing $\|\boldsymbol{w}\|$, or, equivalently, $\frac{1}{2}\|\boldsymbol{w}\|^2$.

The resulting optimisation problem takes the following form:

$$\begin{aligned} &\text{minimize}_{\boldsymbol{w},b,\gamma} && \frac{1}{2}\|\boldsymbol{w}\|^2 \\ &\text{subject to} && y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq 1 \\ & && i = 1, \ldots, N. \end{aligned} \tag{1.1}$$

This is an optimisation problem with quadratic objective and linear constraints, therefore it is guaranteed to have a solution – which can be found algorithmically. However, it is favourable to modify the problem to its Lagrange dual.

1.3 Lagrangian Transformation

For the purpose of finding the dual problem to the primal 1.1, let's rewrite the constraints as

$$g_i(\boldsymbol{w}) \coloneqq -y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) + 1 \leq 0, i = 1, \ldots, N, \tag{1.2}$$

and let's denote the primal and dual objective functions as $\mathcal{P}(\boldsymbol{w})$, and $\mathcal{D}(\boldsymbol{\alpha})$ resp. The primal objective satisfies $\mathcal{P}(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$, and the dual $\mathcal{D}(\boldsymbol{\alpha})$ is to be found using the Lagrange transformation of the primal.

The Lagrangian of the original problem is then as follows:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1], \tag{1.3}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$ is an N-dimensional vector of Lagrange multipliers.

The primal objective function $\mathcal{P}(\boldsymbol{w})$, as well as the constraints $g_i(\boldsymbol{w})$ (1.2), are convex in $\boldsymbol{w}$. Also $\{\boldsymbol{w} : g_i(\boldsymbol{w}) \leq 0, i = 1, \ldots, N\} \neq \emptyset$, thanks to the initial assumption that the dataset $\boldsymbol{X}$ be separable.

Therefore, there exists a certain vector $(\boldsymbol{w}^*, b^*, \boldsymbol{\alpha}^*)$, which satisfies the equality $\mathcal{P}(\boldsymbol{w}^*) = \mathcal{L}(\boldsymbol{w}^*, b^*, \boldsymbol{\alpha}^*) = \mathcal{D}(\boldsymbol{\alpha}^*)$. Furthermore, $(\boldsymbol{w}^*, b^*, \boldsymbol{\alpha}^*)$ solves both the primal and the dual problem and the *Karush-Kuhn-Tucker* conditions (KKT) hold as follows:

1. $(\boldsymbol{w}^*, b^*, \boldsymbol{\alpha}^*)$ is the saddle point of $\mathcal{L}$, i.e. $\nabla_{(\boldsymbol{w},b)}\mathcal{L}(\boldsymbol{w}^*, b^*, \boldsymbol{\alpha}^*) = \mathbf{0}$
2. $\alpha_i^* g_i(\boldsymbol{w}^*) = 0, i = 1, \ldots, N$
3. $g_i(\boldsymbol{w}^*) \leq 0, i = 1, \ldots, N$
4. $\alpha_i^* \geq 0, i = 1, \ldots, N$.

Finding the saddle point of $\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})$ by setting the partial derivatives with respect to all variables to zero:

$$\begin{aligned} \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i &= \mathbf{0} \\ \frac{\partial \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i &= 0. \end{aligned} \tag{1.4}$$

After plugging the equations 1.4 into the Lagrangian 1.3, we obtain the final form of the objective function:

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle, \tag{1.5}$$

which leads directly to the final form of the optimisation problem under consideration:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle \\ \text{subject to} \quad & \alpha_i \geq 0, i = 1, \ldots, N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{1.6}$$

The value of the weight vector $\boldsymbol{w}^*$ of the desired separating hyperplane can be obtained by plugging the solution $\boldsymbol{\alpha}^*$ of this optimisation problem into the first equality of the saddle point (1.4). The bias value b^* is then given by the requirement that the functional margins of the nearest examples on both sides be equal, and therefore takes the following form:

$$b^* = -\frac{1}{2} \left(\max_{i:y_i=-1} \{ \langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \rangle \} + \min_{i:y_i=1} \{ \langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \rangle \} \right).$$

At this point, it is possible to classify a new unlabelled input $\boldsymbol{x}$ as

$$y = \text{sgn}(h(\boldsymbol{x})) = \text{sgn}(\langle \boldsymbol{w}^* \cdot \boldsymbol{x} \rangle + b^*) = \text{sgn} \left(\sum_{i=1}^{N} \alpha_i^* y_i \langle \boldsymbol{x}_i \cdot \boldsymbol{x} \rangle + b^* \right), \tag{1.7}$$

which is a particularly convenient form for the following reasons:

- The quantity that is calculated depends on the (scalar) inner products between the training inputs and the new one, which is a computationally cheap operation.
- The quantity $y_i \langle \boldsymbol{x}_i \cdot \boldsymbol{x} \rangle$ is calculated only in the cases of $\alpha_i^* > 0$, which, as follows from the KKT conditions, is equivalent to $g_i(\boldsymbol{w}^*) = 0$. This means, that only the training examples that define the geometric margin by laying on its border, are calculated for the evaluation of a new input.

Note. The set

$$\mathbb{SV} \coloneqq \{ \boldsymbol{x}_i \in \boldsymbol{X}, i = 1, \ldots, N : \hat{\gamma}_i = 1 \} = \{ \boldsymbol{x}_i \in \boldsymbol{X}, i = 1, \ldots, N : g_i(\boldsymbol{w}) = 0 \}$$

is usually relatively small in comparison to the whole dataset. The vectors $\boldsymbol{x}_i \in \mathbb{SV}$ are referred to as *support vectors*.

It is the fact that the SVM classifies only using a small subset of inputs, that is so beneficial to the overall performance of the classifier, and that also gives this classifier its name.

The SVM, as it is derived so far, is a so-called hard-margin SVM. It is a linear machine able to classify an unlabelled input $\boldsymbol{x}$ based on its attributes $x_1, \ldots, x_n$. For a real-life problem, the use and performance of such machine has some limitations:

- The presumption of the dataset $\boldsymbol{X}$ being separable still has to hold. Section 1.5 is dedicated to removing this constraint.
- The original attribute set $x_1, \ldots, x_n$ may not optimally expose the information that the dataset carries, due to being unclean or highly inseparable in the original space. Section 1.4 addresses this issue.

1.4 Kernels

The formerly mentioned issue is addressed by applying so called *kernel trick*: Mapping the input dataset onto a suitable, purposedly chosen new set, but without having to evaluate the mapped values, or even knowing the mapping itself, in order to be able to classify a new example.

Definition 7 (kernel)**.** *Let $\boldsymbol{X} \subset \mathbb{R}^{n \times N}$ be an unlabelled dataset. Let*

$$\begin{aligned} \phi : \mathbb{R}^n &\to F \\ \boldsymbol{x} &\mapsto \phi(\boldsymbol{x}) \end{aligned}$$

be some feature mappping *from the input space to some* feature space $F \subset \mathbb{R}^d$, $d \in \mathbb{N}$. *A* kernel *is then defined as a function*

$$\begin{aligned} K : \mathbb{R}^n \times \mathbb{R}^n &\to \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{z}) &\mapsto K(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}) \rangle . \end{aligned}$$

Note that a kernel has to have an underlying feature mapping ϕ, in order to be correctly defined.

The SVM is then trained on the feature space $\phi(\boldsymbol{X})$, using the *features* $\phi(\boldsymbol{x}_i)$, instead of the attribute (input) space $\boldsymbol{X}$ and the attributes $\boldsymbol{x}_i$. However, mapping each attribute on the corresponding feature one by one is not the actual workflow of the SVM. The computational complexity of such algorithm would be $\mathcal{O}(n)$. Furthermore, no requirements were laid upon the mapping ϕ, so evaluating the feature $\phi(\boldsymbol{x})$, which depends on the number of the attributes N, can be expensive itself.

All of the former theory of developing the classifier is valid in the feature space $\phi(\boldsymbol{X}) \subset \mathbb{R}^d$, as well as it is in the input space $\boldsymbol{X} \subset \mathbb{R}^n$, since no presumption was made about neither d nor n. Therefore, the hypothesis in the feature space looks as follows:

$$h(\boldsymbol{x}) = \sum_{i=1}^{d} w_i \phi_i(\boldsymbol{x}) + b.$$

Utilising the dual interpretation derived in the previous section (see 1.7), and applying the kernel of chosen feature mapping ϕ, this equality can equivalently be rewritten as

$$h(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \langle \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}) \rangle + b = \sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{1.8}$$

Note. The last expression of 1.8 reveals, that the knowledge of the feature map is redundant for the purpose of classification of a new example with the hypothesis $h(\boldsymbol{x})$. Only the kernel needs to be evaluated. Same is the situation in the case of training the SVM on $\phi(\boldsymbol{X})$, because here, the objective to maximize is $\sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ (see Equation 1.6 for comparison).

Usualy, only the kernel, rather than the underlying mapping, is used to specify an SVM classifier. Commonly used kernels and other implementation details are to be found in Chang and Lin (2011).

Not every function $\phi : \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ is a kernel. For the *Mercer theorem* that states the necessary and sufficient conditions for a function to be a valid kernel, refer to Cristianni and Shawe-Taylor (2000).

1.5 Soft Margin and Non-Separability

Although mapping the data to a high-dimensional feature space generally increases the likelihood of linear separability (Ng (2011)), it is not guaranteed it is going to be so in every case. Furthermore, an outlying input can alter the decision boundary of the hard margin classifier dramatically. For the sake of robustness, it is therefore favourable to allow the misclassification and rather introduce an additional variable to penalize it.

The optimisation problem to solve is as follows:

$$\begin{aligned} \text{minimize}_{\boldsymbol{w},b,\gamma} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq 1 - \xi_i, i = 1, \ldots, N \\ & \xi_i \geq 0, i = 1, \ldots, N. \end{aligned}$$

The classifier is allowed to classify with functional margin lower than 1, as well as to misclassify at all. Such cases are penalized by the cost C being added to the objective with the weight equal to the misclassification.

The process of derivation of the dual is similar to the maximal margin case. The Lagrangian of the objective

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}, \boldsymbol{\xi}, \boldsymbol{r}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{N} r_i \xi_i \tag{1.9}$$

lends itself to partial derivatives, which have to be stationary at the saddle point of $\mathcal{L}$:

$$\begin{aligned} \nabla_{\boldsymbol{w}} \mathcal{L} = \boldsymbol{w} - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \quad &= \boldsymbol{0} \\ \frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i \quad &= 0 \\ \nabla_{\boldsymbol{\xi}} \mathcal{L} = \boldsymbol{C} - \boldsymbol{\alpha} - \boldsymbol{r} \quad &= \boldsymbol{0}. \end{aligned} \tag{1.10}$$

The objective function, that results from plugging the KKT equations 1.10 back to the original Lagrangian 1.9, is identical to the Lagrangian of the maximal margin classifier (1.5). The resulting dual problem, with a kernel K applied, therefore takes the following form:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{subject to} \quad & C \geq \alpha_i \geq 0, i = 1, \ldots, N \\ & \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned}$$

Again, the constraints are inherited from the KKT conditions, which are guaranteed to hold.

The quadratic objective of the final form of the optimisation task can be solved by a commercial optimisation software. The final value of the weight vector of the classifier is then obtained by plugging the optimum $\boldsymbol{\alpha}^*$ back to the equations 1.10.

This way, the resulting SVM classifier is obtained.

Note. A softness of an SVM classifier can be referred to as a quantity. An SVM that is sensitive to outlying examples and penalizes misclassification with a high cost, is called a harder model. On the contrary, a softer model is characterized by a low cost setting and insensitivity to noisy examples.

Chapter 2

Feature Selection using SVM

In the process of training a classifier, using all the attributes available in the training data may turn out to be counterproductive. One goal of this chapter is to justify the need for feature pruning and selection, and relate this topic to the theory of SVM.

The latter sections then study two existing feature selection heuristics and their performance, using the data corpus called VPS-30-En, the same one used by Holub et al. (2012) for the task of lexical verb sense disambiguation.

The same machine learning task was being performed in order to measure the quality of the feature subsets that the studied feature selection models produced. The results published by Holub et al. (2012) can therefore serve the purpose of performance comparison baseline. The lexical verb sense disambiguation task is thoroughly discussed in Section 2.5, including the description of the data.

Note. In the theory of SVM, it is crucial to distinguish between the terms *attribute* and *feature*. An attribute is a dimension of an observation, e.g. an age of a person. In the context of the entire dataset, an attribute is a vector of one dimension of all observations – e.g. a vector of the ages of every person in the dataset. On the other hand, a feature is a one to one projection of an attribute into the feature space. The feature selection theory is not exclusively bound to the theory of kernel-based machine learning. Furthermore, throughout this chapter, the SVM can be thought of as a blackbox that turns a labelled training dataset into a separating hyperplane.

This justifies the use of the term *feature* in the sense of the term *attribute*, which is how both these terms will be used from now on.

Note. In this chapter it is necessary to be able to evaluate how useful a feature subset can be as an input for a linear classifier. From now on, the performance of a feature subset will be defined as the best achievable performance of a linear machine which is given said feature subset as its input.

Similarly, the performance of a feature selection algorithm (or, equivalently, a heuristic, a method, etc.) is defined as the performance of the feature subset it produces.

2.1 The Curse of Dimensionality

The *curse of dimensionality* is a term used by Marimont and Shapiro (1979). It is a reference to the fact that a given dataset becomes more sparse with the increasing number of the features in the data. It is therefore easier to train a classifier in more dimensions. However, such a classifier is more likely to reflect anomalies of that specific dataset, rather than inherent universal patterns. This problem is known as *overfitting*.

The fact that for a given dataset the predictive ability of a classifier decreases with increasing dimensionality of the feature space, is known as the *Hughes Phenomenon* (Hughes (1968)).

The following example illustrates that with a richer feature space a random dataset is more likely to be sparse.

Example. Let $\mathrm{U}_d, d \in \mathbb{N}$ be a uniform probabilistic distribution on a d-dimensional cube $\mathcal{C} = (-1,1) \times \cdots \times (-1,1) = (-1,1)^d$.

Consider an observation $\boldsymbol{x}$ of a random variable $X \sim \mathrm{U}_d$.

What is the probability $\mathrm{P} := \mathbb{P}(\|\boldsymbol{x} - \mathbf{0}\| = \|\boldsymbol{x}\| \leq 1)$, i.e. that the random observation $\boldsymbol{x}$ will be no further from the center than by the distance of 1, measured by the Eucleidean distance?

The volume of the d-dimensional ball with a radius equal to 1 is

$$V(\mathcal{B}(\mathbf{0},1)) = \frac{2\pi^{\frac{d}{2}}}{d\Gamma(\frac{d}{2})},$$

whereas the volume of the mentioned cube $\mathcal{C}$ satisfies $V(\mathcal{C}) = 2^d$. What follows is that

$$\mathrm{P} = \frac{V(\mathcal{B}(\mathbf{0},1))}{V(\mathcal{C})} = \frac{\pi^{\frac{d}{2}}}{2^{d-1}d\Gamma(\frac{d}{2})} = \frac{1}{d} \cdot \left(\frac{\pi}{4}\right)^d \cdot \frac{2}{\Gamma\left(\frac{d}{2}\right)} \xrightarrow[d\to\infty]{} 0.$$

The interpretation of this example is as follows: Provided that the distribution of new examples in the neighbourhood of an observed point $\boldsymbol{x}$ is homogenous, the likelihood of a new example being close to $\boldsymbol{x}$ is lower with the increasing number of features d.

Therefore, in the cases, where the number of features exceeds the number of observations, it is necessary to employ a feature selection or pruning mechanism. The Following sections are dedicated to specific examples of such algorithms.

2.2 Recursive Feature Elimination

Recursive Feature Elimination (SVM-RFE) is a backward feature selection algorithm published by Guyon, Weston, Barnhill, and Vapnik (2002). This algorithm iteratively trains an SVM model using the current feature set, ranks the features using the squares of the weights of the classifier $\boldsymbol{w}'\boldsymbol{w}$ as a ranking criteria, and removes the feature with the lowest rank from the feature set.

The weight vector $\boldsymbol{w}$ can indeed serve the purpose of feature ranking. It becomes clear, if it is obtained by an algorithm referred to as *Rosenblatt's Perceptron* by Rosenblatt (1957).

2.3 Intuition using Rosenblatt's Perceptron

The perceptron finds the separating hyperplane $(\boldsymbol{w}, b)$ by building up the training examples. It updates the initial zero hypothesis by adding the misclassified positive (and subtracting the negative) training examples, as the Algorithm 1 demonstrates. As soon as the algorithm iterates through all the input examples and finds all of them correctly classified, it terminates.

Algorithm 1 Rosenblatt's Perceptron algorithm workflow (Rosenblatt (1957))

Require: $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top$ a linearly separable dataset

choose $\eta > 0$ ▷ learning rate
$k \leftarrow 0$ ▷ number of mistakes
$\boldsymbol{w}_0 \leftarrow \mathbf{0}$
$b_0 \leftarrow 0$
$\mathrm{R} \leftarrow \max_{1 \leq i \leq N} \|\boldsymbol{x}_i\|$
repeat
  for $i \leftarrow 1$ to N **do**
    if $y_i(\langle \boldsymbol{w}_k \cdot \boldsymbol{x}_i \rangle + b_k) \leq 0$ **then** ▷ $(\boldsymbol{w}, b)$ misclassifies $\boldsymbol{x}_i$
      $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + \delta y_i \boldsymbol{x}_i$
      $b_{k+1} \leftarrow b_k + \delta y_i R^2$
      $k \leftarrow k + 1$
    end if
  end for
until no mistakes within for loop
return $(\boldsymbol{w}_k, b_k)$

The following theorem and its corollary justify the convergence and correctness of the Perceptron. The proof of the Novikoff theorem can be found in Cristianni and Shawe-Taylor (2000).

Theorem 1 (Novikoff)**.** *Let $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top$ be a linearly separable dataset. Denote $R = \max_{1 \leq i \leq N} \|\boldsymbol{x}_i\|$. Suppose there exists a hyperplane $(\boldsymbol{w}^*, b^*)$ such that $(\langle \boldsymbol{w}^* \cdot \boldsymbol{x} \rangle + b^*) = 0$ separates $\boldsymbol{X}$ and $\|\boldsymbol{w}^*\| = 1$. Let γ be the functional margin of $(\boldsymbol{w}^*, b^*)$ with respect to $\boldsymbol{X}$. Then the number of mistakes t made by the perceptron algorithm satisfies*

$$t \leq \left(\frac{2R}{\gamma}\right)^2.$$

Corollary. Novikoff's theorem implies that the number of mistakes made by the perceptron algorithm is finite, and therefore so is the number of iterations of the algorithm itself. Since the perceptron only converges in case it has found a separating hyperplane, its convergence also implies its correctness.

Assuming the initial hypothesis is a zero vector, the weight vector of the final hypothesis takes the form of a linear combination of the input examples:

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i.$$

Interestingly, this is the same expression as in the case of the weights of the SVM classifier (see the Equations 1.4). However, the interpretation of the α_i's, which were present as Lagrange multipliers, is now different. The value of α_i equals the number of cases when the misclassification of $\boldsymbol{x}_i$ caused an update of $\boldsymbol{w}$. If an input $\boldsymbol{x}_i$ that is rather dominant in one of its components (i.e. large $|x_{ij}|$), proves to be influential in the learning process (i.e. large α_i and therefore large $|w_i|$), it is natural to expect the feature $j \in 1, \ldots, n$ to be influential itself.

By this intuition, the weight vector of a linear classifier is a measure of how each of the features influenced the learning process, and therefore how useful it is as a predictor. SVM-RFE uses the SVMs as such a classifier.

2.4 SVM-RFE Workflow

The algorithm initiates with a labelled input dataset $\boldsymbol{X}$. At each step, an SVM model is trained on $\boldsymbol{X}$, and the vector of the second powers of the components of $\boldsymbol{w}$ is used to rank the features. Then the feature with the lowest ranking is removed from the set of the remaining data columns. The next step is conducted using a dataset without the removed column.

This means that at each iteration, one feature is removed. The algorithm therefore converges after n steps, when the last feature is removed. The algorithm returns the ranking of all the features (i.e. the order they were removed in).

See the Algorithm 2 for the SVM-RFE workflow pseudocode.

Algorithm 2 Original SVM-RFE workflow

Require: $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top$, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{in})$ is a training dataset

```
surviving features ← 1:N                          ▷ starting with all features
rankings ← empty vector
while # of surviving features > 0 do
    X_SF ← copy of X containing only surviving features
    (w, b) ← train SVM using X_SF
    ranking ← w'w                                  ▷ vector of second powers
    worst feature ← the one from the surviving features with the lowest
ranking
    concatenate rankings with the worst feature
    remove the worst feature from surviving features
end while
return rankings
```

2.5 SVM-RFE Implementation

One of the goals of this thesis is to evaluate the studied feature selection heuristic's performance using an existing data corpus as an input and the results of a related machine learning task as a performance baseline. For a given feature subset, an SVM is trained on each dataset of the corpus and the average SVM

performance is calculated using cross-fold validation. The measured SVM performance then defines the quality of the feature selection heuristic.

The data used for this purpose are the datasets from the VPS-30-En corpus. The related machine learning task of lexical verb sense disambiguation, including feature pruning techniques and their performance results, was published by Holub et al. (2012). It is a multi-class supervised machine learning problem that uses the VPS-30-En datasets. The classified examples are observations of contexts of English verbs in natural language sentences, and the classes they are classified into are the multiple possibilities of sense each of the verbs can have.

The VPS-30-En data corpus consists of 30 separate datasets, each of them for one of the studied verbs. Some of the main properties of this corpus are as follows:

- Each verb is classified into its own classes. The number of the classes varies from 4 to 19. Most of the datasets have 5 to 9 classes.
- The majority of the features are common for all datasets. However, for every verb a new set of tailored features is provided. This means that the datasets have to be processed separately, which results in a unique feature subset for each of them.
- Each of the datasets has more features, than observations. It is likely, that after pruning the features using a feature selection mechanism such as SVM-RFE, the SVM that is trained on the dataset will perform better. Such a data corpus is suitable for experiments with feature selection techniques.
- The datasets are divided into 9 folds of approximately the same size. These folds were used for cross-validation in the original research by Holub et al. (2012). The cross-validation folds and methods used in this thesis are the same. The results are therefore comparable.
- Most of the datasets are rather imbalanced – some classes are dominant, some are very rare. See figure 2.1 for the illustration of some of the datasets' imbalance.

The original code by Guyon (2002) published for the purposes of Guyon et al. (2002) results reproduction was used as a base for custom implementation. The entire implementation code for the studied methods can be found as an attachment to this thesis.

2.6 Preprocessing and Tuning

The performance of the SVM-RFE is highly dependent on the setup of its parameters. This section discusses the process and impacts of choosing the SVM-RFE parameter combination.

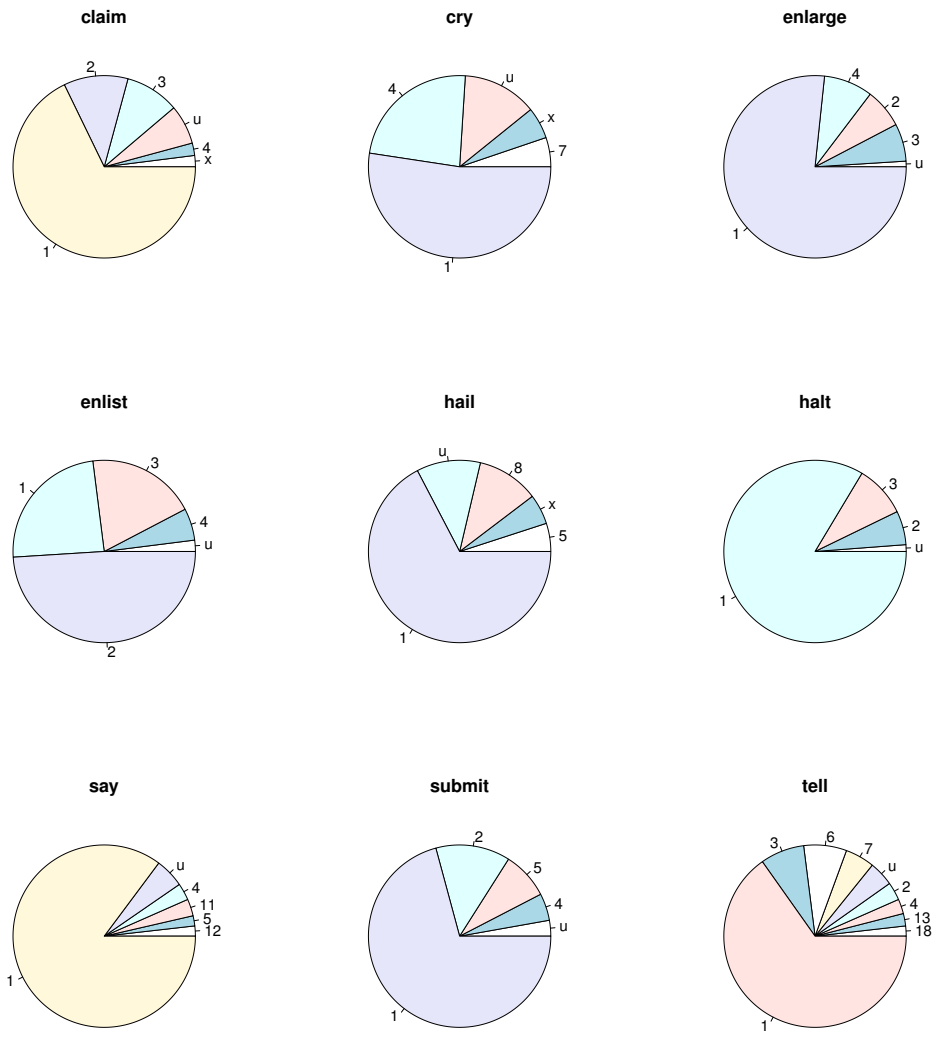

Figure 2.1: The distribution of class labels of some of the datasets from the VPS-30-En corpus is imbalanced.

cost

The `e1071 R` package interface to `libsvm` library does not allow for specific data folds to be passed to the `tune.svm` method instead of the default random ones. For this reason, a custom parameter tuning mechanism was implemented:

A cross-validation step was added to the RFE routine. The result is the mean performance of the SVM with the current feature set. This cross-validation is executed repeatedly with different `cost` parameter settings, and the best performing cost setting is used in the next step (which trains on the same feature set, but the entire dataset).

kernel

The original SVM-RFE implementation uses the linear kernel. However, for the studied task, the radial basis kernel has proven to perform significantly better and was kept as the kernel of choice. Since the underlying mapping of this kernel maps to a space with a different dimension, the weight vector in the feature space does not correspond to the original features. The `svm$coefs` property of the `svm` object carrying the trained model maps the weight vector in the feature space back to the input space. It was therefore used as a measure of the feature importance.

The gamma parameter of the radial kernel was also experimented with. Based on the results, the default value of $\gamma = \frac{1}{\#\ features}$ was kept and was being used for all further experiments.

class weights

For the sake of comparability, the only measure of model performance used in this thesis is merely the *accuracy* of the model, i.e.

$$Acc = \frac{\#\ correctly\ classified}{\#\ all\ test\ examples}.$$

A trained model is used to predict the classes of the test data, and the fraction of correctly classified examples over the test dataset size is said to be the performance of the model.

As it was previously mentioned, most of the studied datasets a have rather imbalanced distribution of the class labels. Using the prediction accuracy as a performance measure on such data leads to a model which ignores the rare classes and predicts well the dominant ones to be chosen as the best performing. However, due to its inability to predict the rare-during-traing classes, such a model would not perform well on a dataset with a higher representation of the classes it fails to predict.

The accuracy of a model which does not even attempt to predict the rare classes, is limited by the incidence of the rare examples it surely misclassifies. For that reason, experiments with the class weights were also conducted. Three separate class weights settings were applied for each SVM-RFE run:

- $cw_1 = 1$
- $cw_2 = \frac{1}{\#class\ examples}$

- $cw_3 = \frac{1}{\#class\ examples} + \#classes$,

i.e. all classes having the same weight, then the weights of the classes being inversely proportional to the frequency of their occurence, and finally a setting with a somewhat weaker preference of the rare classes.

noise pruning

It is hard to train an accurate SVM model, which is necessary to measure the contribution of the features, on a noisy dataset (Tan et al. (2010)). The SVM-RFE heuristic, which makes decisions solely based on the weight vector of an SVM model trained on all of the features, is therefore prone to underperformance on noisy data.

For this reason, finally, other experiments were conducted at the level of data preprocessing. The features with variance lower than the preset threshold were removed. Although this is the most elementary step of data processing, the experiments with the threshold had to be conducted with fine-tuned models. That is due to the significant variance and unpredictability of the performance of a given model across different subsets of features supplied to the SVM-RFE routine.

Based on the initial experiments, three levels of the threshold were preset for further testing: var $\boldsymbol{x}_i > 0.01$, var $\boldsymbol{x}_i > 0.02$, and var $\boldsymbol{x}_i > 0.05$.

conclusions

The following observations emerged during the tuning process:

- The SVM kernel selection and cost and gamma parameters tuning results are independent on the other aspects of optimisation. i.e. variance-based feature pruning and class weight selection. Therefore, the kernel and cost range for tuning should be selected (and fixed) first, to reduce the experiment combinations.

- For the final accuracy, it is not critical to prune the features drastically in the preprocessing phase. The SVM-RFE routine eliminates the features with low variance anyway. However, with preprocessed data, the SVM-RFE runs much faster. That is because SVM is not well scalable, so the first steps, when the SVM is trained on most features, are the most computationally expensive. These steps are ommited with pruned data.

- SVM-RFE is likely to produce a smaller feature subset with pruned data.

- The cost tuning subroutine tends to choose higher cost values at each step, creating a harder margin model that is sensitive to outliers, when the provided data is pruned, and therefore cleaner. On the contrary, providing a raw dataset results in a soft SVM being chosen as the best performing one.

- Forcing the SVM-RFE to be more sensitive to the rare classes by giving them greater weights has the same effect as providing less pruned data – small performance gain, significantly softer model at each step, larger output feature set.

2.7 SVM-RFE Performance

The Table 2.1 compares the best results obtained by the tailored models T1, T2, T3 used by Holub et al. (2012) with the SVM-RFE performance. It also documents some of the observations mentioned in section 2.6.

Verb	**Best tailored**			**R1**		**R2**		**R3**		**R4**		**R5**	
	M	#F	Acc	#F	Acc	#F	Acc	#F	Acc	#F	Acc	#F	Acc
access	T3	55	79.7	7	78.7	37	79.3	56	77.7	17	79.3	38	78
ally	T2	54	79.6	12	72.8	12	72.4	21	68.4	87	71.6	58	73.2
arrive	T3	41	82.6	41	73.6	63	73.2	55	73.6	9	73.2	68	72.8
breathe	T3	41	81.0	35	76.3	43	76	30	76.8	63	80	67	78.9
claim	T1	75	87.4	17	84.4	13	84.2	31	83.8	36	85.6	52	86
cool	T3	36	67.6	28	70.7	76	71	57	68	75	72	80	71.3
crush	T3	56	53.5	32	53.7	33	53.1	45	51.4	46	52.6	62	53.4
cry	T3	44	80.4	41	80.4	41	80.8	61	79.2	68	78.8	39	79.2
deny	T1	74	67.7	6	62.4	18	61.3	60	57.3	77	65	20	65.7
enlarge	T3	43	84.8	25	86.7	48	86.1	45	85.1	25	87	30	87
enlist	T3	51	89.9	11	86.7	20	87.7	38	88	17	89.7	52	89.7
forge	T1	86	59.7	41	57.7	60	59.4	26	58.6	85	63.1	90	62.3
furnish	T3	49	79.0	72	73	35	73.6	56	75	51	74	36	74
hail	T1	73	85.4	75	85.6	53	85.6	43	84.3	37	88.3	44	87
halt	T3	59	90.9	17	89.6	15	88.8	23	86	11	90	30	89.6
part	T1	74	72.7	57	80	36	80	40	78.6	69	80	80	79.6
plough	T3	44	76.5	28	77.7	31	77.7	34	74.9	48	73.6	51	73.3
plug	T3	41	61.7	30	65.3	38	65.3	28	63	67	64.7	63	64.7
pour	T3	77	63.8	72	63.6	72	63.3	63	63.3	82	64	105	64.3
say	T1	82	90.8	51	89.6	49	89.4	64	89.2	77	90.2	73	89.4
smash	T3	46	77.7	25	77.4	24	77.7	51	74.4	74	75	103	76.3
smell	T3	37	63.7	63	62.3	64	58.3	53	58.3	67	61.7	62	61.3
steer	T3	55	50.6	64	48.4	55	47.4	72	48	30	54	33	54.7
submit	T3	76	86.8	19	89.2	30	90	30	90.4	11	88.4	8	88.8
swell	T3	45	62.8	39	57.6	36	57.6	49	57.3	92	57.6	76	58.2
tell	T3	69	81.2	36	82.2	78	81.6	64	81.8	77	83	82	83
throw	T3	147	56.6	95	49.3	94	48.9	95	48.2	128	54.7	146	54.7
trouble	T1	75	72.4	46	73.3	93	72.3	74	73	95	75	91	74.3
wake	T1	75	77.7	69	81.7	22	80	33	78.3	13	82	35	81.7
yield	T3	46	56.0	56	59	71	58.3	77	56.7	70	54.7	73	56.3
average				40	72.96	45	72.68	49	71.62	57	73.63	62	73.62

Table 2.1: Comparison of the best of T1, T2 and T3 models against SVM-RFE models R1 – R5 with various setups. The three columns named as "Best tailored" are taken directly from Holub et al. (2012).

SVM-RFE models R1 - R5 are five of the best performing setups that were tuned. Table 2.2 summarizes the setups of these models. An SVM model that is trained on the output of R4, usually fails completely even to try to predict the rare classes with the sensitivity being 0 or less than the proportion of such class. That is due to the equal class weights. Therefore, the model R5 was chosen over R4 as the final SVM-RFE choice for comparison with T1 – T3 models and for

Model	var. threshold	class weights	costs	#F	accuracy
R1	0.05	cw_1	$(\frac{1}{4}, 1, 4, 16, 64)$	40	72.96
R2	0.05	cw_3	$(\frac{1}{4}, 1, 4, 16, 64)$	45	72.68
R3	0.05	cw_2	$(\frac{1}{4}, 1, 4, 16, 64)$	49	71.62
R4	0.02	cw_1	$(\frac{1}{16}, \frac{1}{4}, 1, 4, 16, 64)$	57	73.63
R5	0.02	cw_3	$(\frac{1}{16}, \frac{1}{4}, 1, 4, 16, 64)$	62	73.62

Table 2.2: Five of the best performing SVM-RFE setups.

further experiments with SVM-RFE. The performance measures of the baseline models, which are referred to as "Best tailored", were copied from Holub et al. (2012). For additional details regarding these models refer to this paper.

2.8 Flip Heuristics

As Guyon (2003) points out on page 1159, in a search for performance improving modifications of a feature selection algorithm, "The only recommendation that is almost surely valid is to try the simplest things first".

Bit-Flip (BF) and Attribute-Flip (AF), published by Samb et al. (2012) are a pair of such algorithms. Their approach is driven by a known drawback of SVM-RFE (discussed, for instance, by Tan et al. (2010)), that the nested feature subsets are monotonic. This means, that at some point it might be beneficial for the routine to return back to a previously discarded attribute. SVM-RFE is not capable of such action.

Both algorithms examine the neighboring attribute subsets of the SVM-RFE result. If a better solution is found, it is used instead of the original one. This way a previously discarded attribute can be brought back.

For the sake of the BF and AF description, let $\mathbb{S}$ be the set of the features present in an SVM-RFE result.

The Bit-Flip algorithm examines every solution with exactly one feature inverted, i.e. if $x_i \in \mathbb{S}$ ($x_j \notin \mathbb{S}$), the set $\mathbb{S} \setminus x_i$ ($\mathbb{S} \cup x_j$) will be examined. The set of all the subsets that Bit-Flip creates is then $\mathbb{S}_{BF} = \mathbb{S}^+_{BF} \cup \mathbb{S}^-_{BF}$, where $\mathbb{S}^+_{BF} = \{X : X = \mathbb{S} \cup \{x_j\}, x_j \notin \mathbb{S}\}$ and $\mathbb{S}^-_{BF} = \{X : X = \mathbb{S} \setminus \{x_j\}, x_j \in \mathbb{S}\}$.

The Attribute-Flip algorithm swaps one feature $x_i \in \mathbb{S}$, with one feature $x_j \notin \mathbb{S}$ that was discarded. The set of all solutions that are examined is therefore $\mathbb{S}_{AF} = \{X : X = \mathbb{S} \cup \{x_i\} \setminus \{x_j\}, x_i \notin \mathbb{S}, x_j \in \mathbb{S}\}$.

The design of the flip algorithms faces some challenges that may limit the usability:

- They do not search through the neighboring solution space at each step of the SVM-RFE. If a feature x_i is discarded, all of the following nested subsets are optimised based on the weight vector that does not include the x_i variable. This influence to every other step cannot be taken back by finally enlisting the feature x_i back.

- The BF algorithm does not preserve the number of the features of the subset. It either has one more, or one less. That is an advantage, since it is desirable to have a smaller feature subset, but it also limits the use at every step of the RFE, since such a setup would not be guaranteed to converge.
- The flip heuristics are computationally expensive – BF has a complexity of $\mathcal{O}(n)$ and AF of $\mathcal{O}(n^2)$, where n stands for the number of the features. This limits their use at each step of SVM-RFE as well.
- No validity criteria is considered when the neighboring solutions are being chosen. The flip algorithms simply use all the solutions that neighbor in the defined way.

Chapter 3

Evalution-Based RFE

Out of all the experiments that had been conducted in the search for an improvement of the SVM-RFE performance, one particular method has immensely outperformed any other approach. This section is dedicated to the description of this SVM-RFE modification. For the sake of readability, let's refer to this routine as the Evaluation-Based RFE (EB-RFE).

The EB-RFE is a modification that affects each step of the SVM-RFE with a computational complexity of $\mathcal{O}(1)$. Thanks to this, it is applicable repeatedly during the SVM-RFE runtime.

3.1 Algorithm Workflow

The EB-RFE algorithm requires the C constant to be preset. At each step of the SVM-RFE routine, instead of the feature with the smallest weight being instantly discarded, multiple features with the worst rankings are marked for discarding in the next step. Multiple subsets $\mathbb{S}_1, \ldots, \mathbb{S}_C$, are created this way, each one with a different feature missing.

At the next step, each of the subsets $\mathbb{S}_1, \ldots, \mathbb{S}_C$ is cross-validated, and the best performing subset is chosen.

This modification does not affect the number of steps of the routine, and therefore it has no effect on the convergence as well. The EB-RFE routine has the same input and output requirements and possibilities, as the original SVM-RFE.

The original SVM-RFE routine only returns the ranking of the features. However, thanks to the possibility to tune and cross-validate at each step, it is possible to return the best performing subset together with its performance as well. The pseudocode of the EB-RFE workflow, including these modifications, is shown in Algorithm 3. It is also the workflow of the `R` script attached to this thesis, which was used to obtain the presented results.

3.2 Scalability and Other Properties

EB-RFE attempts to overcome all of the previously mentioned drawbacks of the flip algorithms. Namely:

Algorithm 3 Evaluation-based SVM-RFE workflow

Require: $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top$, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{in})$ is a training dataset divided F times into a train/test pair

set C ▷ # of the worst features to evaluate
set $Costs$ ▷ vector of cost values to tune through
$\texttt{SF} \leftarrow 1{:}N$ ▷ surviving features
$\texttt{best subset} \leftarrow 1{:}N$
$\texttt{best acc} \leftarrow 0$
for $c \leftarrow 1$ to C **do**
    $\texttt{SF}_C \leftarrow 1{:}N$ ▷ surviving feature set candidates
end for
$\texttt{rankings} \leftarrow$ empty vector
while # of $\texttt{SF} > 0$ **do**
    for $c \leftarrow 1$ to C **do**
        for *cost* in $Costs$ **do**
            $\boldsymbol{X}_c \leftarrow$ copy of $\boldsymbol{X}$ containing only $\texttt{SF}_c$
            for $fold \leftarrow 1$ to F **do**
                $(\boldsymbol{w}, b) \leftarrow$ train SVM using train portion of $\boldsymbol{X}_{SF}$
                $\texttt{acc}_{fold} \leftarrow$ evaluate $(\boldsymbol{w}, b)$ using test portion of $\boldsymbol{X}_{SF}$
            end for
            $\texttt{acc}_{cost} \leftarrow$ average of all $\texttt{acc}_{fold}$
        end for
        $\texttt{acc}_c \leftarrow$ best of all $\texttt{acc}_{cost}$
        if $\texttt{acc}_c \geq \texttt{best acc}$ **then**
            $\texttt{best subset} \leftarrow \texttt{SF}_c$
            $\texttt{best acc} \leftarrow \texttt{acc}_c$
        end if
    end for
    $\texttt{worst feature} \leftarrow$ the feature with worst $\texttt{acc}_c$
    concatenate `rankings` with the `worst feature`
    remove the `worst feature` from `SF`
    $\boldsymbol{X}_{SF} \leftarrow$ copy of $\boldsymbol{X}$ containing only `SF`
    $(\boldsymbol{w}, b) \leftarrow$ train SVM using $\boldsymbol{X}_{SF}$
    $\texttt{ranking} \leftarrow \boldsymbol{w}'\boldsymbol{w}$
    for $c \leftarrow 1$ to C **do** ▷ prepare candidates for removal
        $\texttt{wf}_c \leftarrow$ the feature with the c-th lowest `ranking`
        $\texttt{SF}_c \leftarrow$ `SF` without $\texttt{wf}_c$
    end for
end while
return `rankings, best subset, best acc`

- Unlike the flip algorithms, EB-RFE drives the flow of the SVM-RFE at each step by picking one of the C neighboring solutions. The evaluation of the subsets using cross-validation is therefore conducted at each step of the feature elimination as well.

- EB-RFE only marks *some* of the attributes for removal, not all that are neighboring in the defined way. It does so based on their weights, which is a relevant criteria to choose upon.

- EB-RFE has a constant computational complexity.

- EB-RFE requires the dataset to be divided into the train and test portions. Another option is to cross-validate the subset candidates $\mathbb{S}_1, \ldots, \mathbb{S}_C$ on the whole dataset, which is also how the current implementation works.

EB-RFE adds a new level of scalability to the algorithm that both the original SVM-RFE and the flip methods lack. There is no constant, which could be modified to "flip more". Such need would require the algorithm to be modified. Similarly, the SVM-RFE cannot scale over the minimum granularity of the number of features removed at a time being 1, which is a standard setting.

The performance of the EB-RFE routine increases with greater C. In the case of the studied data, this relation is monotonic for the low settings of C. Therefore, EB-RFE can scale towards better results with greater demands for computational resources.

3.3 Performance

No parameter tuning was performed during the EB-RFE performance testing. For the sake of comparability, all of the presented EB-RFE models in this section have the same settings, as the R5 model (Table 2.2 shows the specification).

The performance of the EB-RFE has been tested with several C setups. It is apparent, that the higher the C is set, the better is the resulting accuracy. This dependence has been tested within the range $C \in [1, 20]$. Note that EB-RFE with $C = 1$ is the same heuristic as the plain SVM-RFE.

Table 3.1 compares the baseline, which is given by the "Best tailored" model by Holub et al. (2012), the original SVM-RFE with the R5 setup, and various setups of the EB-RFE heuristic.

An EB-RFE model with $C = \mathrm{X}$ setup is listed as EBX.

The results of the "Best tailored" model in this table were copied from Holub et al. (2012), which is where this model was originally published. The last five rows of this table are clarified in the Table 3.2. Note that in the improvement rows, the higher accuraccies and smaller feature subsets are considered better.

The probabilities $\mathbb{P}_1$ and $\mathbb{P}_2$ in the Table 3.1 were obtained by a student's t-Test, with a sample of the 30 improvements measured for a given column. This means, that an assumption is being made that the improvements are observations of a normal random variable, with an unknown mean μ, and with the variance being estimated by the variance of the sample.

The test is then conducted under the assumption of the H_0 hypothesis being valid, and with the hypotheses as follows:

$$H_0 : \mu \leq 0$$
$$H_1 : \mu > 0.$$

The probabilities $\mathbb{P}_1$ and $\mathbb{P}_2$ in the Table 3.1 are then equal to $100\% - P$, where P stands for the P-value of the corresponding test.

The column $\pm$ refers to the EB20 model accuracy 95% confidence interval radius. It is the 95 % confidence interval of a t-Test conducted with a sample of 9 results, each one from a different cross-validation fold. The hypothesis here is that the real mean of the accuracy is equal to the sample mean, with the unequality as an alternative.

In some cases (namely words "deny", "smash", "smell", "wake", "yield"), the EB-RFE routine produced two or three subsets with very similar performances. In this cases, the smallest set (measured by count of features) is listed, even if the performance (measured by accuracy) of such feature set is slightly suboptimal.

3.4 Runtime

The performance measures were conducted on a consumer grade computer with an 8 virtual core Intel 6700hq CPU, and 16GB of DDR4 RAM. All of the computation was performed on the CPU (not GPU).

The current EB-RFE implementation employs a virtual CPU cluster of the size equal to the number of cores available on the hardware it is ran on. Each verb from the corpus runs its own routine, which is attached to one of the cores. This way, all the CPU power available is utilized, up to the point, when there is no other task to replace a finished one (e.g. 23 words finished, 7 still running on an 8 core CPU). To minimize the uneffective phase, the tasks were ordered decreasingly according to their demands. This way, the computation runtime of all the 30 tasks is proportional to the C parameter, with the dependence being

$$runtime \sim 3.5 \cdot C \ hours.$$

More than a half of the runtime accounts for the task corresponding to the word "throw" to finish, even though it is triggered at the very beginning.

3.5 Discussion

This section discusses the possibilities of further development of the EB-RFE model and some of its properties that have not yet been mentioned.

parallel computation

The current implementation of the EB-RFE algorithm is designed to be launched on a consumer grade computer with a low number of CPU cores. However, the code can be easily rewritten for the use on a large scale CPU cluster.

Verb	Best tailored			R5		EB3		EB5		EB8		EB12		EB20				
	M	#F	Acc	#F	Acc	#F	Acc	#F	Acc	#F	Acc	#F	Acc	#F	Acc	±	I_{BT}	I_{R5}
access	T3	55	79.7	38	78	11	78.7	9	79	6	80	18	82.3	32	81.7	6.4	2.5	4.7
ally	T2	54	79.6	58	73.2	58	73.2	65	74	22	74.8	31	75.6	27	77.2	6.1	-3	5.5
arrive	T3	41	82.6	68	72.8	2	73.2	5	75.6	8	77.2	5	77.6	24	80.4	6	-2.6	10.4
breathe	T3	41	81.0	67	78.9	68	79.7	53	80.3	53	81.4	57	82	82	83.4	3.3	3	5.7
claim	T1	75	87.4	52	86	50	86	51	86.8	11	87.4	43	87.6	19	89.6	3	2.5	4.2
cool	T3	36	67.6	80	71.3	45	74.7	63	74.3	31	74.3	31	75.4	34	77.7	6.9	14.9	9
crush	T3	56	53.5	62	53.4	52	54.6	30	56.6	30	56.6	31	57.4	37	58.8	5.2	9.9	10.1
cry	T3	44	80.4	39	79.2	58	79.3	67	81.2	22	80.8	26	82.5	18	83.6	6	4	5.6
deny	T1	74	67.7	20	65.7	18	67.7	18	67	22	68	10	69.4	34	69.7	5	3	6.1
enlarge	T3	43	84.8	30	87	40	88	34	88.7	34	89.7	33	90.4	29	91.7	2.5	8.1	5.4
enlist	T3	51	89.9	52	89.7	22	90.3	21	90.3	16	91.4	28	92	33	92.4	3.5	2.8	3
forge	T1	86	59.7	90	62.3	77	63.2	88	63.1	70	64.8	74	66.6	67	67.4	3.3	12.9	8.2
furnish	T3	49	79.0	36	74	46	75.3	40	76	18	77	25	78	51	78.7	4.3	-0.4	6.4
hail	T1	73	85.4	44	87	31	87.6	25	89	25	90.3	25	90.9	27	91.3	4.2	6.9	4.9
halt	T3	59	90.9	30	89.6	25	88.8	32	90.4	26	90.8	20	92	27	92	3.4	1.2	2.7
part	T1	74	72.7	80	79.6	43	80.6	31	81.6	72	82	52	83.6	31	84.3	3.5	16	5.9
plough	T3	44	76.5	51	73.3	27	75.6	24	78	21	80.5	31	81.3	19	81.2	6.7	6.1	10.8
plug	T3	41	61.7	63	64.7	81	64.7	66	67.4	70	63.7	45	68.7	36	69	3.2	12.3	7.1
pour	T3	77	63.8	105	64.3	89	65	62	65.6	72	66.3	86	67	58	70.3	6.4	10.2	9.3
say	T1	82	90.8	73	89.4	69	89.6	63	90.2	44	90.8	26	90.6	20	93.2	2.1	2.6	4.3
smash	T3	46	77.7	103	76.3	47	76.3	41	77.3	102	77.7	27	78	19	79.3	4.6	2.1	3.9
smell	T3	37	63.7	62	61.3	91	62.3	51	62.3	16	62.4	14	63.3	20	64.7	2.8	1.6	5.5
steer	T3	55	50.6	33	54.7	35	55.7	61	55.3	16	57.3	54	57.3	39	59	8.3	17	7.9
submit	T3	76	86.8	8	88.8	7	89.2	8	90.4	8	90.4	7	89.6	29	91.6	2.4	5.5	3.2
swell	T3	45	62.8	76	58.2	49	60	48	60.6	51	63.2	33	64	47	68	3.4	8.3	16.8
tell	T3	69	81.2	82	83	79	83.6	64	83.6	72	84.6	65	84.6	34	86.4	2.6	6.4	4.1
throw	T3	147	56.6	146	54.7	146	55.5	140	56.3	140	56.3	85	57.1	91	58.4	3.9	3.2	6.8
trouble	T1	75	72.4	91	74.3	86	75	88	75.3	38	77	16	77.4	56	77.7	5	7.3	4.6
wake	T1	75	77.7	35	81.7	40	83	18	82.7	29	83.7	11	83.3	25	84	4.6	8.1	2.8
yield	T3	46	56.0	73	56.3	78	57.3	70	59	43	60.6	14	58.3	18	62.7	6.3	12	11.4
average	×	60.9	74.01	61.6	73.62	52.3	74.46	47.9	75.26	39.6	76.12	34.1	76.79	36.1	78.19	×	6.13	6.54
I_{PREV}	×	×	×	-1	-0.52	18	1.13	9	1.08	21	1.14	16	0.88	-6	1.82	×	×	×
I_{BT}	×	×	×	-1	-0.52	16	0.61	27	1.7	53	2.86	78	3.77	69	5.65	×	×	×
$\mathbb{P}_0(I_{BT} > 0)$	×	×	×	×	34.54	×	82.43	×	98.29	×	99.92	×	99.99	×	100	×	×	×
I_{R5}	×	×	×	×	×	18	1.13	29	2.23	55	3.4	81	4.31	71	6.2	×	×	×
$\mathbb{P}_0(I_{R5} > 0)$	×	×	×	×	×	×	100	×	100	×	100	×	100	×	100	×	×	×

Table 3.1: Comparison of the EB-RFE (EB3 – EB20), the best of the tailored models, and the SVM-RFE (R5). The EB20 model performance is listed including the 95 % confidence interval radius and the relative percentual improvement of the accuracy over the "Best tailored" (I_{BT}) and the "R5" (I_{R5}) models. The I_{PREV} row demonstrates the monotonicity of the EB-RFE performance, the I_{BT} compares each model against the best tailored model, and the I_{R5} row compares each model against the SVM-RFE model R5.

I_{PREV}	Average relative improvement over the model listed to the left, in %
I_{BT}	Average relative improvement over the "Best tailored" model, in %
$\mathbb{P}_0(I_{BT} > 0)$	$\mathbb{P}_1 :=$ Probability of I_{BT} being greater than 0, in %
I_{R5}	Average relative improvement over the SVM-RFE "R5" model, in %
$\mathbb{P}_0(I_{R5} > 0)$	$\mathbb{P}_2 :=$ Probability of I_{R5} being greater than 0, in %

Table 3.2: Performance statistics notation in Table 3.1

Instead of running the cross-validation of all C subsets serially, a separate task can be sent to the cluster thread pool for each subset and each fold. This would reduce the computational time approximately to the level of the original, unvalidated, SVM-RFE.

accuracy and compactness trade-off

Every feature subset, which is found during the EB-RFE workflow, is evaluated. The relation between the iteration number and the measured performance is usually concave, with one maximum. The subset found at the maximal phase is then returned as the optimal solution. With the higher C settings, the algorithm is more likely to find its optimum sooner. Also, this bigger feature subset usually outperforms any of the smaller sets found with the smaller C settings.

However, as the "higher C" algorithm proceeds, it still produces better subsets, than the "smaller C" models during their optimal phases.

The accuracy and compactness trade-off is a problem yet to be studied. Even though the mentioned behavior is not exclusive, the current results indicate, that a higher C setting is not harmful neither to the accuracy, nor sparsity, of the resulting SVM classifier.

dynamic C parameter

A dynamic adjustment of the C parameter could provide the possibility of the algorithm flow regulation. In a step, when a small subset of features with apparently small weights is found, there is no need for the parameter C to be high. On the other hand, if the decision rule is uncertain, it might be beneficial to try to evaluate more of the candidate subsets.

Conclusion

In this thesis, a thorough mathematical description of the machine learning model called Support Vector Machines (SVM) was provided. The possibilities of its use for the problem of feature selection are discussed and two existing feature selection algorithms, namely Recursive Feature Selection (SVM-RFE) by Guyon et al. (2002) and Bit Flip and Attribute Flip by Samb et al. (2012) are studied theoretically.

Furthermore, the SVM-RFE performance was measured using an experimental `R` implementation. One of the important conclusions of these experiments is the fact, that SVM-RFE, as well as the SVM itself, is greatly dependent on careful parameter tuning, data processing and implementation decisions. The SVM-RFE heuristic has also shown its computational inexpensivity and potential for further modifications, which results from its simplicity.

The main contribution of this thesis is the proposition of a SVM-RFE modification called Evaluation-Based RFE (EB-RFE). To the best of our knowledge, however simple, this method has neither been published nor used yet. Thorough description and analysis of this heuristic is provided. Comparable results of the EB-RFE performance were calculated using the reference VPS-30-En dataset and the results of the related lexical verb sense disambiguation machine learning task published by Holub et al. (2012).

The original SVM-RFE algorithm repeatedly removes the feature that contributes the least to an SVM decision rule. By introducing an evaluation substep into the SVM-RFE routine, the EB-RFE algorithm can determine which feature to remove not only based on this heuristic, but also taking into account the real performance of the possible resulting subsets.

A statistically significant performance gain of EB-RFE was proven during the performance experiments. In addition to greater accuracy, the EB-RFE routine has been able to produce much smaller feature subsets, which is one of the main requirements laid upon any feature selection heuristic.

Finally, unlike the reference heuristics, the proposed algorithm has shown to be capable of producing SVM models that are more sparse and yet perform better in exchange for greater computational demands. Consequently, since it can be easily implemented and launched on a CPU cluster of any size, EB-RFE has no usability limitations as far as large scale machine learning applications go.

Bibliography

B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers, 1992.

C. Chang and C. Lin. Libsvm: A library for support vector machines, 2011.

C. Cortes and V. Vapnik. Support-vector networks, 1995.

N. Cristianni and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, 2000.

I. Guyon. R implementation of the support vector mahine recursive feature extraction (svm-rfe) algorithm. `www.uccor.edu.ar/paginas/seminarios/Software/SVM-RFE.zip`, 2002.

I. Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines, 2002.

M. Holub, V. Kríz, S. Cinková, and E. Bick. Tailored feature extraction for lexical disambiguation of english verbs based on corpus pattern analysis. In *COLING*, 2012.

G. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, Jan 1968. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054102.

R. B. Marimont and M. B. Shapiro. Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70, 1979. doi: 10.1093/imamat/24.1.59. `http://imamat.oxfordjournals.org/content/24/1/59.abstract`.

A. Ng. Machine learning course cs229 lecture notes. `http://cs229.stanford.edu/notes/cs229-notes3.pdf`, 2011.

F. Rosenblatt. The perceptron: A perceiving and recognizing automaton (project para), 1957.

M. L. Samb, F. Camara, S. Ndiaye, Y. Slimani, and M. A. Esseghir. A novel rfe-svm-based feature selection approach for classification., 2012.

M. Tan, L. Wang, and I. W. Tsang. Learning sparse svm for feature selection on very high dimensional datasets. In Johannes Frnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1047–1054. Omnipress, 2010. `http://www.icml2010.org/papers/227.pdf`.

VPS-30-En. Verb pattern sample, 30 english verbs. `http://ufal.mff.cuni.cz/spr/vps-30-en`.