UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

Draft

Analysis of the SVM-RFE algorithm for feature selection

Author:
Robert PLANAS

Director:
Luis A. BELANCHE

Bachelor Degree in Informatics Engineering
Specialization: Computing

June 8, 2021

List of Symbols

n	Number of observations	
d	Number of dimensions	
X	Dataset	$\{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$
$\vec{x_i}$	Example, observation, ...	$\{x_1, x_2, \ldots, x_d\}$
Y	Dataset labels	$\{y_1, y_2, \ldots, y_n\}$
$\vec{w}$, $\mathbf{w}$	Weight vector	$\{w_1, w_2, \ldots, w_d\}$
b	Constant term, bias	E.g. $\vec{w} \cdot \vec{x} + b = 0$
$\vec{u} \cdot \vec{v}$	Dot/Scalar product (vector)	$u_1v_1 + u_2v_2 + \cdots + u_nv_n$
$\mathbf{u}^{\mathrm{T}}\mathbf{v}$	Dot/Scalar product (matrix)	$u_1v_1 + u_2v_2 + \cdots + u_nv_n$
α_i, γ_i	Lagrange multipliers	
ξ_i	Slack variables	
C	Regularization parameter	
$\langle \mathbf{u}, \mathbf{v} \rangle$	Inner product	E.g. $\vec{u} \cdot \vec{v}$
$\lvert x \rvert$	Absolute value	$\sqrt{x^2}$
$\lvert\lvert \vec{u} \rvert\rvert$	Euclidean length	$\sqrt{\vec{u} \cdot \vec{u}}$
$d(\vec{u}, \vec{v})$	Euclidean distance	$\lvert\lvert \vec{u} - \vec{v} \rvert\rvert$
$\phi(\vec{x})$	Feature map	$\phi : D \to H$
$k(\mathbf{x_i}, \mathbf{x_j})$	Kernel function	$\langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle$
γ, σ	RBF Kernel Parameter	
$\mathbf{H}$	Hessian Matrix of dual SVM	$\mathbf{H_{i,j}} = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$
$\vec{s}$	Surviving features	$[1, 2, \ldots, d]$
t	Constant Step	

Contents

Chapter 1

Introduction

This bachelor thesis of the Computer Engineering Degree, specialization in Computing, has been done in the Facultat d'Informàtica de Barcelona of the Universitat Politècnica de Catalunya and directed by Luis Antonio Belanche Muñoz, doctorate in Computer Science.

1.1 Context

In statistics, machine-learning, data-mining, and other related disciplines, it is often the case that there is redundant or irrelevant data in a dataset[1]. Indeed, before we can start working with the data, some form of data analysis and cleaning is required. Data cleaning may include removing duplicated rows or rows with missing values, removing observations that are clearly out-layers, removing irrelevant variables (e.g. name, surname, email address), etc.

With the new era of Big Data, datasets have increased in size, both in number of observations and in dimensions. Applying classical data-mining and machine-learning algorithms to this high-dimensional data rises multiple issues collectively known as "the curse of dimensionality". One such issue is the elevated, usually intractable, cost and memory requirements derived from the non-linear (on number of dimensions) complexities of the algorithms. Another issue has to do with data in a high-dimensional space becoming sparse and negatively affecting the performance of algorithms designed to work in a low-dimensional space. And finally, a third issue is that with a high number of dimensions the algorithms tend to overfit, that is, they don't generalize enough and end up producing models that perform worse with real data that their predicted performance with the training data. (Li et al., 2017)

Simple manual data cleaning is not enough to achieve satisfactory amounts of dimensionality reduction. In this case we can use automatic techniques. We can classify such techniques in two categories, feature extraction, and feature selection.

1.1.1 Feature Extraction

Feature extraction techniques transform the original high-dimensional space into a new low-dimensional space by extracting or deriving information from the original features. The premise is to compress the data in order to pack the same information at the expense of model explainability[2]. Continuing with our data compression analogy, virtually all feature extraction techniques perform lossy compression, that

[1] A table with rows / records / observations and columns / variables / features / dimensions / predictors / attributes.

[2] The ability to explain why certain predictions are made. Also, interpretability.

is, some data is lost which makes the process irreversible. Notice that, if the process was reversible then feature extraction would not decrement explainability.

Some well known feature extraction algorithms include Principal Component Analysis (PCA) and auto-encoders, the first being a linear transformation over the feature-space and the second a neuronal network. PCA may be extended with a kernel method in order to make non-linear transformations. Similarly, we will also use non-linear kernels to extend SVM-RFE.

1.1.2 Feature Selection

In contrast, feature selection only selects a subset of the existing features, ideally the most relevant or useful. This may imply a greater loss of information compared to feature extraction, but it doesn't reduce explainability. Some problems require feature selection explicitly. In domains such as genetic analysis and text mining, feature selection is not necessarily used to build predictors. For example in micro-array analysis feature selection is used to identify genes (i.e. features) that discriminate between healthy and disease patients.

Feature selection methods may be classified by how they are constructed in three categories:

- **Filter:** A *feature ranking criteria* is used to sort the features in order of relevance, then select the k-most relevant.
- **Wrapper:** They use a learning machine (treated as a black box) to train and test the dataset with different subsets of variables. They rank the subsets based on the performance (score) of the model. A wide range of learning machines and search strategies can be used. Within the greedy strategies we find *forward selection* and *backward elimination*.
- **Embedded:** Like wrapper methods but more efficient. They use information from the trained model itself to make feature selection. Because they don't use the score, they can also skip testing the model.

In both wrapper and embedded methods greedy strategies can be used. **SVM-RFE** is a feature selection algorithm, of the embedded class, that uses Support Vector Machines (SVM) and a greedy strategy called Recursive Feature Elimination (RFE). This algorithm is an instance of backward elimination, it starts with a set of all features and eliminates the less relevant each iteration. Within each iteration SVM-RFE behaves like a filter method and uses a feature ranking criteria to decide which feature to eliminate. SVM-RFE takes advantage of the fact that, for linear SVM, the ranking criteria is to take the variable with the smallest weight in each iteration, where the wights are the coefficients of the hyperplane resulting from the SVM training. (Guyon and Elisseeff, 2003)

1.2 State of the art

The SVM-RFE algorithm was first proposed in a paper on the topic of cancer classification (Guyon et al., 2002). This paper uses the SVM-RFE algorithm to identify highly discriminant genes, encoded as features, that have plausible relevance to cancer diagnosis. Since then, SVM-RFE has remained a popular technique for gene selection and the original paper cited more than four thousand times.

The paper already proposes some natural extensions to the algorithm, such as eliminating multiple features each iteration based on the ranking and a *step* constant. Further research has been on improving and extending different parts of the algorithm. These include the use of a Gaussian kernel (Xue et al., 2018), using multiple SVM in the same iteration (Wang et al., 2011) or simply trying to find a better ranking criterion (Mundra and Rajapakse, 2007).

1.3 Objective

The main objective of this project is to research extensions of the SVM-RFE algorithm and try to optimize it. Optimizations may be in the form of improved performance or a reduction in time utilization. We've classified the possible extensions as follows.

Main extensions

These focus on improving the performance of the selection.

- **Non-linear kernel:** New ranking criteria needs to be found for the kernel used.
- **Multi-class criteria:** Find a ranking criterion that can handle multiple weight vectors in a useful way.

Secondary extensions

Focus on reducing the computational cost.

- **Sampling:** Use a different subset of the observations on each iteration.
- **Dynamic Step:** Instead of using a constant value in each iteration, calculate it dynamically.
- **Stop condition:** Determine the amount of features that are relevant (SVM-RFE only provides a ranking) in an effective and inexpensive manner.

Combination of various extensions

- **Combo:** Mix Sampling, Dynamic Step and Non-linear Kernels into a single implementation.

1.3.1 Objective break down

To accomplish this objective, the project has been subdivided in two parts, each having specific tasks for each extension:

Theoretical Part

- Do research in SVM-RFE and in the extensions that will be tackled in the project.
- For each extension:
  - Design the algorithm and write its formalization in pseudocode.
  - Define the expected advantages or disadvantages of this extension over the base SVM-RFE.
  - Compute the space and time complexities.

Practical Part

- Program the base SVM-RFE algorithm and the extensions.
- For each extension:
  - Analyze its behavior for artificial data sets.
  - Analyze its behavior for real-world data sets.
- Compare the results obtained with the ones expected.
- Combine multiple extensions to further improve the algorithm.
- Draw conclusions about all the results obtained in the project.

1.3.2 Stakeholders

This project is intended to be of use for many involved parties. The most directly involved group, is the tutor and the researcher. Luis Antonio Belanche Muñoz is the tutor of this project. Robert Planas Jimenez would be the researcher. Feature selection algorithms is one of the areas of research of the tutor, and he has wanted to explore extensions to the SVM-RFE algorithm. Thus, he will lead and guide the researcher for the correct development of the project. The researcher is responsible for planning, developing and documenting the project, as well as experimenting, analyzing and drawing conclusions.

The other group of interested parties would be stakeholders that do not interact with the project directly but still benefit from it. In the first place we have researchers on the fields of bioinformatics and data mining, that use machine learning methods (specifically, SVM-RFE) for micro-array analysis, text analysis, or other of its popular applications. Indirectly companies that make use of their findings will also benefit, and finally the general population may also benefit from better diagnostics and more effective drugs.

1.3.3 Potential obstacles and risks

Some obstacles and risks identified that could potentially prevent the correct execution of the project are:

- **Deadline of the project:** There is a deadline for the delivery of the project. This being a research project however, is considerably hard to estimate how much time tasks will take, or even decide whether a task has been finished or not.
- **Bugs on some libraries:** This is considered of low risk, but is still a possibility that errors on the software package used extend to code, making it work incorrectly.
- **Insufficient computational power:** Machine learning algorithms in general can be very resource intensive. It could be the case that our hardware can not handle some datasets.
- **Hardware related issues:** A hard drive failure could occur that would end in lost data, or a failure in a router could disconnect us from the internet.
- **Health related issues:** In addition to health issues that can occur at any time without prior notice, we're in the middle of a pandemic.

1.4 Methodology

1.4.1 Framework

The methodology that I will use for the project is a combination of waterfall and Kanban methodologies. Waterfall will be used to define the general phases of the project, and Kanban for tracking the individual tasks. In waterfall tasks can not start until the previous task has been completed, and thus following strict deadlines is important. Phases will be managed like this, one phase will not start until the previous phase ends. Each phase will be composed of multiple tasks, which will then be managed by a different methodology. That will be Kanban.

Kanban is much more flexible than waterfall, its principal objective is to manage tasks in a general way, by assigning different status to them. Kanban stands out by its simplicity, and we continue to endorse that simplicity by managing the visual representation of the cards in a simple plain text formatting. Each card will be in a row, with the first column defining its name and the other its status. The statuses I've considered are:

- **To do:** A basic idea of the task is present.
- **Definition:** The task is in the theoretical part.
- **Implementation:** The task is in the practical part.
- **Completed:** The task is finished.

An uppercase letter `"X"` will mark the current state of any given card. If more granular information is required, other marks may be used instead. For example, to indicate that the task is paused a `"P"` would be used. To indicate the progress within some stage a percentage would be used. This table will be kept within the `readme.md` file of the `GitHub` repository of the project, for easy monitoring.

1.4.2 Validation

We will use a GitHub repository as a tool for version control, which will allows us to share code easily and the recover from data lose. The repository will contain both the code for the different experiments, each in one subfolder, and code for the documentation. In order to verify the implemented code, it will be tested with multiple standard data sets. In the practical part, hyper-parameters will be selected using some model validation technique, such as the cross-validation. Each experiment will be done 6 times and the average of the results will be the final result. Lastly, face-to-face meetings will be scheduled once every two weeks with the tutor of the project. In these meetings it will be discussed the project status and the tasks to do during the following two weeks, before the next meeting. In case of problems in the project, extraordinary meetings can be arranged.

Chapter 2

Project Planning

This thesis is worth 15 ECTS credits, each of which with an estimate cost of 25 to 30 hours. Therefore, the total time allocated for this project, as indicated by the faculty, is of 375 to 450 hours. This time is to be distributed in 100 days, from 03/08/21 to 06/15/21, with an estimated work of 4 to 5 hours a day. The date of the oral defense is planned for the first week of July, this sets the deadline to be on the 06/18/21.

An extra 3 ECTS credits are to be used for project managment, this is roughly 80 hours, which makes the total time estimate for the whole project (Theisis + Project Managment) to be at best 450 hours and at worse 540 hours. In order to make a proper plannification, we have defined the estimated cost to be at 500 hours.

2.1 Task definition

In this section it is presented all the tasks that will be carried out along the project. For each, a description, duration and a list of dependencies with other tasks are given.

Project management is a mandatory group of such tasks, albeit not very useful, considering that: One, this project is done by a single individual, with assistance from a project director; And two, this is a research project, which makes planning of specific tasks difficult, since it is the results from the research that drive the next steps to be done.

- **Context and scope:** We have to indicate the general objective(s) of the project, contextualize it and justify the reason for selecting this subject area.
- **Project planning:** This will help us not lose focus while we're working on the project.
- **Budget and sustainability:** For this specific project, this is irrelevant. The budget required is negligible and already defrayed; and the impact, beyond trivial matters, is likely zero and otherwise unknown.
- **Final project definition:** Review the work done in the project management tasks.
- **Meetings:** Online meetings are scheduled once every two weeks with the tutor of the project. Discussion of the status and next tasks to do will be done.

This project is research focused. Therefore, before starting the practical tasks research on the various topics needs to be done. This will involve collecting and analyzing previous studies that tried new methods and extensions to the SVM-RFE algorithm. We will also have to document ourselves in the SVM and feature selection

areas, as well as the algorithms and the statistical theory used in the studies. Some basic understanding of bioinformatics may also be required considering the use case of most if these studies.

- **Research** previous work on the literature on extensions of the algorithm and create a short **report** with the findings.
- Write the algorithm formalization in pseudocode.
- Define the expected advantages or disadvantages of this method over the base SVM-RFE.
- Compute the space and time complexities.

Once the initial research is done the algorithm must be codified and tested. This group is composed of the following tasks:

- **Program the base SVM-RFE algorithm.** For this we will use a library for the SVM. For the RFE part, since we need to be able to extend the algorithm, we will program it from scratch.
- **Program the extensions of the SVM-RFE algorithm** based on the research done and the pseudocode.
- **Test the new extensions with artificial data.** This requires creating models and testing their performance with artificial data sets.
- **Obtain data sets with real data.** Multiple papers refer to publically available data-sets, we could use those and compare our results.
- **Test the new extensions with real data.** This requires creating models and testing their performance with real data sets.
- **Analyze the results** obtained in the experiments and draw conclusions. A report will be made for reference during the final documentation.

Related to testing and analyzing the results, if we want to draw fair conclusions, comparing our results with the state of the art is important. However, the start of the art is bast and fluid, and making a fair comparison with all available research papers is a daunting task. Instead, we will use a common ground, in this case the NIPS 2003 feature selection challenge. The results of this challenge, as well as a general analysis of the algorithms used, will serve as a reference point. Thankfully an analysis of these results has already been done (Guyon et al., 2004). This simplifies our problem, now we only need to compare our algorithms performance with those already presented in the challenge.

Once finished, the final documenting phase will begin. Firstly, we will collect all the information obtained in the experimental and analysis part, which will be available in the form of the reports that we've done along the tasks. Afterwards, we can start writing the documentation of the project. This will include a fairly extensive review on concepts related to SVM, statistics, feature selection and RFE. Finally, we will have to prepare for the oral defense of the project.

ID	Description	Hours	Dependencies
T1	Project Managment	80	
T1.1	Context and Scope	20	T2.1
T1.2	Project planning	10	
T1.3	Budget and sustainability	10	T1.2
T1.4	Final project definition	20	T1.1, T1.2, T1.3
T1.5	Meetings	20	
T2	Theoretical Part	160	
T2.1	Research	90	
T2.2	Formalize	20	T2.1
T2.3	Analyze	50	T2.2
T3	Practical Part	160	
T3.1	Program the base SVM-RFE algorithm	10	T2.1
T3.2	Program the extensions	50	T2.1, T3.1
T3.3	Test with artificial data	20	T3.2
T3.4	Test with real data	30	T3.2
T3.5	Analyze the results	50	T3.3, T3.4
T4	Documentation	100	
T4.1	Writte the documentation	80	T2, T3
T4.2	Prepare the thesis defense	20	T4.1

TABLE 2.1: Summary and time estimates of the tasks.

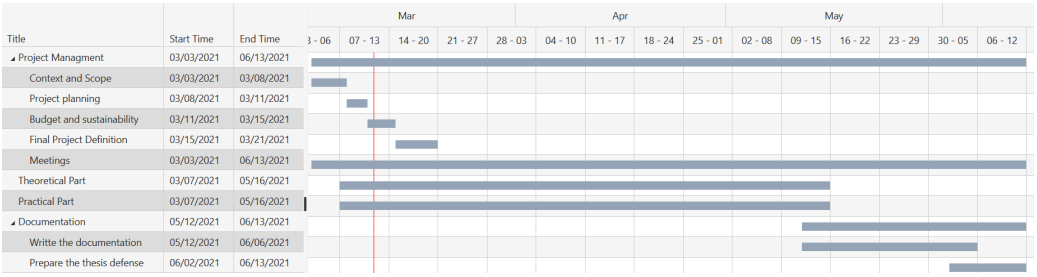


FIGURE 2.1: A summary of the tasks represented with a gantt chart. Notice that all the theoretical and practical tasks are done in parallel.

2.2 Resources

Our project needs resources to carry out its correct development. These resources have been divided in 4 different groups: human, hardware, software and material resources.

2.2.1 Human resources

There are three human resources that are directly involved in this project.

- **The researcher:** He is responsible for the development of the project, that is, he will have to plan, analyze, program, experiment and document the project.
- **The director/tutor:** He is responsible for leading and guiding the researcher for the correct development of the project.

- **The GEP tutor:** He is in charge of reviewing the project management tasks done in the initial stage of the project.

2.2.2 Hardware Resources

The most essential resource needed is a computer connected to the internet. In this project a personal computer will be used. Its specializations are 16 GB of RAM and a CPU *AMD Ryzen 7 4800HS*, with a base speed of 2.9 GHz and 8 cores. Hardware required for a connection to the internet (a router, an access point, etc) is also taken into account.

2.2.3 Software Resources

For project management tasks `Google Calendar` will be used. A number of other Google products such as `Google Mail`, `Google Drive` or `Google Meet` will also be used as tools required for communication with the director or storing of information.

The documentation will be written in LaTeX, a document preparation system often used in academia. It has the advantage to integrate well with version control systems. A LaTeX template named "`Masters/Doctoral Thesis`" will be used to facilitate the typesetting and styling of the document. This template was made by Vel and Johannes Böttcher and licensed under `LPPL 1.3`, based on previous work from Steve R. Gun and Sunil Patel and used with minor modifications. The original is available at `http://www.latextemplates.com/`. The document will be written with the `Microsft Visual Studio Code` editor, using the `LaTeX Workshop` extension, and it will be compiled with the `MiKTex` distribuition installed on a `Linux` machine running virtualized within a `WSL` container on top of the actual operating system, a `Microsoft Windows 10`. For browsing the internet the latest available version of the `Mozilla Firefox` web browser will be used, and references to papers will be kept with the `Zotero` reference manager.

For the practical part, the programming language of choice will be `Python3`. This is currently one of the programming languages with better popularity in machine learning and related applications. Various libraries and software packages will be used for different tasks. For parsing datasets the `pandas` library will be used. For data visualization the `mathplotlib` library will be used. Also, a general tool-set designed for machine learning, the library `sklearn`, will be used. Finally, code will be documented in-place during its development with the `Jupyter Notebook` software.

Other libraries and software packages could also be used if the need arises.

2.2.4 Material Resources

In a research focused project such as this, access to scientific journals, books and similar research material is needed. Some articles related to this research are freely available on the internet, but some require a subscription or single time payment. Fortunately, the UPC, the university this project is being developed at, has a subscription agreement with most of these journals and provides free access to their memebers, including the students.

2.3 Risk Management

The potential risks and obstacles have already been introduced in section 1.3.2. In this section we will focus on a contingency plan to mitigate the risks.

- **Deadline of the project:** The flexibility of the Kanban methodology should help us modify our schedule and working hours if required. If it becomes apparent that the deadline will not be meet, that is, more than 50 hours a week of work are required to finish the project in time, an extension of the deadline can be requested.
- **Bugs on some libraries:** Alternative libraries can be used. If no alternative is found, because most of the used libraries are open source, a *bugfix* could be implemented.
- **Insufficient computational power:** This can be mitigated by using a small sample of the data-set. Working with fewer data, although faster, can induce a small performance reduction and make the results not comparable with each other. Therefore, this will solution is not ideal.
- **Hardware related issues:** To avoid data loss, all code and documentation will be routinely uploaded to the cloud in a `GitHub` repository and `Google Drive` account.
- **Health related issues:** Not much can be done if a health related problem occurs, but various preventive actions, such as a good diet, exercise and resting habits can be promoted.

Chapter 3

Budget and Sustainability

In this section a budget estimation will be made. It will include personnel costs per task, generic costs and other costs. Moreover, some questions regarding the sustainability aspect of the project will be answered.

3.1 Budget

3.1.1 Costs by role and activity

In this section we will add the personnel costs to the tasks defined in section 2.1. For each task a cost will be calculated based on the hourly pay wage per role and the time spend each. Four roles have been defined with their corresponding hourly wage, these are:

- The **project manager** (T1) who is responsible for leading the project direction, planning and correct development.
- The **researcher** (T2), who must perform research in the topic and related ideas, experiment, analyze the results and draw conclusions from them.
- The **programmer** (T3), who must set up the developer environment, code the algorithms in the specified programming language and test them for correctness.
- The **technical writer** (T4), who is responsible for writing this project documentation. This includes the reports for each task that requires it and the final thesis.

These roles have a clear mapping to the task groups defined in table 2.1. All roles will be played by the researcher (see section 2.2.1) except the project manager role, which will be played by the researcher, the director and the GEP tutor. For a detailed description of each task cost see table 3.1.

Notice that, although we aligned the project roles with the task groups so that we could simplify our calculations, it is not required to do so. A more complicated scheme where multiple tasks are assigned to a given role is also possible. In such cases we would also have to think about weather the work distribution is uniform or not, and assign percentages if it isn't.

Role	Year (€)	Year +SS (€)	Hour +SS (€)	Task	Task Cost (€)
Project Manager	39 000	50 700	28.7	T1	2 296
Researcher	32 000	41 600	23.5	T2	3 760
Programmer	26 000	33 800	19.1	T3	3 065
Technical writer	22 000	28 600	16.2	T4	1 260
Total					**10 381**

TABLE 3.1: Annual estimated salary for the different project roles (*Salarios, ingresos, cohesión social* 2017). The amount of working hours in a year is defined to be 1764. *+SS indicates "Social Security included"*.

3.1.2 Generic costs

Amortization

In this section the amortization costs for the resources purchased in a single payment are calculated. Notice that since all the software used is free and open source, it doesn't contribute to the cost, thus it is not displayed here. In fact, the only resource that is valid for an amortization analysis is the computer specified in section 2.2.2.

The equation we use to compute the amortization cost for each resource is the following:

$$\text{Amortization (€)} = \text{Cost (€)} \times \frac{1}{4 \text{ years}} \times \frac{1}{100 \text{ days}} \times \frac{1}{5 \text{ hours}} \times \text{Hours Used} \quad (3.1)$$

If we apply the amortization equation to the computer, which we purchased for €999.95, and assuming 500 hours of usage, its estimated amortized cost is €249.98.

Electric cost

In this section we only calculate a rough estimate. Calculating accurately the cost of electricity involves many variables, and it's outside the scope of this thesis. The average cost of electricity in Spain, in therms of kWh, is €0.12. We only count the cost when the hardware is turned on. The following table (3.2) shows the individual and total cost per item.

Item	Power (W)	Time used (h)	Consumption (kWh)	Cost (€)
Computer	180	500	90	10.8
Router	10	500	5	0.6
Total				**11.4**

TABLE 3.2: Electric cost estimate.

Internet cost

The internet cost in my current location is €29.00 per month, this is roughly about €0.95 per day (variance is introduced because a month length is not constant). The

amount of working hours per day is assumed to be 5, thus the total cost the whole project is:

$$€0.95\ /\text{day} \times 100\ \text{days} \times 5/24\ \text{hours} = €19.79$$

Work space

This project will be developed at my parents home at Figueres, with a rent of €400. Since the amount of people living there is 2, the actual cost is €200.

Total generic costs

Table 3.3 summarizes the total generic costs of this project.

Group	Cost (€)
Amortization	249.98
Electricity	11.4
Internet	19.79
Rent	200
Total	**481.17**

TABLE 3.3: Total generic cost estimate.

3.1.3 Other costs

Contingencies

Unexpected problems that were not foreseen may appear during the development of the project, which would take part of our budget. For this reason it as always a good idea to prepare a contingency budget calculated from the budgets we've calculated up to now. We will apply a 10% contingency margin, that is €108.62.

Incidental cost

Unexpected problems that were foreseen and can be mitigated are described in section 2.3. Some of these mitigations may incur some extra cost. To be able to handle that cost in this section we will calculate a budget based on the predicable problems, their chances of actually occurring, and the expected cost associated for solving them.

- **Deadline of the project:** If an extension of the deadline is requested, that would require extra hours expend by the researcher. Assuming the extra time required to be 50 hours that would give us a cost of €1,175.
- **Bugs on some libraries:** Implementing a *bugfix* we assume would cost around 20 hours, a task done by the programmer. The estimated cost is €382. The risk is small.
- **Insufficient computational power:** If using smaller datasets where not an option, we would abandon the project, since purchasing a new, more powerful, computer or renting a supercomputer is too expensive. The risk is very small.

- **Hardware related issues:** New hardware would be purchased, if possible only the faulty component would be replaced. This would have an estimated cost of about €100. The risk is small.

The following table summarized the expected cost due to foreseen incidents.

Risk	**Expected (€)**	**Risk (%)**	**Cost (€)**
Deadline of the project	1 175	30	352.5
Bugs on some libraries	382	10	38.2
Insufficient computational power		5	
Hardware related issues	100	10	10
Total			**400.7**

TABLE 3.4: Total incidental cost estimate.

3.1.4 Total cost

The total cost for the project is summarized in table 3.5. The cost has been computed as the sum of the justified costs explained in the other sections of the budget plan.

Section	**Cost (€)**
Costs by role and activity	10 381.00
Generic costs	481.17
Other costs	509.32
Total	**11 371.49**

TABLE 3.5: Total cost estimate.

3.2 Sustainability

This project does not have a major environmental impact, since it's a research project. Still, some impact, a very small amount, is expected from the electricity consumption and hardware used. This can hardly be reduced, as is required for the development of the project. If the project succeeds at finding improvements to the SVM-RFE algorithm, it will reduce computing power requirements for future researches that use it compared to the state-of-the-art solutions. Because this algorithm is used in medical research, a substantial improvement could create a chain reaction that benefits the health of the population in general. This, however, is a very optimistic expectation.

3.2.1 Environmental dimension

Have you estimated the environmental impact of undertaking the project?

This project will not have a major environmental impact. Still, some impact, a very small amount, is present in the form of electricity consumption and the method used by the provider to generate such electricity. Also, the hardware used will eventually contribute to the technological waste problem, not to mention the environmental cost for their production.

Have you considered how to minimize the impact, for example by reusing resources?

The impact is directly derived from the project requirements and can thus not be reduced easily. A low-cost computer doesn't necessarily imply a lower impact on the environment or lower electricity consumption. Recycling is of course always an option, but the decision to do so will be taken way after the project has been completed.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution environmentally improve existing solutions?

This project will use the same resources as those used in state-of-the-art alternatives. Therefore, this solution does not environmentally improve existing solutions. If the project succeeds at finding an improvement to the SVM-RFE algorithm however, it could imply a reduction on computational power required to solve some problems, and thus also a reduction in power consumption.

3.2.2 Economic dimension

Have you estimated the cost of undertaking the project (human and material resources)?

Yes, see section 3.1.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution economically improve existing solutions?

If improvements to the SVM-RFE are found in this project, this will make any research that makes use of it less expensive and produce better results.

3.2.3 Social dimension

What do you think undertaking the project has contributed to you personally?

This is the last step required to finish my degree in computer science. Finalizing this degree will allow me to enter the work force and become self-sufficient. Beyond that, it has introduced me to the fields of bioinformatics and data mining, as well as shown the importance of feature analysis and how it can be not just a cleaning prerequisite for a more important problem, but the main dish.

How is the problem that you wish to address resolved currently (state of the art)? In what ways will your solution socially improve (quality of life) existing. Is there a real need for the project?

Improvements to the SVM-RFE algorithm may or may not be found. Even if found it is still unknown if they will be substantial enough. Moreover, an analysis of the variants of the algorithm will be useful for those who want to use SVM-RFE in the most optimal way.

Chapter 4

Background

In this section we're going to review the concepts required to understand the SVM-RFE algorithm. Section 1.1 already introduced some concepts around the SVM-RFE algorithm, placed it in context, and enumerated some of its applications. In this section we'll focus on the inner workings of the algorithm and how these parts add together.

4.1 Machine learning

Machine learning is a subfield in the broader discipline that is artificial intelligence, with a particular take in statistics. These algorithms, also called learning machines or just machines, use data to learn patterns and make predictions. The data is collected in a separated unrelated process, and structured in the form of a *dataset* (a collection of data). Once the dataset is further cleaned and prepared, the learning machine finally consumes it in a process called *training*. After this process the machine produces a *model*, which is a function that can be used to make predictions on new data.

Two canonical problems in machine learning are regression and classification problems.

4.1.1 The dataset

A dataset is simply a collection of data. In the context of machine learning this dataset will be used to make predictions, take decisions, or find patterns. For a machine learning algorithm to be able to consume a dataset the first step is to represent it in tabular form.

Most datasets come already in tabular form. Some of the most notable exceptions are datasets involving images. In this case computer vision methods are often used to extract numeric data representing characteristics of the image. Sometimes a more direct transformation can also be made, for example making each feature be the intensity level of a single pixel in the image. This of course produces a high number of features, most of which are redundant or irrelevant (e.g. features representing pixels in the background).

In a dataset represented as a table, columns describe different *features*, *properties* or *attributes* of some group of objects and rows represent *instances* or *examples* of that group. For example, if objects were vehicles then features could include the brand, power, weight, maximum speed, and other such characteristics of various vehicles, each of which would be in a row. Different names are used in different contexts. One of the most typical naming conventions comes from the statistics domain which refers to columns as *variables* and rows as *observations*. Sometimes different nomenclature is used to differentiate features in different stages of the cleaning and preparation process, but in this thesis we use them all instinctively.

Src	Dst	NAT-Src	NAT-Dst	Action	Sent (B)	Rcvd. (B)	Packets	Elapsed (sec)
57222	53	54587	53	allow	94	83	2	30
56258	3389	56258	3389	allow	1600	3168	19	17
6881	50321	43265	50321	allow	118	120	2	1199
43537	2323	0	0	deny	60	0	1	0
50002	443	45848	443	allow	6778	18580	31	16

TABLE 4.1: Example dataset extracted from the Internet Firewall Data (Ertam, 2019). Only five observations and main variables shown.

4.1.2 Classification

For the classification problem we want to predict in witch group or *class* to assign some new observation. Table 4.1 provides an example of what could be a good classification problem. Imagine we are building a firewall and want it to predict if some new packet in the network should be allowed to pass. We could train a classification learning machine with the dataset represented in the table (the full version) and produce a model capable of doing such predictions. Although it is trivial to identify a pattern in the example, it may not be so for real world examples. Automatic *pattern recognition* is key in solving many real world problems, and it is one of the features of these algorithms.

Mathematically, the set of observations is defined as $X = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}\}$ and the set of the corresponding labels or target classes is $y = \{y_1, y_2, \ldots, y_n\}$ with every element of this set being some class $y_i = C_k$ of a discrete set of classes with size K. They are called *domain set* and *label set* respectively. For our example, the classes would be $C = \{\text{allow}, \text{deny}\}$. Thus, we can describe the goal of a classification problem as assigning some class C_k to an input vector $\vec{x}$.

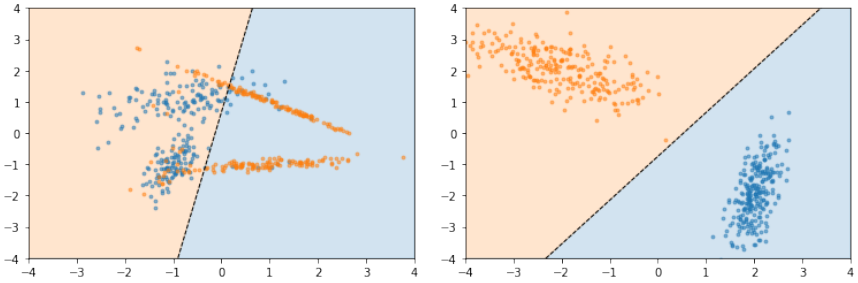
Although we've used strings to represent classes, part of the preparation process of the dataset involves turning all values into numerical scalars, so our classes would actually be some natural numbers. Also notice that every $\vec{x_i}$ is a vector containing a single numerical value for each feature excluding the label.

We make a distinction between two-class problems (or binary) and multi-class problems. Two-class problems typically use $C = \{0, 1\}$ as classes and thus can be modeled with a boolean function or, if a probability is desired, a function that returns values between 0 and 1. This simplifies the model substantially, in fact some learning machines such as the SVM can only work with two-class problems, and use different methods to extend to the multi-class version.

4.1.3 Visualization

A visualization of the problem in some euclidean space is required to understand how most classification algorithms work, and in particular SVM. Typically, classification models divide the input vector space[1] into *decision regions*. The boundaries of such regions are called *decision boundaries* or *decision surfaces*. If the decision boundary is in the form of some hyperplane of dimension $(D - 1)$, with D being the dimension of the space, then we say that it's a linear model. A dataset whose classes can be completely separated by such linear decision boundary is said to be *linearly separable*.

[1] The euclidean space of minimum dimension containing all possible input vectors $\vec{x}$.



Not linearly separable Linearly separable

FIGURE 4.1: Decision regions and 1-D hyperplane boundary of some linear model for two datasets. Points are observations, colors indicate the class.

Given a boundary we can determine in which region some new point $\vec{x}$ falls using a *discriminant function*. One of the most trivial cases is when the boundary is lineal. It is convenient now to do a small refresh on the equation that describes a hyperplane.

Equation of a line (slope-intercept form)

$$y = mx + t \tag{4.1}$$

Where m is the *slope* or *gradient*, x is the independent variable of the function $y = f(x)$ and t is the y-intercept value, the point of the function where the line crosses the y-axis, i.e. $t = f(0)$. This description has the advantage that can be directly represented as a function $f(x) = mx + t$.

Equation of a line (standard form)

$$ax + by = c \tag{4.2}$$

This is an equivalent form, this one however is not representable by a function $f : \mathbb{R} \to \mathbb{R}$, instead it is often represented as a set $L = \{(x, y) \mid ax + by = c\}$. This allows representing vertical lines and also features the useful property that (a, b) is the normal vector[2] of the line.

Equation of a line (general form)

$$ax + by - c = 0 \tag{4.3}$$

A simple linear transformation of the standard form produces the general form (*Wikipedia - Line (geometry), 2021*). This conserves the normal vector and also has the advantage of being representable by an implicit function[3]. Notice that if represented in 3-D it would produce a plane that is perpendicular to the xy plane, since its normal would always have the form $(a, b, 0)$.

[2] A vector that is perpendicular to the surface.

[3] A function of multiple variables $f : \mathbb{R}^n \to \mathbb{R}$ such that we only consider solutions where $f(X) = 0$. For example a circle can be defined with an implicit function $f(x, y) = x^2 + y^2 - r$, but if we were to plot it in 3-D we would instead see an inverted cone.

Equation of a 3-D Plane

$$ax + by + cz + d = 0 \tag{4.4}$$

We can extend the general form of a line with one more dimension, it only requires adding the new term cz for the new dimension. We've also changed the sign of the constant term d for compactness. This is a conceptual move, not an algebraic one.

Equation of a Hyperplane

$$w_1 x_1 + \cdots + w_n x_n + w_0 = 0 \tag{4.5}$$

This is a generalization of the equation of a 3-D plane for n dimensions. It also happens to be the definition of a *linear equation*. The variables $w_1, \ldots, w_n$ are called *coefficients*, *parameters* or *weights*, and the variable w_0 is the constant term. It is important to avoid confusing x_i with $\vec{x_i}$, the first is the one the coordinates of a point in some dimension, and the other is an observation of a dataset (a point). In particular, it may be the case that $\vec{x_j} = (x_1, x_2, \ldots, x_n)$.

We may want to compact this expression more by using vectors. So, another form for representing the equation of a hyperplane is:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = 0 \tag{4.6}$$

Where $\mathbf{w}$ is called the *vector weight* and w_0 the *bias*. We can turn this equation into a discriminant function for two classes by simply considering what happens with points that are not in the hyperplane. By definition such points meet one of the two inequations:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 > 0$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 < 0$$

A point will be a solution of one of these inequations depending on whether it is in the subspace, i.e. discriminant region, facing the direction of the normal or the opposite.

A linear binary classification learning machine is thus an algorithm that given some dataset finds appropriate values for the parameters $\mathbf{w}$ and w_0 of the hyperplane in order to produce a discriminant function $y : \mathbb{R}^n \to \mathbb{R}$ such as:

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 \tag{4.7}$$

4.1.4 Performance

Usually models produced by a learning machine do not always make correct predictions, instead we consider them good enough if they can classify new data correctly most of the time. In order to quantize how good of a predictor some model is we can use various performance metrics.

The most typical metric for classification problems is *accuracy*. This is the ratio of correct predictions versus the total number of predictions made. The inverse to the accuracy is defined as the *error*. Notice that the classification accuracy will always be some percentage above 50%. This is because a classifier performing consistently

worse than that can be turned into a good classifier by simply flipping the output of the discriminant function. Thus, the worst possible classifier is that of a coin toss, i.e. a uniform random distribution, with an expected accuracy of exactly 50%.

In some problems a distinction is made between misclassifications depending on which class is misclassified. An example of this is how misclassifying a patient with cancer with a healthy diagnosis is worse than misclassifying a healthy patient with a cancer diagnosis, the consequence of one is potentially death due to lack of treatment while the other is simply more investigation. For these problems a quadratic amount of cases appears. For the binary classification the standard nomenclature is to name the classes *positive* and *negative* and then prefix them with *true* or *false* depending on whether the prediction was correct or not. In this way a matrix called *confusion matrix* is created, with the diagonal containing all the correct predictions.

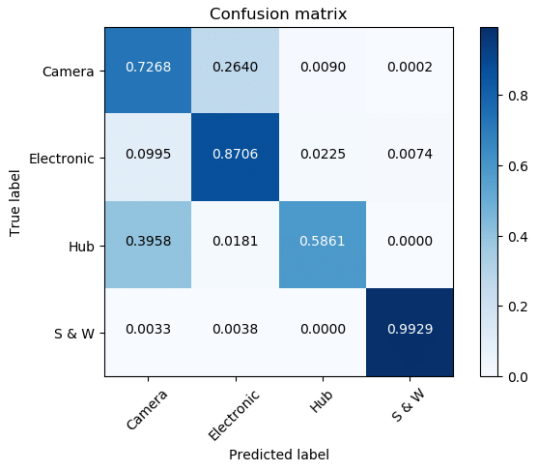


FIGURE 4.2: Confusion matrix extracted from the paper "Automatic Device Classification from Network Traffic Streams of Internet of Things" (Bai et al., 2018).

From a confusion matrix various other metrics can be extracted, such as *precision*, *specificity* or *recall*, but it is unlikely that we make use of them in this project. Often models internally use a *loss* or *cost* function that they try to minimize in order to optimize the parameters, the inverse of which is called *utility function*. Even if the model doesn't internally use one such function, one can be constructed form performance metrics, which is then referred as the *score*.

It is known that learning requires both *generalization* and *memorization*. If a model memorizes the dataset, thus has high accuracy on data already in the dataset, but doesn't generalize well, thus has low accuracy when predictions are done on new data, we say there is *overfitting*. Notice that for this to be detected we must test the dataset with new data. In order to do so a dataset is often split in *train* and *test* subsets, and although accuracies form both subsets may be reported, only the accuracy of the test subset may be taken as good.

One of the possible causes of overfitting is the *Bias-Variance* trade-off. It's an effect in which if you select a very simple model then the algorithm fails to generalize (bias) but selecting a very complex algorithm increases memorization and leads to high variability in the presence of an unseen observation (variance). The best model is thus one with a middle-ground complexity.

Selecting the complexity of an algorithm can be done with the use of some extra parameters, not fitted by the training phase of the leaning machine, that we call

hyper-parameters. The existence of this extra parameters depends on the models, some may not have any. Because these parameters modify the model, searching for the best hyper-parameters is called *model selection*. Model selection is often done in a brute force manner, by simply trying different values of the hyper-parameters and selecting the ones that give the best result. Although this coarse strategy is usually enough, more complex search strategies such as successive halving or genetic algorithms (Claesen and De Moor, 2015) can also be used. In the case of one single hyper-parameter we can draw a plot and visualize how the score evolves. Because model selection may be understood as some kind high level training, specially when complex search strategies are used, a third division on the dataset may be made, named the *validation* set.

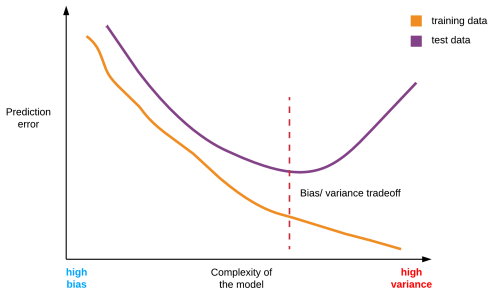


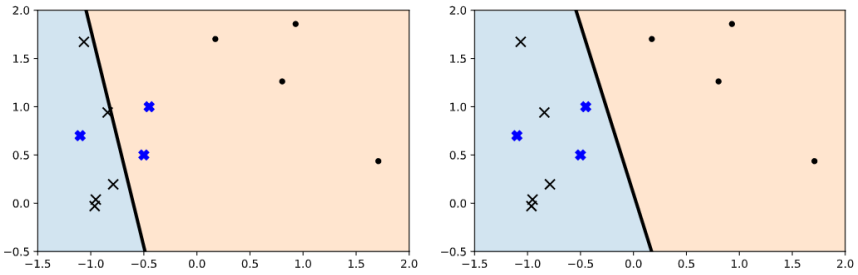
FIGURE 4.3: Bias-Variance trade-off. Source (Bisong, 2021).

Machine learning algorithms work better the more observations they are trained with, however it is not often the case that we have an unlimited amount of them. In analyzing the performance of a leaning machine, or doing model selection, it may be useful to repeat the experiments multiple times with different data of the same distribution, i.e. from the same dataset. This provides second order statistics, such as the mean or the covariance, on the resulting scores, which may prove very useful to get a better idea of what is going on. This however implies further slicing the dataset in as many slices as experiments one wants to do. After so much slicing, the remaining training subsets would often contain very few observations, thus reducing the performance of the models and nullifying the advantages of having a probability distribution on the scores.

A method called *cross-validation* can be used to make multiple experiments without increasing the amount of data. The method first makes K partitions called *folds*. For each fold, one experiment is made such that the fold is used as the validation set and the remaining folds become the training set. This allows a portion $(K-1)/K$ of the observations to be used for training in each run and allows assessing performance using the whole dataset. In the particular case where $K = N$, useful for when the amount of observations is really low, we name this method *leave-one-out* cross-validation. It is also possible to shuffle the data, which allows even more re-utilization. Although this method is computationally expensive (since it requires running the whole experiment K times), it has the advantage of being trivial to parallelize, and thus it can run at increased speeds in computers with a multi-core CPU architecture.

4.2 Support Vector Machines

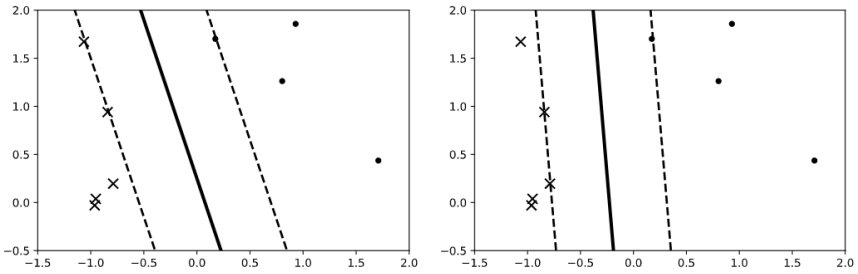
As it has been discussed in section 4.1.3, the decision boundary of a binary classification problem may be represented with a hyperplane, and it is the weights of this hyperplane what the learning machine tries to fit such that the hyperplane correctly separates the two classes. There may be infinitely many hyperplanes that correctly separate the examples of two classes, but some of them are better suited than others. For the purpose of generalization, we want decision regions that can accommodate new data outside the convex hulls[4] of each group of examples. That is, we want some kind of separation between the hyperplane and the observations.



Bad boundary Good boundary

FIGURE 4.4: Showing how one decision boundary may be better than another. `X` filled in blue represents new data.

There exist various learning machines that accomplish this separation, e.g. the *Fisher's linear discriminant*. Another such machine is the Support Vector Machine (SVM), which is based on the idea of finding the biggest margin between the extreme points of each class and setting the decision boundary in the center of such margin. For this reason SVM machines are said to be *Maximum Margin Classifiers*.



Maximum margin Quite good margin

FIGURE 4.5: Two quite good decision boundaries with a margin, a maximum margin classifier would choose the left one.

The *margin* is defined as the perpendicular distance (i.e. in the direction of the normal) between the decision boundary and the closest of the examples. There is only one decision boundary that maximizes the margin. Only a limited amount of examples, defined by the number of dimensions, will reach the limits of that margin (and thus be the closest), these are called the *support vectors*.

[4] In geometry, the *convex hull* or *convex envelope* of a shape is the smallest convex set that contains it.

4.2.1 Discriminant function

We've already seen how to we can build a discriminant function from the equation of the hyperplane. Here we will explore this in a bit more depth and take a different approx.

Given a vector of unknown length $\vec{w}$ that is perpendicular to the hyperplane (thus the normal) such that its origin is at the origin of the coordinate system. And given another vector $\vec{x}$ also with origin at the origin of the coordinate system. Notice that, by how we are defining them, they are actually points, but we may want to think of them as vectors for now. We are interested in knowing in which decision region the point $\vec{x}$ is placed.

To do so, we make the dot product between $\vec{w}$ and $\vec{x}$, which, in particular, will give us a scalar corresponding to the length of the projection of $\vec{x}$ over $\vec{w}$. Notice that for points within the hyperplane that length will be some constant value c. Thus, we can know if point $\vec{x}$ is in one region or another by comparing it with c. This can be expressed with the inequality $\vec{w} \cdot \vec{x} \geq c$. By making a variable $b = -c$ and expressing the dot product as a product of matrices, we can conveniently rearrange this equation as the decision rule:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b \geq 0 \tag{4.8}$$

This is analogous to the decision rule found in section 4.1.3, in fact if we make an equality instead of an inequality we discover the equation of the hyperplane, from there we could do the process in reverse until we find the general equation of a line.

4.2.2 Margin formalization

Notice from equation 4.8 that the constant value of b depends directly on the length $||\vec{w}||$, and because this length is not limited (there are infinite vectors that are normal to the hyperplane), an infinite amount of combinations exists.

Therefore, we may want to restrict the decision rule to a single combination. Also, it is convenient that this restriction takes such a form that allows defining the margin. First we assign to our classes the numerical values $C = \{-1, 1\}$. Then we impose the condition that points in a decision region outside the margin must have values greater than ± 1, thus for points $\mathbf{x}_{(+)}$ in class "1" and $\mathbf{x}_{(-)}$ in class "-1" we get:

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_{(+)} + b \geq 1$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_{(-)} + b \leq -1$$

This pair of equations can be simplified to a single equation by considering the vector $\vec{y}$ that we defined in section 4.1.2. Because of the numerical values we've assigned to the classes, it will contain elements such that every $y_i \in \{-1, 1\}$. Notice that if we multiply both sides by y_i (and given that we know what the value of y_i will be in each case), we can produce the single equation:

$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) \geq 1 \tag{4.9}$$

From this equation we may find the values of the hyperplanes that border the margin (named *gutter*), that is, the points where the last equation is exactly 1.

Equation of the gutters

$$y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) - 1 = 0 \tag{4.10}$$

Although now we know the equation for the gutters (analogy for the margin being a street), what we're really interested about is in the distance between them. We can calculate that distance as the difference vector form any two points in the gutters projected using the unit vector of $\vec{w}$:

$$(\vec{x}_{(+)} - \vec{x}_{(-)}) \cdot \frac{\vec{w}}{||\vec{w}||}$$

Notice that form equation 4.10 we can derive $\vec{w} \cdot \vec{x}_{(+)} = 1 - b$ and $\vec{w} \cdot \vec{x}_{(-)} = b - 1$, applying the distributive property we obtain:

$$\frac{(1-b) - (b-1)}{||\vec{w}||} \implies \frac{2}{||\vec{w}||}$$

Since we've defined the margin to be the distance from the decision boundary to a gutter (i.e. where support vectors are), and since the decision boundary is at the same distance to both gutters, then we can see that the length of the margin will be half the distance between the gutters.

Length of the margin

$$\frac{1}{||\vec{w}||} = \frac{1}{\sqrt{\mathbf{w}^{\mathrm{T}}\mathbf{w}}} \tag{4.11}$$

4.2.3 Optimization problem

The problem of finding the greatest margin can be formulated as an optimization problem. Specifically we want to maximize $1/||\vec{w}||$ subject to the constraints defined in equation 4.9. This is equivalent to minimizing $||\vec{w}||$ subject to the same constraints. For mathematical convenience (which we will see later), we can also divide by 2 and square the objective function without it affecting the optimal solution.

Primal form

$$\arg\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 \quad \text{s.t.} \quad \forall i : y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) - 1 \geq 0 \tag{4.12}$$

This is a constrained quadratic optimization problem. There are various known algorithms that can solve constrained optimization problems, such as the *simplex* algorithm. However, in our case it may be appropriate to use the *Lagrangian* method. This method doesn't directly return a solution, instead it transforms the problem into another version, from which a direct solution may be more easily computed.

Lagrangian

Imagine we had an easier optimization problem. E.g. a maximization problem such that its optimization function $f : \mathbb{R}^{D-1} \to \mathbb{R}$ is a function $f(\vec{x})$ of D dimensions,

and it has a single equality constrain $g(\vec{x}) = 0$ of $D-1$ dimensions. Notice how the constraint is "projected" on top of $f(\vec{x})$. Then the Lagrangian method consists on optimizing a new function (equation 4.13), via introducing a new parameter λ called the *Lagrangian multiplier*.

$$L(\vec{x}, \lambda) = f(x) - \lambda g(x) \tag{4.13}$$

Notice how there is only a subset of the images of $f(\vec{x})$ for which f and g intersect. Where the intersection is defined as $\{\vec{x} \in \mathbb{R}^{D-1} \mid \exists \vec{c} : f(\vec{x}) = c \wedge g(\vec{x}) = 0\}$. Many values of c contribute to the intersection, however we are only interested in the maximum or minimum values, since for these $\vec{x}$ becomes a *stationary point*.

An interesting property of these functions is that in the stationary points the direction of the normal of both functions must be the same. Note that if the normal of f was not orthogonal to the surface of g, we could increase c by moving a short distance along the constraint surface. Also remember that the normal is the same as the gradient of that function, i.e. ∇f. Therefore, there must exist a parameter $\lambda \neq 0$ (or $\lambda > 0$ for a minimization problem) such that:

$$\nabla f - \lambda \nabla g = 0 \tag{4.14}$$

From this we see that the constrained stationary condition is obtained by setting $\nabla_x L = 0$. Notice that the gradient is defined as the partial derivatives of each coordinate in the vector, that is:

$$\nabla_x L = \left(\frac{\partial L}{\partial x_1}, \frac{\partial L}{\partial x_2}, \cdots, \frac{\partial L}{\partial x_n} \right)$$

At first glance it may look overwhelming, but given that L is the same for all x_i and that we can reuse the arithmetic operators for operations between vectors, doing the gradient is very similar than doing the partial derivate over a single scalar.

Going back to the SVM formalization, we can define the Lagrangian of the primal form (equation 4.12) as:

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}||\vec{w}||^2 - \sum \alpha_i \left[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) - 1 \right] \tag{4.15}$$

Now we want to take the partial derivative with respect to $\vec{x}$ and equal to 0. Note that $||\vec{w}|| = \sqrt{\mathbf{w}^{\mathrm{T}}\mathbf{w}}$, if we elevate this expression to the power of two we can eliminate the square root. The derivative for one component is $\frac{\partial[(1/2)\mathbf{w}^{\mathrm{T}}\mathbf{w}]}{\partial w_i} = w_i$, combining them (derivative of the vector) we get $\vec{w}$. Therefore:

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum \alpha_i y_i \vec{x}_i = 0 \implies \vec{w} = \sum \alpha_i y_i \vec{x}_i \tag{4.16}$$

Also, by doing the gradient with respect to b and comparing with 0 we get:

$$\frac{\partial L}{\partial b} = -\sum \alpha_i y_i = 0 \implies \sum \alpha_i y_i = 0 \tag{4.17}$$

If we plug these expressions back in equation 4.15 we can construct the *dual form*. Note that $||\vec{w}||^2 = \mathbf{w}^{\mathrm{T}}\mathbf{w}$.

$$
\begin{aligned}
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \alpha_i \left[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) - 1\right] \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \alpha_i \left[y_i \mathbf{w}^{\mathrm{T}}\mathbf{x_i} + y_i b - 1\right] \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \left[\alpha_i y_i \mathbf{w}^{\mathrm{T}}\mathbf{x_i} + \alpha_i y_i b - \alpha_i\right] \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \alpha_i y_i \mathbf{w}^{\mathrm{T}}\mathbf{x_i} - \sum \alpha_i y_i b + \sum \alpha_i \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \alpha_i y_i \mathbf{w}^{\mathrm{T}}\mathbf{x_i} - b\sum \alpha_i y_i + \sum \alpha_i \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \sum \alpha_i y_i \mathbf{w}^{\mathrm{T}}\mathbf{x_i} + \sum \alpha_i \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \vec{w} \cdot \sum \alpha_i y_i \mathbf{x_i} + \sum \alpha_i \\
L &= \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) - \left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right) + \sum \alpha_i \\
L &= \sum \alpha_i - \frac{1}{2}\left(\sum \alpha_i y_i \vec{x_i}\right) \cdot \left(\sum \alpha_j y_j \vec{x_j}\right)
\end{aligned}
$$

Dual form

$$
\arg\max_{\vec{\alpha}} \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \quad \text{s.t.} \quad \forall \alpha_i : \alpha_i \geq 0 \wedge \sum \alpha_i y_i = 0 \tag{4.18}
$$

This variation of the optimization problem has the immediate advantage of optimizing over $\vec{\alpha}$ instead of $\vec{w}$. In cases where the number of dimensions exceeds the number of examples, solving this formulation is more efficient. Another historical reason for which the dual form has been preferred is because it allows using kernels (Chapelle, 2007), something we will see later in section 4.3.

Both this form and the primal form will require the use of a quadratic optimization solver. The general complexity of such algorithms is $O(n^3)$. However, various techniques can be used that accomplish to reduce this complexity for both the primal and the dual forms.

For this dual formulation, instead of directly finding the values of $\vec{w}$ we find the values of $\vec{\alpha}$. If we want to later calculate the values of the weight vector we can use the *representer theorem* (equation 4.16). For the constant parameter b we can use the margin boundary equation 4.10, which with some algebra can be expressed as $b = y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x_i}$.

4.2.4 Regularization

Until now, we've seen how it is desirable to maximize the margin, but we've purposely ignored the common situation where examples are not linearly separable. It could simply be that classes follow a distribution that is not separable using a hyperplane, in which case we would use kernels. But it could also be the case that although classes are close to be linearly separable there is some overlapping in their distributions.

We can solve this problem by redefining our margin as a *soft margin*, such that we allow some amount of observations to be incorrectly classified, thus falling within the margin or in the wrong side of the decision boundary, but with a penalty that

increases with the distance to correct side of the margin. To do so we introduce *slack variables*, one for each example, such that $\xi_i = 0$ if the example i is correctly classified, $0 < \xi_i < 1$ if within the margin, and $\xi_i \geq 1$ if it's in the wrong side of the margin.

Now we can reformulate our *primal* with this new variables, penalizing the sum of misclassifications and also introducing a *regularization parameter* C that allows to specify the desired trade-off strength between correct classification and slack.

Primal form with regularization

$$\arg\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\xi_i \quad \text{s.t.} \quad \forall i : y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) \geq 1 - \xi_i \quad \wedge \quad \xi_i \geq 0 \tag{4.19}$$

An equivalent way to perform regularization is by using an empirical risk minimization approx. Given our decision rule (equation 4.8), we need to find a *loss function* that works well for classification problems, such as the *hinge loss*, defined as:

$$l(t) = \max\{0, 1 - t\} \quad \text{where} \quad t = y_i f(x_i) = y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) \tag{4.20}$$

This can also be written:

$$l(t) = \begin{cases} 0 & \text{if} \quad t \geq 1 \\ 1 - t & \text{if} \quad t < 1 \end{cases}$$

For a given training set we seek to minimize the total loss, while regularizing the objective with l_2-regularization (i.e. $||\vec{w}||^2$). This gives the unconstrained regularization problem equivalent to equation 4.19:

$$\arg\min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{N}\max\{0, 1 - y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b)\} \tag{4.21}$$

Using the same Lagrangian process shown for the hard margin version, we can obtain the dual form of the soft margin version as follows:

$$L(\vec{w}, b, \vec{\xi}, \vec{\alpha}, \vec{\gamma}) = \frac{1}{2}||\vec{w}||^2 + C\sum\xi_i - \sum\alpha_i\left[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x_i} + b) - 1 + \xi_i\right] - \sum\gamma_i\xi_i$$

Dual form with regularization

$$\arg\max_{\vec{\alpha}} \sum\alpha_i - \frac{1}{2}\sum\sum\alpha_i\alpha_j y_i y_j \vec{x_i}\cdot\vec{x_j} \quad \text{s.t.} \quad \forall\alpha_i : 0 \leq \alpha_i \leq C \wedge \sum\alpha_i y_i = 0 \tag{4.22}$$

This is notably similar to the hard margin version, where only one constrain has changed.

4.3 Kernel Methods

The use of kernel functions enables learning machines such as SVM to have non-linear decision boundaries, thus allowing them to make correct predictions even if the dataset is not linearly separable (but separable nonetheless), see Figure 4.6.

4.3.1 Feature map

A feature map is a function $\phi : D \to H$ that transforms, or maps, or projects, vectors in some space D (typically $\mathbb{R}^D$) to another space H. In the context of machine learning, D is often the *input space* (the space we've been working with until now) and H the *feature space*. When no feature map is used, there is no distinction between the two.

For the sake of using kernels, we restrict feature maps to those whose range H is a *Hilbert space*. A Hilbert space is a *metric space* that defines an inner product and also is *complete* with respect to the distance function induced by that inner product (*Wikipedia - Hilbert space*, 2021).

To verify that a space is a real metric space we need to see that, for any vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ and scalars a, b:

1. The inner product is symmetric:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle$$

2. The inner product is lineal:

$$\langle a\mathbf{x} + b\mathbf{y}, \mathbf{z} \rangle = a\langle \mathbf{x}, \mathbf{z} \rangle + b\langle \mathbf{y}, \mathbf{z} \rangle$$

3. The inner product of the same element is positive definite:

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$$

   Where $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ only if $\mathbf{x}$ is neutral, i.e. $\vec{x} = (0, 0, \ldots, 0)$.
   These three properties are enough to define a *product space*, to make it also a metric space we need to add the next two.

4. The triangle inequality holds:

$$d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$$

5. The more general Cauchy–Schwarz inequality, from which the triangle inequality can actually be derived.

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq ||\mathbf{x}||\,||\mathbf{y}||$$

It is not hard to extend these properties to complex spaces, although outside the scope of this project. Finally, to see that this space is complete, we need to check that every Cauchy sequence in this space is convergent, i.e. has a limit also in the space. In other words, given a metric, such as the euclidean distance, there will always be points $\mathbf{x}$ and $\mathbf{y}$ such that for any $r > 0$ we have that $d(\mathbf{x}, \mathbf{y}) < r$. Euclidean spaces $\mathbb{R}^n$ as well as the complex space $\mathbb{C}$, and others, are examples of Hilbert spaces.

4.3.2 Kernel functions

A *similarity function* is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that uses some similarity measure (e.g. euclidean distance) to determine if two objects (e.g. points, vectors) are similar and returns a real number specifying how much. Kernel functions are a class of similarity functions for which a feature map is implicitly defined, such that:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle \tag{4.23}$$

We can compute explicitly a kernel function if we have the feature mapping defined, for instance, given:

$$\phi : \mathbb{R}^2 \to \mathbb{R}^4 \qquad \phi(x_1, x_2) = (x_1^2, x_2^2, x_1 x_2, x_1 x_2)$$

Remember that the inner product in an euclidean space $\mathbb{R}^n$ is defined as the dot product. Then the kernel function, for two points $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$ is:

$$\begin{aligned} k(\mathbf{a}, \mathbf{b}) &= \langle \phi(\mathbf{a}), \phi(\mathbf{b}) \rangle = \langle \phi(a_1, a_2), \phi(b_1, b_2) \rangle \\ &= a_1^2 b_1^2 + a_2^2 b_2^2 + a_1 a_2 b_1 b_2 + a_1 a_2 b_1 b_2 \\ &= (a_1 b_1)^2 + (a_2 b_2)^2 + 2(a_1 b_1)(a_2 b_2) \\ &= (a_1 b_1 + a_2 b_2)^2 = \langle \mathbf{a}, \mathbf{b} \rangle^2 \end{aligned}$$

Notice how a kernel function may actually simplify the computation and reduce the required amount of operations compared to actually transforming the points and then applying the inner product. In this case, for instance, we see that although the feature mapping is doubling the amount of dimensions, the kernel function that uses it can be simplified to an inner product in the domain. Thus, we can also define a kernel function without defining the feature map explicitly, e.g. $k(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle^2$. Also note that there may be multiple feature maps that result in the same kernel.

One powerful alternative technique to construct kernels implicitly is to build them out of simpler kernels as building blocks. This can be done using a set of known properties. Given valid kernels $k_1(\mathbf{a}, \mathbf{b})$ and $k_2(\mathbf{a}, \mathbf{b})$, with $\mathbf{a}, \mathbf{b} \in \mathcal{X}$, the following kernels are also valid:

$$k(\mathbf{a}, \mathbf{b}) = c k_1(\mathbf{a}, \mathbf{b}) \tag{4.24}$$
$$k(\mathbf{a}, \mathbf{b}) = f(\mathbf{a}) k_1(\mathbf{a}, \mathbf{b}) f(\mathbf{b}) \tag{4.25}$$
$$k(\mathbf{a}, \mathbf{b}) = q(k_1(\mathbf{a}, \mathbf{b})) \tag{4.26}$$
$$k(\mathbf{a}, \mathbf{b}) = \exp(k_1(\mathbf{a}, \mathbf{b})) \tag{4.27}$$
$$k(\mathbf{a}, \mathbf{b}) = k_1(\mathbf{a}, \mathbf{b}) + k_2(\mathbf{a}, \mathbf{b}) \tag{4.28}$$
$$k(\mathbf{a}, \mathbf{b}) = k_1(\mathbf{a}, \mathbf{b}) k_2(\mathbf{a}, \mathbf{b}) \tag{4.29}$$
$$k(\mathbf{a}, \mathbf{b}) = k_r(\phi(\mathbf{a}), \phi(\mathbf{b})) \tag{4.30}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial function with nonnegative coefficients, and k_r only applies to euclidean spaces $\mathbb{R}^n$ (Bishop, 2006).

Using this knowledge we can construct some of the most popular kernels:

Linear kernel

$$k(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle \tag{4.31}$$

Polynomial kernel

$$k(\mathbf{a}, \mathbf{b}) = (\langle \mathbf{a}, \mathbf{b} \rangle + c)^d \tag{4.32}$$

For some $d \in \mathbb{N}$ and $c \geq 0$, when $c = 0$ is said to be homogeneous.

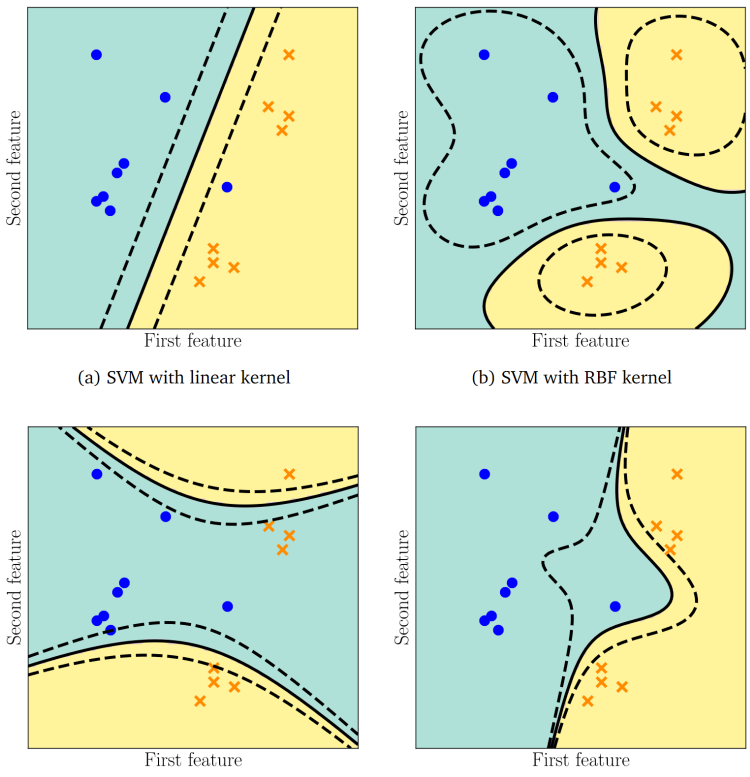
RBF (Radial Basis Function) / Gaussian / Squared Exponential Kernel

$$k(\mathbf{a}, \mathbf{b}) = \exp(-\gamma ||\mathbf{a} - \mathbf{b}||^2) \tag{4.33}$$

Where γ is a parameter that sets the "spread" of the kernel and $\exp(x) = e^x$. Recall that a Gaussian distribution has a bell-shaped curve, that is, closer points have more similarity (and thus a greater value) than more separated points. By setting $\gamma = \frac{1}{2\sigma^2}$ we can write it in the equivalent Gaussian form:

$$k(\mathbf{a}, \mathbf{b}) = \exp\left(-\frac{||\mathbf{a} - \mathbf{b}||^2}{2\sigma^2}\right) \tag{4.34}$$

This kernel has multiple interesting properties. It can be expressed as an infinite sum of polynomial kernels, this implies that the projection, i.e. the feature space, is a space with infinite dimension (Bernstein, 2017). In contrast with other kernels, where using the implicit kernel function only provides an improvement in computational cost, in this case not needing to calculate the feature map explicitly allows turning a problem that is not computable into one that can be computed trivially.



(a) SVM with linear kernel

(b) SVM with RBF kernel

(c) SVM with polynomial (degree 2) kernel

(d) SVM with polynomial (degree 3) kernel

FIGURE 4.6: Input space decision boundary of SVM with different kernels. Source (Deisenroth, Faisal, and Cheng Soon Ong, 2020).

4.3.3 The kernel trick

Recalling the SVM formulation with regularization, equation 4.19, what we want to do now is to use a feature map such that the SVM operates on the feature space instead of the input space. If we modify every instance $\mathbf{x}$ with $\phi(\mathbf{x})$, replace the dot product with the inner product, and produce the dual using the Lagrangian method over this new formulation, we will obtain:

$$\arg\max_{\vec{\alpha}} \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j \langle \phi(\vec{x_i}), \phi(\vec{x_j})\rangle \quad \text{s.t.} \quad \forall \alpha_i : 0 \leq \alpha_i \leq C \wedge \sum \alpha_i y_i = 0$$

Notice that with this new formulation we can replace the inner product with a kernel function and profit from all the advantages that kernels provide over having to explicitly compute the inner product in the feature space. This is called the "kernel trick".

SVM Dual form (kernel version)

$$\arg\max_{\vec{\alpha}} \sum \alpha_i - \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j k(\mathbf{x_i}, \mathbf{x_j}) \quad \text{s.t.} \quad \forall \alpha_i : 0 \leq \alpha_i \leq C \wedge \sum \alpha_i y_i = 0 \quad (4.35)$$

With this new version we also have the possibility to precompute all values of the kernel. We do this by defining a matrix $K_{i,j} = k(\mathbf{x_i}, \mathbf{x_j})$ called *Gram matrix* or sometimes *kernel matrix*. Because of the properties of kernels, a kernel matrix will always be symmetric and positive semi-definite, i.e. $\forall \mathbf{z} \in \mathbb{R}^n : \mathbf{z}^T K \mathbf{z} \geq 0$.

In the context of image processing, a convolution between a kernel matrix (also called *convolution matrix*) and an image is used to do all kinds of filtering, including blurring, sharpening, embossing, edge detection, and more. This specific kernel matrix is generated using all discrete points representing pixel positions (the center of cells in a grid) with an origin in the center of the grid.

Sometimes the dual SVM formulation will be expressed in terms of the kernel matrix directly. Another matrix formulation can be done by using the *Hessian* matrix, which for SVM is defined as $\mathbf{H_{i,j}} = y_i y_j k(\mathbf{x_i}, \mathbf{x_j})$. This allows expressing equation 4.35 as a cost function (to be minimized) with the formula $(1/2)\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} - \alpha^{\mathrm{T}}\mathbf{1}$, where $\mathbf{1}$ (in bold) is a vector of 1.

Another advantage of using kernels is that, since they don't restrict the input space to real numbers, you can now use SVM to classify between all kind of objects, e.g. sets, sequences, strings, graphs and distributions.

4.4 SVM-RFE

Continuing our discussion on section 1.1.2, SVM-RFE is an embedded feature selection algorithm. In this section we shall explore in more detail how and why this algorithm works.

4.4.1 Ranking criteria

One basic idea of feature selection is to use a feature ranking and then select a number r of the best ranking features as the final selection. For classification problems, one possible way to produce such a ranking is to evaluate how well a single

feature contributes to the separation of the classes, these are called *correlation filter methods* and the evaluation function *correlation coefficients*, e.g:

$$w_i = \frac{\mu_i(+) - \mu_i(-)}{\sigma_i(+) + \sigma_i(-)}$$

where μ_i and σ_i are the mean and standard deviations for the observation in the $(+)$ and $(-)$ class respectively. Large positive values of w_i indicate strong correlation with class $(+)$ and large negative values with class $(-)$. The absolute or the square of w_i can be used as the *ranking criteria*, i.e. the scores of each feature such that when sorted produce a feature ranking. Other similar correlation based ranking criterion include *Fisher*, *Pearson*, *Spearman* or *Kendall* coefficients.

Note that a feature that is correlated to some class is by itself a class predictor (albeit an imperfect one). This class predictors may be combined, either directly or with some scheme, e.g. *weighted voting*, to produce a linear discriminant classifier (Guyon et al., 2002).

$$D(\vec{x}) = \vec{w} \cdot (\vec{x} - \mu)$$

The opposite is also true, given a wight vector, e.g. one produced by an SVM, the individual coordinates of the vector can be interpreted as individual correlation coefficients. The inputs that are weighted by the largest value influence most the classification decision. Therefore, if the classifier performs well, those inputs with the largest weights correspond to the most informative features.

An alternative geometric interpretation, which leads to the same result, can also be made. By definition, the wight vector produced by a linear SVM corresponds to the normal vector of the decision boundary (a hyperplane). Setting one of the coordinates of such vector $\vec{w}_i$ to 0 will change its direction and thus produce a rotation on the hyperplane. This rotation will modify the decision regions, thus, points located in the space where the region changes (the intersection space) will be classified incorrectly (assuming no regularization). If we want to reduce the classification error a good idea is to reduce the space in the intersection, which is equivalent to reducing the amount of rotation on the hyperplane or the change in direction of the normal vector. Coordinates with a value close to 0, when the value is set to 0, will produce small changes in direction, whilst coordinates where the difference $|0 - w_i|$ is bigger will produce bigger changes. I.e, coordinates with the largest weights correspond to the most informative features (when they are removed more error is produced) and coordinates with the lowest weight are the less informative. As such, we can make a ranking criterion by simply taking the absolute value $|w_i|$, or the square $(w_i)^2$, of the coordinates of the wight vector produced by an SVM.

One problem of this interpretation is that it only works when the decision boundary is lineal. However, it gives us a hint on how to make a generalization for non-linear decision boundaries: We can use the change in objective function (or cost function) when a feature is removed as a ranking criterion. Let J be the SVM cost function $J = (1/2)\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} - \alpha^{\mathrm{T}}\mathbf{1}$. First, we train the machine and compute the $\boldsymbol{\alpha}$ parameters. Then, to compute the change in cost function $DJ(i)$ caused by removing the feature i, we leave the $\boldsymbol{\alpha}$ unchanged (for performance reasons) and recompute the matrix $\mathbf{H}$, yielding $\mathbf{H}(-i)$ where the notation $(-i)$ means that the component i has

been removed. The resulting ranking coefficient is:

$$DJ(i) = [(1/2)\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} - \alpha^{\mathrm{T}}\mathbf{1}] - [(1/2)\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}(-i)\boldsymbol{\alpha} - \alpha^{\mathrm{T}}\mathbf{1}]$$

General ranking coefficient for SVM

$$DJ(i) = (1/2)(\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}(-i)\boldsymbol{\alpha}) \tag{4.36}$$

Note that when the kernel is lineal then $\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{H}\boldsymbol{\alpha} = ||\mathbf{w}||^2$, therefore:

$$\begin{aligned} DJ(i) &= (1/2)(||\mathbf{w}||^2 - ||\mathbf{w}(-i)||^2) \\ &= (1/2)[(w_1^2 + w_2^2 + \cdots + w_D^2) - (w_1^2 + \cdots + w_{i-1}^2 + 0 + w_{i+1}^2 + \cdots + w_D^2)] \\ &= (1/2)(w_i^2) \quad \sim \quad (w_i)^2 \end{aligned}$$

Computationally, the non-lineal version is more expensive, however we may be able to optimize it by only recomputing support vectors (since $\boldsymbol{\alpha}$ is unchanged), and by caching partial results on the other components of these vectors.

There are other ranking coefficients that use the information form the model trained by some learning machine, e.g. *Relevancy and Redundancy Criteria* (Mundra and Rajapakse, 2007), or the *Span Estimate Gradient* (Rakotomamonjy, 2003).

Particularly, one method that also returns a ranking coefficient $(w_i)^2$ for the lineal case is based on the OBD algorithm (Guyon et al., 2002). It approximates the difference with a Taylor series expansion[5] to second order. At the optimum of J the first order term can be neglected, yielding:

$$DJ(i) = (1/2)\frac{\partial^2 J}{\partial w_i^2}(0 - w_i)^2 = \frac{\partial \mathbf{w}}{\partial w_i}(w_i)^2 = (w_i)^2$$

4.4.2 Recursive Feature Elimination

The criteria $DJ(i)$ estimates the effect of removing one feature at a time on the objective function. It doesn't perform particularly well when several features are removed. This problem can be overcome by using an iterative procedure, called RFE (Recursive Feature Elimination), where we eliminate one feature and retrain the learning machine each iteration.

This is an instance of backward feature elimination, and since we're using the wight vector of the learning machine in order to compute the feature ranking, we're performing an embedded method. For performance reasons it may be more efficient to remove multiple features at a time, thus we can introduce a *step* parameter t to indicate how many features to do at a time. In this case the method produces a feature subset ranking, a nested set of the form $F_1 \subset F_2 \subset \cdots \subset F$. This feature subset ranking may better be represented as a list of vectors, such that when flattened produces the final feature ranking.

Note that with this method, top ranked features are not necessarily the most relevant when taken individually, it is when taken together that features of a subset F_m are relevant in some sense.

[5] Any real or complex differentiable function can be expressed as a Taylor series, which is a polynomial of the form $f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$, where a is some point in the domain. The function can thus be approximated by restricting the polynomial to some order n.

In the following figure we present a pseudocode implementation of the SVM-RFE algorithm, which is the particular case of RFE where SVM are used. For clarity purposes this implementation is restricted to linear SVM and the associated ranking criteria, it doesn't calculate performance nor does it store individual feature subsets.

Algorithm 1: SVM-RFE

Input: t `// t = step`
Output: $\vec{r}$
Data: $X_0, \vec{y}$

1 $\vec{s} = [1, 2, \ldots, d]$ `// subset of surviving features`
2 $\vec{r} = []$ `// feature ranking list`
3 **while** $|\vec{s}| > 0$ **do**
`/* Restrict training examples to good feature indices */`
4 $X = X_0(:, \vec{s})$
`/* Train the classifier */`
5 $\vec{\alpha} =$ `SVM-train`(X, y)
`/* Compute the weight vector of dimension length` $|\vec{s}|$ `*/`
6 $\vec{w} = \sum_k \vec{\alpha}_k \vec{y}_k \vec{x}_k$
`/* Compute the ranking criteria */`
7 $\vec{c} = [(w_i)^2$ for all $i]$
`/* Find the` t `features with the smallest ranking criterion */`
8 $\vec{f} =$ `argsort`$(\vec{c})(:t)$
`/* Iterate over the feature subset */`
9 **for** $f_i \in \vec{f}$ **do**
`/* Update the feature ranking list */`
10 $\vec{r} = [\vec{s}(f_i), ...\vec{r}]$
`/* Eliminate the feature selected */`
11 $\vec{s} = [...\vec{s}(1 : f_i - 1), ...\vec{s}(f_i + 1 : |\vec{s}|)]$
12 **end**
13 **end**

Note that, although we're using vectors to store the feature ranking, by using another structure (like a dictionary in `Python`) we could store the specific feature subsets, this is actually done in our implementation.

Given that the complexity of solving the dual quadratic problem of an SVM is $O(dn^2)$, this procedure requires $O(d^2n^2)$ time to finish. Note that the whole loop of line 9 can be done in constant time with the appropriate data structures.

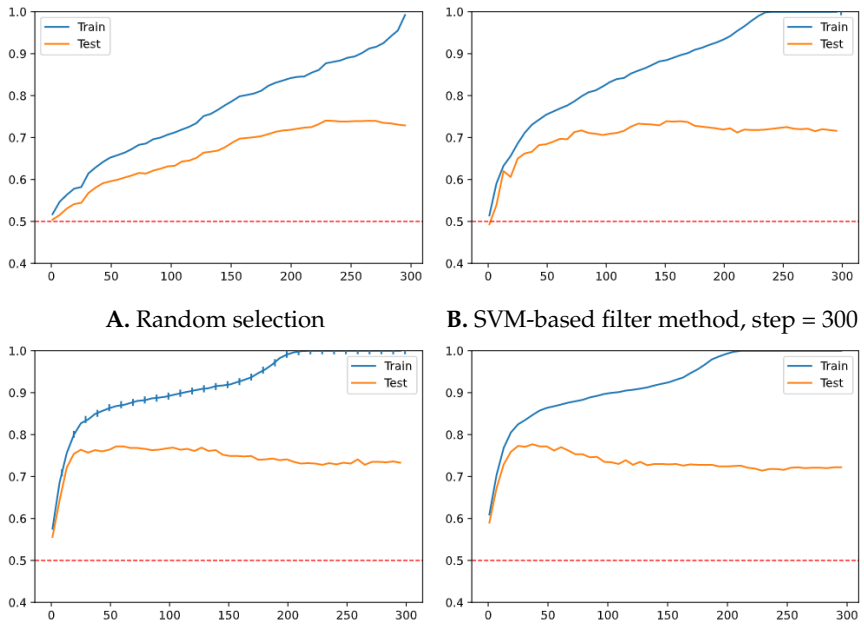
4.4.3 Assessing performance

Since learning machines will produce different levels of performance depending on the dataset, a baseline feature selection method is required in order to make proper comparisons. We can use a *random feature selection* as the baseline method. This method is a good baseline because we expect that no method should perform worse than a purely uninformed one. It simply consists in making a random feature ranking and testing the accuracy of SVM with multiple feature subsets, specifically we make each feature subset by removing t features each time based on the ranking

order. For numerical stability we may also use cross-validation and take the mean of the results. Note that, although with this method the feature ranking is made at no cost, evaluating the accuracy at each step is just as computationally expensive as SVM-RFE itself, since one SVM training has to be performed each iteration. It is expected that this method will return acceptable results when all features are used, and decrease linearly in accuracy as features are removed.

When using SVM-RFE we actually have two different methods for evaluating the performance of the selection. As an embedded method, we can evaluate the accuracy at each iteration of the algorithm, by first passing a validation set to it. If the feature subsets evaluated are of the same size as in the random selection, i.e. the same step is used, this will make a fair comparison. Another option is to use SVM-RFE purely for generating the ranking and, after that, take the same approx we've used for random selection, fit and test at each iteration. This second approx will be as computationally expensive as running SVM-RFE twice, but it allows producing performance metrics at the same intervals as the baseline method, even when the SVM-RFE algorithm uses different ones, thus making a fair comparison. In this case SVM-RFE is actually acting like a wrapper method instead of an embedded one.

The following comparison (Figure 4.7) has been made with an artificial dataset of 300 features, only 50 of which informative. Note that we provide both train and test accuracy, only the second is a valid metric of the expected accuracy on new samples, however, plotting the train accuracy is also useful to determine the overfitting as well as provide clues on the behavior of the selection algorithm.



A. Random selection

B. SVM-based filter method, step = 300

C. SVM-RFE, step = 10

D. SVM-RFE, step = 1

FIGURE 4.7: Mean accuracy using 20-fold CV of an SVM classifier with different selection methods.

As expected, random selection (plot **A**) performs significatively worse than any other method. We also verify, by comparing plots **B** and **C**, our previous assumption that the more iterations (smaller step) the better the accuracy. This relationship, however, doesn't seem to be lineal, as the same effect can not be appreciated when comparing plots **C** and **D**.

Notice that when evaluating the performance of a feature selection algorithm two variables must be considered, one is accuracy performance and the other is amount of features. A selection with a great accuracy but a lot of features may not be as good as a section with a decent accuracy and only a handful of features. That is, we want to maximize accuracy while minimizing amount of features. These two variables do not operate on the same range, and it can even be the case that the importance of one over the other is problem-dependent. This makes comparing multiple instances hard, this is why we are comparing pairs of plots instead of simply a pair of values.

Formally this is known as a multi-objective optimization problem. In this kind of problems a solution is not necessary unique, instead it is a set of *Pareto optimal*[6] solutions, i.e. solutions such that no other solution in the set of feasible solutions is better. Of course this also means that all Pareto optimal solutions are just as good. To score the solutions a *scalarization function* is used, such that the multi-objective problem becomes a single objective optimization problem. This function establishes a trade-off between the different objectives and must be chosen based on expert criteria, i.e. the user decides what is more important for the specific problem at hand.

If the criteria is known beforehand, it can be used to guide the optimization problem. In our case, for example, it could be used to guide model selection, see (Section 4.1.4). This is known as an a priori method. If all feasible solutions (or a representative subset) are first computed, this is known as a posteriori method. One typical scalarization function is the weighted sum, also called *linear scalarization*.

Linear Scalarization

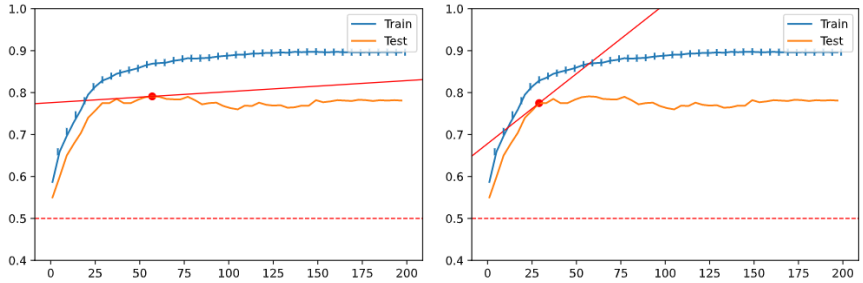
$$\arg\min_{\mathbf{x}\in\mathbf{X}} \sum_{i=1}^{k} w_i f_i(\mathbf{x}) \tag{4.37}$$

where each w_i is a weight and each $f_i(\mathbf{x})$ is a cost function (Vasumathi and S, 2019). In our case we would only have two objective functions, the accuracy at each feature subset $\texttt{Acc}(F_i)$, and the percentage of features selected $|F_i|/|F|$. Thus, the scalarization function will be of the form:

$$\arg\min_{F_i\in F} w_1(1 - \texttt{Acc}(F_i)) + w_2(|F_i|/|F|)$$

Using the wrapper form of SVM-RFE we can calculate the optimum on the validation set and plot it. This is an a posteriori method.

[6] A concept more commonly used in game theory, decision theory or economics.



$w = (0.95, 0.05)$ $w = (0.6, 0.4)$

FIGURE 4.8: Different optimums caused by changing the criteria for the linear scalarization function, projected as a red line.

Chapter 5

Experiments

This chapter is organized in sections corresponding to each of the extensions we've outlined in section 1.3. For each extension we provide a general description of the idea with the rationale behind it, a brief complexity analysis, a pseudocode, and a detailed analysis of the experimental results.

The first section of this chapter does not correspond to any extension and is instead an introductory block.

5.1 Introduction

5.1.1 General Framework

`Sklearn`, being a general machine learning library, has its own implementation of the base `RFE` algorithm. Given that it is open source, we were able to base our own implementation on it. After pruning the code to its bare minimum (no dependencies), we began to add our own extensions. We organized each extension in its own folder, isolated from the rest, and copied the base implementation there.

For the actual experiments we've used `Jupiter Notebooks` (integrated in `Visual Studio Code`). Each experiment may use different notebooks depending on what is being tested exactly. Some code that is common to all notebooks, such a code for plotting, has been placed at the beginning (for convenience). The implementation of SVM-RFE and associated usability methods has been moved to separate Python files. This is both a convenient software pattern and a requirement of the parallelization library on Windows.

We use *k-fold cross-validation* to perform the experiments multiple times with different folds, as explained in section 4.1.4. For the same experiment, the same amount of folds is used, but different amounts may be used in different experiments. We've parallelized this procedure with the standard `multiprocessing.pool` library. Given that our CPU has 8 cores, for the most computationally expensive experiments we're using 6 or 7-fold cross-validation in order to be able to finish the experiment in a single round (with one or two spare cores to be able to keep working in the machine). The execution time is always calculated as the mean of the elapsed times in every SVM-RFE execution (single core).

A limitation of Python parallelization when using Jupiter is that standard output gets suppressed. In order to debug (since debugging tools also don't work in this context) we raised exceptions, which are not suppressed because they reach the main thread. In some situations we've used other methods such a file logging or temporally running the code without parallelization.

For plotting we've used the `Mathplotlib` library. We have made a custom plot function that includes all relevant information for SVM-RFE. Having a standard plot design allows for easy comparison as shown in Figure 4.8. This plot displays the

accuracy (vertical axis) of each feature subset (by size, horizontal axis) selected by the SVM-RFE algorithm. We calculate and display both the train accuracy and the test accuracy (which is that of the validation set when cross-validation is used). We also show where the optimal is based on the linear scalarization method. Furthermore, we also show what the minimum accuracy is, based on the amount of classes, with a doted red horizontal line. And finally we show with an overlay scatter plot of small vertical lines what the actual feature selection subset sizes where and what accuracy they had during the execution of SVM-RFE (training accuracy). It is important not to confuse this plot with plots describing an iterative optimization process or model selection.

5.1.2 The data

Sklearn Generator

The general machine learning library `Sklearn` provides tools to generate artificial datasets for various problems. We're using the `make_classification` generator, which creates normally-distributed clusters of points placed at the vertices of an hypercube. Multiple clusters can corresponding to a single class. This specific generator also introduces interdependence between features and various types of noise.

We've selected this generator because it allows to specify the amount of informative (i.e. non-redundant, useful) features the dataset should have, it can generate multi-class datasets, and by changing the amount of clusters per class it can control the separability of the data.

We don't use a single dataset of this kind in our experiments. Instead, we use datasets generated with the parameters that we think can better help evaluate the expected properties and correctness of the extensions.

MADELON

This is one of the five datasets proposed for the NIPS (Neural Information Processing Systems) 2003 challenge in feature selection. The dataset remains publicly available in the UCI (University of California, Irvine) machine learning repository. The results of this challenge can be found at the workshop web page (Guyon et al., 2004).

The winner of the challenge got an accuracy of 92.89% with 8 selected features. However, this is using test data that is not publicly available. Using only the available data we've found articles describing the use of this dataset where the maximum accuracy reached is 88%.

This dataset is constructed similarly to the `sklearn make_classification` generator. It uses clusters of points placed at the vertices of some hypercube, however, instead of a single hypercube it uses five of them. Also, this method labels each in hypercube in one of two classes randomly. Five features correspond to the which hypercube a point is in, with an extra 15 being redundant features extracted from linear combinations of the first 5. The total amount of features per data point is 500, 480 of them being noise (also called probes).

Of this dataset, 2000 observations are publicly available and where used in this project, 600 are part of a separate validation set not used in this project, and 1800 are part of a test set not used in this project and also not publicly available. All sets have the same amount of positive and negative samples.

Name	Observations	Features	Informative	Classes
`make_classification`	-	-	-	-
MADELON	2000	500	5	2
Digits	1797	64	??	10

TABLE 5.1: Summary of dataset properties.

Digits

For the multi-class classification extension we've used the digits dataset (Optical Recognition of Handwritten Digits Data Set). This is also a dataset available in the UCI machine learning repository and is also provided ready to use in `Sklearn`. This dataset contains 5620 instances of 64 features corresponding to 10 possible handwritten digits. Each feature has been extracted from a 32×32 bitmap by counting the pixels of 4×4 non-overlapping blocks. There are 64 informative features.

For this dataset common accuracy results are around 98%, but we've found some cases where it can reach 100%.

5.2 Dynamic Step

This extension is based on the constant step variant of SVM-RFE (Algorithm 1), however, instead of using some constant number t as the step in each iteration, we calculate that number dynamically. The most straightforward way to do this is by using a percentage.

5.2.1 Description and reasoning

The percentage is a hyper-parameter. It is used within every iteration to eliminate a number of the least ranked features. A constant step has already been used in practice, but it is expected that this method will be significantly faster without effecting the accuracy performance, or even improving it.

Other similar modifications are also found in the literature, including using the square root of the remaining features `SQRT-RFE`, an entropy based function `E-RFE`, or `RFE-Annealing` which sets the step at $|\vec{s}|\frac{1}{i+1}$, thus, changing the percentage each iteration (Ding and Wilkins, 2006).

We assume that, the bigger the step each iteration, the worse the performance of the ranking. This is consistent with what we've seen in figure 4.7. However, since we're eliminating the worst variables first, eliminating more of them at once shouldn't affect performance because it is likely they would've been eliminated in the next iterations anyway. The fewer the iterations remaining, the riskier it becomes to eliminate multiple variables at once, and thus a smaller step is beneficial.

Our expectations for this extension are:

- **Improvement in time complexity:** Given that SVM complexity $O(dn^2)$ is linearly dependent on the amount of dimensions d, we know that each iteration is faster than the one before it. By increasing the step in the first iterations we should drastically reduce the time it takes to complete, even if then we decide to reduce the step in later, less time-consuming, iterations.

- **Improvement in accuracy performance:** Using dynamic step, compared to a constant step $t > 1$, can also improve the accuracy. This is because the last iterations may be performed with a step $t \geq i \geq 1$.

Note that by adjusting the percentage you can decide for which of these two objectives you want to optimize, it may also be possible to find a middle case in which both the accuracy and the time are improved.

5.2.2 Pseudocode formalization

Algorithm 2: SVM-RFE with DynamicStep

Input: p // p = percentage, $0 \leq p \leq 1$
Output: $\vec{r}$
Data: $X_0, \vec{y}$
$\vec{s} = [1, 2, \ldots, d]$ // subset of surviving features
$\vec{r} = []$ // feature ranking list
while $|\vec{s}| > 0$ **do**
  /* Restrict training examples to good feature indices */
  $X = X_0(:, \vec{s})$
  /* Train the classifier */
  $\vec{\alpha} =$ SVM-train(X, y)
  /* Compute the weight vector of dimension length $|\vec{s}|$ */
  $\vec{w} = \sum_k \vec{\alpha}_k \vec{y}_k \vec{x}_k$
  /* Compute the ranking criteria */
  $\vec{c} = [(w_i)^2 \text{ for all } i]$
  /* Compute t based on the percentage */
  $t = p|\vec{s}|$
  /* Find the t features with the smallest ranking criterion */
  $\vec{f} =$ argsort$(\vec{c})(:t)$
  /* Iterate over the feature subset */
  for $f_i \in \vec{f}$ **do**
    /* Update the feature ranking list */
    $\vec{r} = [\vec{s}(f_i), \ldots \vec{r}]$
    /* Eliminate the feature selected */
    $\vec{s} = [\ldots \vec{s}(1 : f_i - 1), \ldots \vec{s}(f_i + 1 : |\vec{s}|)]$
  end
end

Note that since we do a non-lineal skip focused on the iterations that take more time we can achieve a reduced complexity of $O(\log(d)dn^2)$. This can be illustrated in Figure 5.1, where we show the relationship between the amount of remaining features and iterations for the different step methods.

Note how the relative time cost of each method can be estimated as the area under the curve.

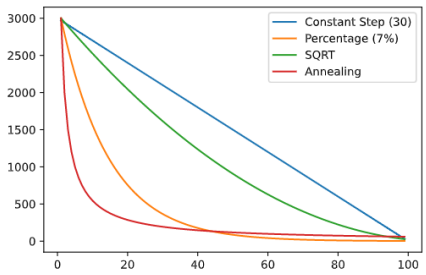


FIGURE 5.1: Amount of features remaining (vertical axis) at each iteration (horizontal axis).

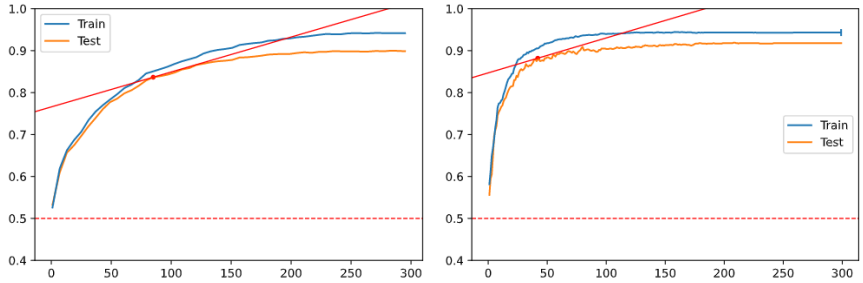
5.2.3 Results

Analysis with artificially generated data

We generate a 2 class dataset with the following code. The scalarization trade-off used is of 80% accuracy, 20% feature subset size. All results are mean values extracted from a 7-fold cross-validation procedure.

```
X, y = make_classification(
    n_samples = 1000, n_clusters_per_class=3, n_features = 300,
    n_informative = 100, n_redundant=100, n_repeated=20,
    flip_y=0.05, random_state=2, class_sep=2
)
```

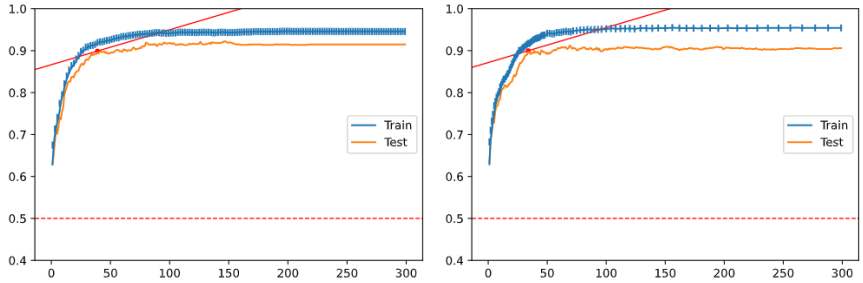
We start by comparing how the dataset performs under a random feature selection or a simple filter method. For the validation phase a linear SVM has been used, with the regularization parameter found by grid search and being $C = 0.00001$.



Random AT (85, 0.83, 0.187) Filter AT (42, 0.88, 0.122)

FIGURE 5.2: Accuracy of feature rankings produced either by a random method or an SVM-based filter method. AT (*feat.*, *acc.*, *cost*)

Based on these results we expect that SVM-RFE must perform better than the filter method. Or objective now is to find if using a dynamic step can perform similarly or better than SVM-RFE in less amount of time. From a grid search model selection procedure we'be selected the best feature rankings, shown in Figure 5.3.



Const Step $t = 2, C = 0.00001$ Dynamic Step $p = 0.05, C = 0.0001$

FIGURE 5.3: Accuracy of the best feature rankings produced by SVM-RFE with constant or dynamic step.

In the following tables we detail the results of the model selection. The cell corresponding to the best model of each algorithm (by cost) and associated time have been highlighted.

Step	**2**			**10**			**50**		
C	Feat.	Acc.	Cost	Feat.	Acc.	Cost	Feat.	Acc.	Cost
0.000001	39	88.40%	0.119	41	89.40%	0.112	38	88.70%	0.116
0.000010	39	89.90%	0.107	57	90.70%	0.112	54	89.80%	0.118
0.000100	55	91.00%	0.109	61	91.30%	0.110	59	90.10%	0.118
0.001000	81	89.40%	0.139	77	87.90%	0.148	81	87.10%	0.157
0.010000	104	88.70%	0.160	103	89.10%	0.156	116	87.90%	0.174

TABLE 5.2: Grid search of SVM-RFE with constant step.

C/Step	**2**	**10**	**50**
0.000001	0:03.691	0:00.643	0:00.124
0.000010	0:05.242	0:01.050	0:00.190
0.000100	0:07.001	0:01.440	0:00.279
0.001000	0:08.039	0:01.522	0:00.334
0.010000	0:11.834	0:02.424	0:00.577

TABLE 5.3: Execution time (min:sec.msec) of SVM-RFE with constant step.

Percentage	**0.04**			**0.12**			**0.20**		
C	Feat.	Acc.	Cost	Feat.	Acc.	Cost	Feat.	Acc.	Cost
0.000001	26	88.20%	0.112	22	86.60%	0.122	37	88.10%	0.120
0.000010	43	89.80%	0.110	36	89.80%	0.106	60	89.30%	0.126
0.000100	34	90.00%	0.103	59	91.00%	0.111	42	90.40%	0.105
0.001000	56	87.30%	0.139	63	87.20%	0.144	91	89.00%	0.149
0.010000	87	88.40%	0.151	104	87.40%	0.170	107	87.90%	0.168

TABLE 5.4: Grid search of SVM-RFE with dynamic step.

C/Percentage	0.04	0.12	0.20
0.000001	0:01.048	0:00.324	0:00.202
0.000010	0:01.455	0:00.443	0:00.283
0.000100	0:01.947	0:00.612	0:00.380
0.001000	0:02.505	0:00.709	0:00.404
0.010000	0:03.487	0:01.079	0:00.639

TABLE 5.5: Execution time (min:sec.msec) of SVM-RFE with dynamic step.

Note that the three best models for dynamic step are all better than the best model using constant step. This results however have to be taken with a grain of salt, given that variance is present and the difference is only marginal.

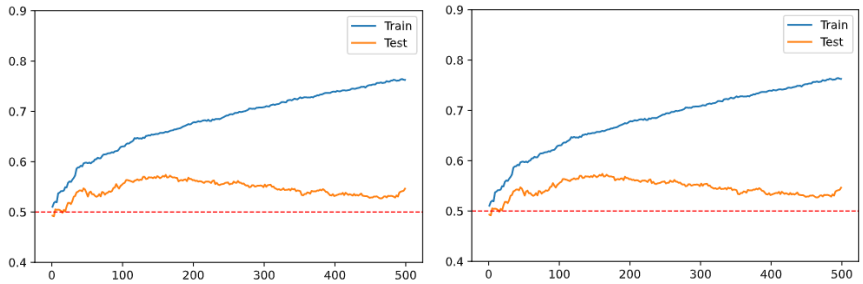
A more clear-cut improvement can be seen on the computational cost, which had a speedup of x2.7 even when the model used by dynamic step had a greater value of C. (Equation 5.2.3).

$$\text{Speedup} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{5.242}{1.947} = 2.692 \tag{5.1}$$

Initial Analysis with Madelon

We've initially run some tests to find how hard it is for the SVM-RFE algorithm to work with it.

The first test consists on simply doing a random feature selection and plotting the accuracy for the training and test splits (Figure 5.4). This random selection algorithm will allow us to make comparisons with other feature selection algorithms to determine whether they work or not. Because we're using the average of many random selection experiments, the curve it produces should be always worse than that of any other working selection algorithm. It is expected that it will produce a somewhat linear function with accuracy dropping consistently when the amount of features selected is reduced.



$C = 1$ $C = 10^3$

FIGURE 5.4: Accuracy of an SVM classifier with a random feature selection with the Madelon dataset.

The results of this test are a bit unexpected. Although the training accuracy behaves as expected, the test accuracy is way too low for even the case where all features are used (0.55). Reducing the amount of features, even randomly, produces a slight increase in the test accuracy. This is a fine example of the effects of the curse of dimensionality, on how even a random selection, which should produce

consistently worse models the less amount of features it has, ends up producing better ones simply because there are fewer features with independence on how the feature selection is made.

We've performed a second test to see if the regularization parameter has any effect and can be of any help in trying to reduce overfitting (Figure 5.5).

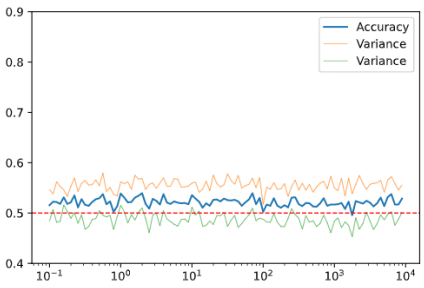


FIGURE 5.5: Test accuracy mean of SVM classifier with 10-fold cross-validation using all features for multiple values of C.

As the plot indicates, no value of the regularization parameter helps improve the performance. These poor results suggest that the dataset is not linearly separable and thus is producing huge amounts of overfitting in all cases, but, how does this affect SVM-RFE?

To see how SVM-RFE performs under these conditions we've designed an experiment very similar to that used in random selection, although instead of using 10 random selections we've used cross-validation to take the mean. In this experiment we also use a constant step of 10 for performance reasons, see Figure 5.6.

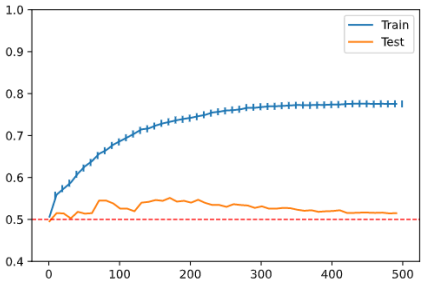
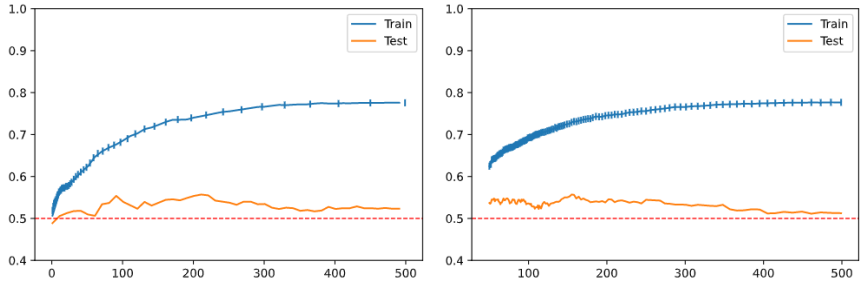


FIGURE 5.6: Mean accuracy using 20-fold CV of an SVM classifier with SVM-RFE selection of the Madelon dataset.

Comparing this plot with that of the random selection we can see that the results are quite disappointing. Although SVM-RFE does improve the performance on the training data, the test performance remains the same. With this we can conclude that SVM-RFE using a lineal kernel does not work well on this specific dataset, and thus, no matter the step strategy used, the results will all be quite bad.

We have also checked how does SVM-RFE perform if we use a dynamic step strategy (Figure 5.7), but no significant difference is appreciated.



10%, 52 iterations, stop at 0 3%, 136 iterations, stop at 50

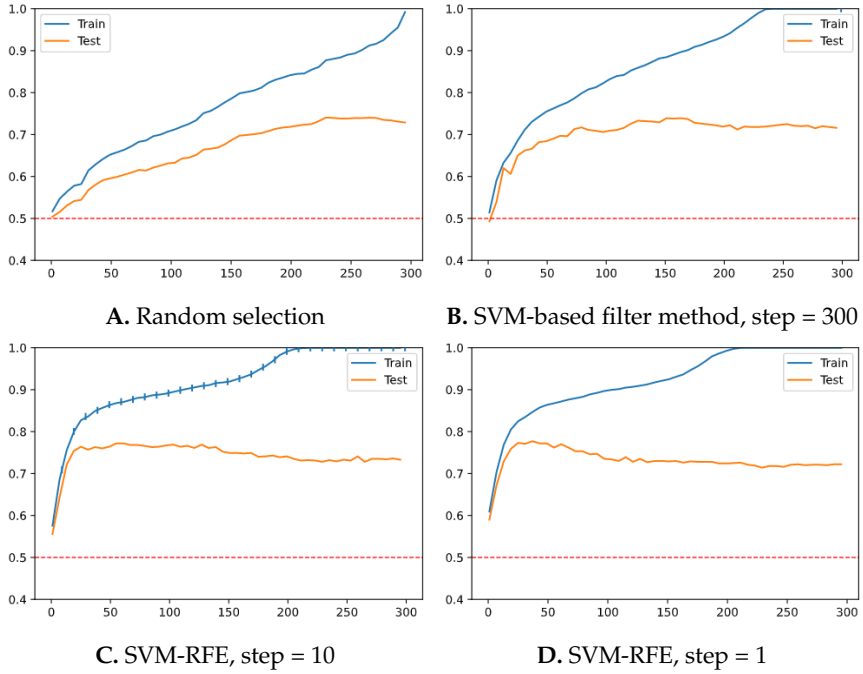
FIGURE 5.7: Mean accuracy using 20-fold CV of an SVM classifier with SVM-RFE selection using dynamic step of the Madelon dataset.

This is a problem. We've initially wanted to use the Madelon dataset to do our experiments, but these results show that the dataset in not adequate for a linear classifier. Thus, we've decided to use simpler artificially generated datasets for this experiment and come back to the Madelon dataset later on with other extensions of the SVM-RFE algorithm.

Initial Analysis with artificially generated data

To generate the dataset we've used the *make_classification* function of the `Sklearn` library. We've created a dataset with 300 features, 50 of which are informative. Ideally, a perfect feature selection algorithm should be able to produce a peak test accuracy at the exact amount of informative features or less.

Similar to the experiments descried before, we've performed random selection and SVM-RFE. Here tough, we also make a comparison with different values of the step to see how it effects the actual performance of the selection. In the particular case where the step is equal to the amount of features this is no longer SVM-RFE but a filter method based on SVM (one single iteration).



A. Random selection

B. SVM-based filter method, step = 300

C. SVM-RFE, step = 10

D. SVM-RFE, step = 1

FIGURE 5.8: Mean accuracy using 20-fold CV of an SVM classifier with different selection methods.

From this analysis we conclude that SVM-RFE works well on this dataset. We also verify, by comparing plots **B** and **C**, our previous assumption that the more iterations (smaller step) the better the accuracy. This relationship, however, doesn't seem to be lineal, as the same effect can not be appreciated when comparing plots **C** and **D**.

Notice that when evaluating the performance of a feature selection algorithm two variables must be considered, one is accuracy performance and the other is amount of features. A selection with a great accuracy but a lot of features may not be as good as a section with a decent accuracy and only a handful of features. That is, we want to maximize accuracy while minimizing amount of features. These two variables do not operate on the same range, and it can even be the case that the importance of one over the other is problem-dependent. This makes comparing multiple instances hard, this is why are comparing pairs of plots instead of simply a pair of values.

Dynamic Step vs Constant Step

Based on our assumptions, dynamic step can never beat in accuracy performance a constant step of one. Beating constant steps bigger than 1 should theoretically be possible when the amount of features is huge, since at some point the specific step used will be smaller than the constant, and there an improvement may be found. For our specific dataset of 300 features however, no such improvement was found, likely because the amount of features was insufficient, but also because of the no linearity of the stated property.

Rather than for improving accuracy, dynamic step is more useful for improving computational cost. Because each iteration reduces the amount of features, training the SVM will be faster with dynamic step even if the same amount of iterations are used, since there will be more iterations using fewer features in the dynamic step case.

This same mechanic does not apply only to using percentages, the other methods stated at section 5.2.1 follow the same logic. In Figure 5.9 you can see a comparison of dynamic step methods. Although all of them draw some kind of curve, they don't use the same shape, and thus can produce slightly different performance results using the same amount of iterations. The time cost of the methods can be estimated as the area under the curve.

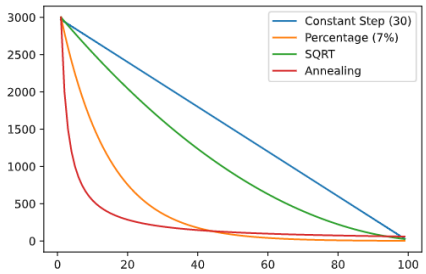


FIGURE 5.9: Amount of features remaining (vertical axis) at each iteration (horizontal axis).

Experimentally we confirm our results and see that using a percentage is faster compared to a constant step, without reducing its accuracy. The following table summarizes the time cost for various experiments. The SVM-RFE time shown is the mean of the times of every fold used during cross-validation.

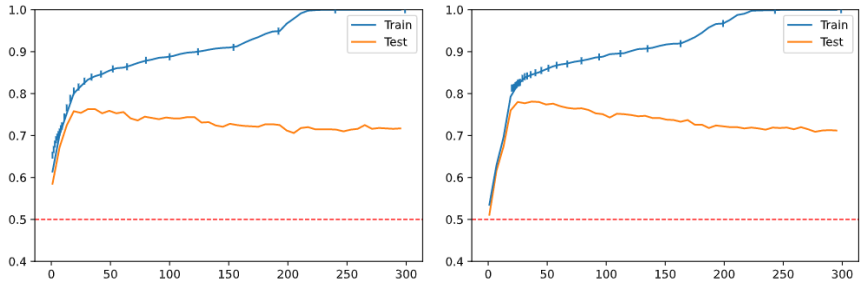
Strategy	**Iterations**	**Time**	**Total Time**
Constant	60	29.16s	00:02:36
Constant	30	15.90s	00:02:00
Constant	6	4.43s	00:01:29
Constant	3	2.74s	00:01:22
Percentage	47	7.03s	00:01:29
Percentage	26	3.83s	00:01:17
Percentage	10	2.59s	00:01:13

TABLE 5.6: Time costs for various experiments.

Notice that although one could think that the smaller the area under the curve the better, and it is true when it comes to time costs, a very pronounced curve would imply a very big initial step, this can result in a drop in accuracy performance in the very first iterations that can not be restored later on even with a step of 1. This is specially important for the annealing method, since it chooses a 50% percentage in the very first iteration. How much acute the curve can be without producing a drop in accuracy depends on the dataset, specifically on the amount of informative features. Thus, introducing a hyperparameter to specify the sharpness of the curve may be useful. The percentage approx already has this hyperparameter by default, but introducing such parameter in the other methods is also trivial. Another option would be to further customize the curve by introducing minimum and maximum steps.

Finally, one last parameter that may be of relevance is the stop value. If the amount of informative variables is known, it doesn't make sense to perform many iterations on selections so small that the performance has already dropped, instead

it may make more sense to increase the amount of iterations the closer you get to the amount of informative variables. This effect can be seen in Figure



2%, 26 iterations, stop at 0 2%, 25 iterations, stop at 20

FIGURE 5.10: Comparison of the dynamic step behavior with and without using a stop parameter.

5.2.4 Annotations

- The Cross Validation procedure has been implemented by hand and parallelized.
- How do I know which method is better?
- How do I measure how good some method is (accuracy vs selection size)? Mathematically.
- Should I do hyperparameter search (of C) also here?
- Explain why plotting takes so much time.
- Should the percentage really be a hyperparameter? Annealing and Square Root do not, and it doesn't make much of a difference anyways?
- The complexity of SVM seems to be $O(\max(n,d)\min(n,d)^2)$.
  If dimensions > nº observations: $O(dn^2)$.
  Otherwise: $O(nd^2)$.

5.3 Sampling

5.4 Stop Condition

5.5 Multi-Class

In this section we extend SVM-RFE to the multi-class classification problem.

5.5.1 Description and reasoning

When it comes to extending SVM to handle a multi-class problems two common methods exists, OvR (One-vs-Rest) and OvO (One-vs-One). In both cases the idea is to divide the problem in a set of binary classification problems and use a joint decision function that operates on the results of each of these. Because we're not really making any predictions during the SVM-RFE procedure, we can not use this

joint decision function. Instead, we must find a way to merge the ranking criteria obtained form each problem to find a joint ranking criteria.

We know that the ranking criteria is an estimator of the importance of some feature for a given binary decision problem. It can be the case that a feature is very useful to distinguish between two classes but useless for the rest. In this case a joint ranking criteria formed by taking the mean, the median or the sum will result in poor selections. A better idea would be to take the maximum. However, it may also be desirable to estimate the joint importance a feature has by considering its individual importance in more than one problem, that is, a feature that is important in more than one binary classification problem is more important than another that is only important in one such classification problem even if the second feature has a greater individual importance. A way to perform such ranking would be, for instance, the sum of the squares. For both methods proposed a normalization of all feature rankings is probably adequate.

Note that `sklearn` only supports OvO, and is therefore the option we will use.

5.5.2 Pseudocode formalization

Definitions:

- $X_0 = [\vec{x_0}, \vec{x_1}, \ldots, \vec{x_k}]^T$ list of observations.
- $\vec{y} = [y_1, y_2, \ldots, y_k]^T$ list of labels.

Algorithm 3: SVM-RFE for multi-class classification problems

Input: t // t = step
Output: $\vec{r}$
Data: $X_0, \vec{y}$

$\vec{s} = [1, 2, \ldots, n]$ // subset of surviving features
$\vec{r} = []$ // feature ranked list
while $|\vec{s}| > 0$ **do**
  /* Restrict training examples to good feature indices */
  $X = X_0(:, \vec{s})$
  /* Compute the joint ranking criteria */
  $\vec{c} = [0, 0, \ldots,]$
  for $\vec{Xl} \subseteq \vec{X}$, $\vec{yl} \subseteq \vec{y}$ *with* $\vec{Xl}$ *and* $\vec{yl}$ *being an instance of OvO* **do**
    /* Train the classifier */
    $\vec{\alpha} =$ `SVM-train`$(\vec{Xl}, \vec{yl})$
    /* Compute the weight vector of dimension length $|\vec{s}|$ */
    $\vec{w} = \sum_k \vec{\alpha_k} \vec{yl_k} \vec{Xl_k}$
    /* Compute the joint ranking criteria */
    $\vec{c} = [\max(c_i, (w_i)^2) \text{ for all } i]$
  end
  /* Find the t features with the smallest ranking criterion */
  $\vec{f} =$ `argsort`$(\vec{c})(:t)$
  /* Iterate over the feature subset */
  for $f_i \in \vec{f}$ **do**
    /* Update the feature ranking list */
    $\vec{r} = [\vec{s}(f_i), ...\vec{r}]$
    /* Eliminate the feature selected */
    $\vec{s} = [...\vec{s}(1 : f_i - 1), ...\vec{s}(f_i + 1 : |\vec{s}|)]$
  end
end

5.5.3 Results

5.6 Non-linear Kernels

5.7 Combo

—

In this section

- Importance of scaling preprocessing
- Complexity analysis and pseudocode.

Bibliography

Bai, Lei et al. (Oct. 2018). "Automatic Device Classification from Network Traffic Streams of Internet of Things". In: pp. 1–9.

Bernstein, Matthew N. (Mar. 2017). *The Radial Basis Function Kernel*. URL: http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf (visited on 04/27/2021).

Bishop, Christopher (2006). *Pattern Recognition and Machine Learning*. Microsoft Research Ltd. ISBN: 0-387-31073-8.

Bisong, Ekaba (2021). *Machine Learning: An Overview*. URL: https://ekababisong.org//ieee-ompi-workshop/ml-overview/ (visited on 04/17/2021).

Chapelle, Olivier (May 2007). "Training a Support Vector Machine in the Primal". In: *Neural Computation* 19.5, pp. 1155–1178. ISSN: 0899-7667. URL: https://doi.org/10.1162/neco.2007.19.5.1155 (visited on 04/19/2021).

Claesen, Marc and Bart De Moor (Apr. 2015). "Hyperparameter Search in Machine Learning". In: *arXiv:1502.02127 [cs, stat]*. arXiv: 1502.02127. URL: http://arxiv.org/abs/1502.02127 (visited on 04/18/2021).

Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong (2020). *Mathematics For Machine Learning*. Cambridge University Press.

Ding, Yuanyuan and Dawn Wilkins (Sept. 2006). "Improving the Performance of SVM-RFE to Select Genes in Microarray Data". en. In: *BMC Bioinformatics* 7.2, S12. ISSN: 1471-2105. URL: https://doi.org/10.1186/1471-2105-7-S2-S12 (visited on 03/28/2021).

Ertam, Fatih (Feb. 2019). *Internet Firewall Data Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/Internet+Firewall+Data (visited on 03/25/2021).

Guyon, Isabelle and André Elisseeff (2003). "An Introduction to Variable and Feature Selection". In: *Journal of Machine Learning Research* 3.Mar, pp. 1157–1182. ISSN: 1533-7928.

Guyon, Isabelle et al. (Jan. 2002). "Gene Selection for Cancer Classification using Support Vector Machines". en. In: *Machine Learning* 46.1, pp. 389–422. ISSN: 1573-0565.

Guyon, Isabelle et al. (Jan. 2004). "Result Analysis of the NIPS 2003 Feature Selection Challenge". In: vol. 17.

Li, Jundong et al. (Dec. 2017). "Feature Selection: A Data Perspective". In: *ACM Comput. Surv.* 50.6, 94:1–94:45. ISSN: 0360-0300. URL: https://doi.org/10.1145/3136625 (visited on 03/01/2021).

Mundra, Piyushkumar A. and Jagath C. Rajapakse (2007). "SVM-RFE with Relevancy and Redundancy Criteria for Gene Selection". en. In: *Pattern Recognition in Bioinformatics*. Ed. by Jagath C. Rajapakse, Bertil Schmidt, and Gwenn Volkert. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 242–252. ISBN: 978-3-540-75286-8.

Rakotomamonjy, Alain (Mar. 2003). "Variable selection using svm based criteria". In: *J. Mach. Learn. Res.* 3.null, pp. 1357–1370. ISSN: 1532-4435.

Salarios, ingresos, cohesión social (2017). Tech. rep. Encuestas de Estructura Salarial. INE. URL: https://www.ine.es/jaxiT3/Tabla.htm?t=10911.

Vasumathi, D. and Thangavelu S (Mar. 2019). "Scalarizing functions in solving multi-objective problem-an evolutionary approach". en. In: *Indonesian Journal of Electrical Engineering and Computer Science* 13.3. Number: 3, pp. 974–981. ISSN: 2502-4760. URL: http://ijeecs.iaescore.com/index.php/IJEECS/article/view/14164 (visited on 04/30/2021).

Wang, Jingjing et al. (Nov. 2011). "Classification of lip color based on multiple SVM-RFE". In: pp. 769–772.

Wikipedia - Hilbert space, (Apr. 2021). en. Page Version ID: 1017590047. URL: https://en.wikipedia.org/w/index.php?title=Hilbert_space&oldid=1017590047 (visited on 04/26/2021).

Wikipedia - Line (geometry), (Apr. 2021). en. Page Version ID: 1010000668. URL: https://en.wikipedia.org/w/index.php?title=Line_(geometry)&oldid=1010000668 (visited on 04/17/2021).

Xue, Yangtao et al. (Oct. 2018). "Nonlinear feature selection using Gaussian kernel SVM-RFE for fault diagnosis". en. In: *Appl Intell* 48.10, pp. 3306–3331. ISSN: 1573-7497. URL: https://doi.org/10.1007/s10489-018-1140-3 (visited on 03/18/2021).