Master of Data Science
Machine Learning and Optimization laboratory
Master Project

High-dimensional knockoff feature selection

Scalable false discovery rate control

School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne
Master thesis of

Quentin Rebjock

Supervised by

Prof. Alexandre d'Aspremont (Sierra team, Inria, ENS, CNRS)
Prof. Martin Jaggi (Machine Learning and Optimization lab, EPFL)

Paris, Inria, March 2020

Abstract

High-dimensional feature selection has become a common task in many sciences over the past decades because of advancements in data acquisition. It is now possible to record dozens of thousands of features at low cost, and it is crucial to find out later which of them are of interest. In this master thesis, we focus on false discovery rate control, that is the problem of making as many relevant selections as possible while maintaining the number of false positives under a certain threshold.

Recently **?** introduced the knockoffs framework to control the false discovery rate when performing feature selection. The key idea is to build a knockoff (fake) feature for each original feature, and compare their significance against a statistical model. Despite the theoretical possibility to employ this framework in high-dimension, several bottlenecks make it unemployable in practice. We take advantage of low-rank approximations to efficiently build knockoff features in high dimension, and show that a reasonable statistical power can be preserved by using accelerated significance statistics. We apply these techniques to genetic data and perform FDR control on $\approx 30\,000$ features in less than a minute on a standard workstation.

Keywords: Feature selection, false discovery rate control, knockoff filter, high dimension

Acknowledgements

I am extremely grateful to my master thesis advisor Alexandre d'Aspremont for his relevant insights and his patience. I would also like to thank Armin Askari and Laurent El Ghaoui with whom I had the opportunity to work, and who provided substantial help for the resolution of SDP problems. More generally, all the people I met at Sierra and Willow contributed to make this experience pleasant and valuable, and I would like to recognize their assistance. I also very much appreciate the support of my EPFL advisor Martin Jaggi. Finally, I cannot leave EPFL without mentioning my parents who encouraged me throughout my studies.

Contents

Abstract i

Acknowledgements ii

Notations v

Introduction 1

1 Feature selection 3
- 1.1 Background on feature selection . . . 3
  - 1.1.1 Definitions . . . 3
  - 1.1.2 Motivation . . . 4
  - 1.1.3 Techniques . . . 4
    - Lasso . . . 5
    - Sparse center classifiers . . . 5
- 1.2 False discovery rate control . . . 5
  - 1.2.1 Definitions . . . 5
  - 1.2.2 Benjamini–Hochberg-Yekutieli procedures . . . 6
    - Benjamini–Hochberg . . . 6
    - Benjamini–Yekutieli . . . 6

2 The knockoffs filter framework 9
- 2.1 Knockoffs construction . . . 9
  - 2.1.1 General case . . . 9
  - 2.1.2 Gaussian case . . . 10
- 2.2 Statistics computation . . . 11
  - 2.2.1 General principle . . . 11
  - 2.2.2 Statistics aggregation . . . 11
  - 2.2.3 Examples . . . 11
- 2.3 Selection thresholds . . . 12
- 2.4 Bottlenecks . . . 12
  - 2.4.1 Knockoffs construction . . . 13
  - 2.4.2 Statistics computation . . . 13
- 2.5 Python implementation . . . 13

3 Efficient knockoffs construction 15
- 3.1 Equi-correlated knockoffs . . . 15
    - A cheap solution . . . 15
    - Why it is not desirable . . . 15
- 3.2 SDP knockoffs . . . 16
- 3.3 A coordinate ascent approach . . . 17
  - 3.3.1 Notations and preliminaries . . . 17
  - 3.3.2 Coordinate ascent . . . 18
  - 3.3.3 Log-barrier . . . 18
- 3.4 Low-rank covariance approximation . . . 19
  - 3.4.1 Factor model . . . 19
  - 3.4.2 Low-rank coordinate ascent . . . 20
  - 3.4.3 Efficient low-rank sampling . . . 21

4 Fast statistics computation **25**
4.1 Lasso is good but slow . . . 25
4.2 Multi-stage procedures . . . 25
4.3 Sparse naive Bayes . . . 25
4.3.1 Reminders on vanilla naive Bayes . . . 26
4.3.2 Sparse naive Bayes (SNB) . . . 28
4.4 Other fast statistics . . . 29
4.4.1 Based on the Lasso Signed Max . . . 29

5 Experiments **31**
5.1 Setup . . . 31
5.1.1 Genetic data . . . 31
5.2 Results . . . 31
5.2.1 Applications . . . 31
5.3 Criteo dataset . . . 31
Genetic and fMRI data . . . 32

Conclusion **33**

Appendices **35**

A Data generation details **37**
A.1 SCS and MOSEK convergence times . . . 37
A.2 Coordinate ascent . . . 37

B Linear algebra notes **39**
B.1 Schur complement and positive definiteness . . . 39
B.2 Sherman–Morrison–Woodbury formula . . . 40
B.3 Block matrix determinant . . . 40

C Cython acceleration **41**

Notations

- $\mathbb{N}$ denotes the set of natural integers and $\mathbb{R}$ the field of real numbers.
- Vectors are written in bold letters, e.g. $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^p$.
- For a vector $\mathbf{v}$, we note $\|\mathbf{v}\|_0$ the number of non-zero entries of $\mathbf{v}$ (that is, its cardinality).
- Matrices are written in capital letters, e.g. $X \in \mathbb{R}^{n \times p}$.
- Vectors $\mathbf{x} \in \mathbb{R}^p$ are by default considered to be columns matrices in $\mathbb{R}^{p \times 1}$ with the same size and entry values.
- $\mathbf{0}$ and $\mathbf{1}$ are vectors (or sometimes matrices) whose entries are all 0 and 1 respectively, and whose sizes are inferred by the context.
- We note $\odot$ the Hadamard (element-wise) product between two vectors or matrices.
- Given two matrices A and B, we note $\left[A, B\right]$ and $\left[A; B\right]$ their horizontal and vertical concatenation respectively (if dimensions match).
- The diag operator may be used in two contexts; either to transform a vector into a diagonal matrix whose diagonal entries match the vector, or to extract the diagonal of a matrix. For example,

$$\operatorname{diag}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}, \qquad \operatorname{diag}\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}1\\4\end{bmatrix}$$

- The sets of $n \times n$ symmetric, symmetric positive semidefinite, and symmetric positive definite matrices are denoted by S^n, S^n_+ and S^n_{++} respectively.
- Functions $\mathbb{R} \to \mathbb{R}$ applied to vectors or matrices are implicitly performed element-wise, unless otherwise specified.
- $\mathbb{E}\left[\mathcal{X}\right]$ denotes the expectation of a random variable (or vector) $\mathcal{X}$.
- Given two random variables (or vectors) $\mathcal{X}$, $\mathcal{Y}$, we note $\mathcal{X} \perp\!\!\!\perp \mathcal{Y}$ the fact that they are independent.
- Given a set $\mathcal{S}$, $|\mathcal{S}|$ and $\#\mathcal{S}$ denote its cardinality (number of elements).

Introduction

Feature selection is an active area of research in statistics and machine learning aiming, among others, at making models more interpretable. The idea is to identify the most relevant observed covariates that best explain a phenomenon. Often, one is interested in selecting as many pertinent features (true positives) as possible, that is, maximizing true discoveries, while maintaining the number of false positives under a certain threshold. This problem is called false discovery rate control and finds applications in many sciences, such as biology, economy, medicine and many others. This is why a multitude of techniques have been developed to this end, the most widely used being the one proposed by **?**. However, in the past two decades, acquiring massive amounts of data has become increasingly cheaper and it is now frequent that datasets contain several dozens of thousands of features. A common practice is to measure as many variables as possible, and figure out afterwards which of them prove to be valuable. On top of that, measuring covariates is often simpler than labeling samples with the right category, as the latter requires human efforts and can hardly be automatized. For that reason, many datasets have more features than samples and it is especially the case in some fields like biology. **?** For instance, genomic and fMRI data contain a huge amount of features but relatively low number of patients. In this high dimension setup, feature selection and false discovery rate control are particularly challenging because estimations based on samples suffer from a high variance.

Recently **?** introduced the knockoffs framework, which is a set of procedures capable of performing feature selection, and that are proven to theoretically control the false discovery rate under mere assumptions. The key idea is to build a knockoff (that is, fake) feature for each authentic feature, and compare their significance against a statistical model. This framework was revisited and extended to a more general inference setting [**?**], allowing in particular to perform feature selection in the high dimension case, that is when the number of features is larger than the number of samples. The generality of this selection procedure and the fact that many statistical models can be plugged in make it particularly appealing. However, the construction of the knockoff features and the computation of test statistics are two costly steps requiring to solve optimization problems whose size grow with the number of features. Because of these two bottlenecks, the knockoff framework can hardly be employed in high dimension in practice despite the theoretical guarantees.

In this work, we focus on the high-dimensional knockoff setting and propose techniques to alleviate the bottlenecks mentioned above. Constructing quality knockoffs relies on solving a SDP problem which quickly becomes intractable as the dimension increases. We propose a coordinate ascent approach with low-rank approximations that both theoretically and experimentally scale well with the number of features.

This master thesis is organized as follows. Chapter 1 introduces the concept of feature selection and false discovery rate control, along with a few techniques that are usually employed. Chapter 2 details the knockoff selection procedure developed by Barber-Candès and principal theoretical guarantees. In Chapter 3, we focus on the knockoff features constitution part and propose a log-barrier coordinate ascent algorithm to solve the main bottleneck of the construction. We show that low-rank covariance approximations Chapter 4 introduces a sparse version of naive Bayes that scales linearly in the number of features against which it is trained. Finally, in Chapter 5 we detail experiments performed on genetic data and prove the scalability and efficiency of our methods.

Chapter 1

Feature selection

We introduce in this chapter the concepts of feature selection and false discovery rate control, as well as a short review of a few methods that are usually employed, including the Lasso and Benjamini–Hochberg-Yekutieli procedures.

1.1 Background on feature selection

1.1.1 Definitions

Feature selection primarily consists in identifying the most relevant features (or covariates) explaining an observed variable. It is closely related to Occam's razor which is a principle stating that the simplest explanation is often the best one. More formally, let $\mathcal{X}$ be a p-dimensional random vector representing observed features and $\mathcal{Y}$ a random target that may depend on $\mathcal{X}$. For example, $\mathcal{X}$ could be the expression levels of the genes of an individual, and $\mathcal{Y} \in \{0, 1\}$ a binary response indicating whether or not the person has some disease. We wish to find the subset of covariates $\mathcal{S} \subseteq \{1, \dots, p\}$ that best explains the target $\mathcal{Y}$, be it a numerical or a categorical variable. The notion of relevance of a feature $\mathcal{X}_j$ can be captured by the dependence of $\mathcal{Y}$ on $\mathcal{X}_j$. To do so, suppose that the joint vector $(\mathcal{X}, \mathcal{Y})$ follows some distribution, that is $(\mathcal{X}, \mathcal{Y}) \sim \mathcal{P}_{\mathcal{X},\mathcal{Y}}$. Even though finding the full conditional distribution $\mathcal{P}_{\mathcal{Y}|\mathcal{X}}$ is beyond hope in general, one may be interested in finding on which subset $\mathcal{S}$ of features $\mathcal{P}_{\mathcal{Y}|\mathcal{X}}$ depends. As this set is not necessarily unique, the principle of parsimony motivates us to find the smallest one. In other words, we are interested in finding the smallest subset $\mathcal{S}$ such that $\mathcal{Y} \mid \{\mathcal{X}_j\}_{j\in\mathcal{S}}$ is independent of $\{\mathcal{X}_j\}_{j\notin\mathcal{S}}$. Such a subset is called a Markov Blanket [**??**] and is not necessarily unique as shown in Figure 1.1. But this example is pathological because a feature is an exact linear combination of the two

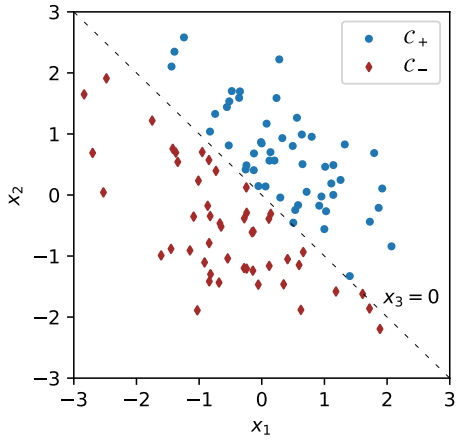

FIGURE 1.1: Illustration of the non-uniqueness of the Markov blanket [**?**]. Here $\mathcal{X}_1 \sim \mathcal{N}(0, 1)$ and $\mathcal{X}_2 \sim \mathcal{N}(0, 1)$ are independent, $\mathcal{X}_3 = \mathcal{X}_1 - \mathcal{X}_2$. $\mathcal{Y}$ is $\mathcal{C}_+$ if $\mathcal{X}_1 + \mathcal{X}_2 \geq 0$, $\mathcal{C}_-$ otherwise. The target $\mathcal{Y}$ can be predicted directly from any of the pairs $(\mathcal{X}_1, \mathcal{X}_2)$, $(\mathcal{X}_2, \mathcal{X}_3)$, $(\mathcal{X}_1, \mathcal{X}_3)$.

others. Under weak conditions, there exist a unique Markov Blanket containing the set of features of interest.

Remark 1. *In that case, the Markov Blanket coincides with the set of features which are said to be pairwise conditional independent. We define the set of null features* $\mathcal{H}_0 \subseteq \{1, \dots, p\}$ *as follows:* $j \in \mathcal{H}_0$ *if and only if* $\mathcal{Y} \perp\!\!\!\perp \mathcal{X}_j \mid \mathcal{X}_{-j}$ *(where* $\mathcal{X}_{-j}$ *denotes that all vector entries are kept except the* j*th). The relevant set of features is then* $\mathcal{S} = \{1, \dots, p\} \setminus \mathcal{H}_0$ *and the goal is to find a subset* $\hat{\mathcal{S}} \subseteq \{1, \dots, p\}$ *as close to* $\mathcal{S}$ *as possible.*

As a matter of example, consider the case of linear regression where we assume that the target and the features follow a linear relationship

$$\mathcal{Y} = \mathcal{X}^\top \boldsymbol{\beta} + \epsilon$$

where $\boldsymbol{\beta} \in \mathbb{R}^p$ and ϵ is a gaussian noise. Feature selection would consist in identifying which coefficients of $\boldsymbol{\beta}$ are non null.

In practice, one would observe several independent realizations of $(\mathcal{X}, \mathcal{Y})$ and would aggregate them into a feature matrix $X \in \mathbb{R}^{n \times p}$ of n samples and p features, and a target vector $\mathbf{y} \in \mathbb{R}^n$ respectively. The independence of the observations is a credible assumption in many real life settings.

1.1.2 Motivation

Feature selection may be used in a multitude of contexts. Sometimes, people are not even interested in the most relevant features but just want to reduce the size of their dataset. We point out here the main benefits of performing feature selection.

1. *Making a model more interpretable.* When a statistical model is built, for example linear regression, one is often interested in the weights attributed to each feature. It gives insights regarding the impact of a given feature on the target. In the era of deep learning [**?**], many machine learning models contain millions of parameters and are not interpretable at all anymore. Feature selection may be a way to reduce the number of parameters and concentrate on the few pertinent ones, thus reducing the model complexity.

2. *Facilitating data visualization.* Visualizing data may give a lot of intuition and understanding on a phenomenon. Our perceptions are known to be misleading in high dimension, and even in 3 dimensions [**?**]. For this reason, many algorithms, as t-SNE for example [**?**], try to project the high-dimensional data to a more accessible space. But that step is costly, and the fewer features, the more representative the output will be. Pre-selecting a small subset of impactful features can usually greatly improve the results of such projections.

3. *Reducing the training time.* Many machine learning algorithm have a super-linear time complexity in the number of features. Pre-selecting a small subset with a cheaper method can noticeably improve performances [**?**].

4. *Improving the generalization of machine learning models.* Test accuracy can be increased when a model is trained only on significant features [**?**]. Training a model against irrelevant features is prone to over-fitting noise.

5. *Avoiding the curse of dimensionality* [**?**]. For example, the k-nearest neighbors algorithm [**?**] is known to perform badly as the feature space dimension increases. The number of samples actually has to grow exponentially in the number of features in order for the algorithm to perform decently.

Recently, the cost of measuring more features has drastically decreased. Many datasets, and especially in biology, end up with several dozens of thousands of features. Most of these features are expected to be insignificant, but there is a priori no reason not to collect them. Expanding the feature matrix is cheap, and it is the role of machine learning algorithms to detect relevant columns subsequently.

1.1.3 Techniques

There exist a lot of different paradigms and techniques to perform feature selection [**??**]. Simplest approaches calculate a score for each feature and rank them accordingly; only those with the highest scores are kept. The problem of this is that features that may seem meaningless alone can turn out to be worthwhile when coupled with other features.

In general, different schemes have to be used depending on the type of data and the field of study [**???**]. We detail here only two methods to which we will come back later.

Lasso

First rediscovered by **?**, the Lasso has become increasingly popular because of its capacity to both shrink the coefficients of a linear regression towards 0, and to select a subset of the features. It is basically a linear least squares regression whose weights are penalized by their ℓ_1 norm, multiplied by some factor $\lambda > 0$.

$$\hat{\boldsymbol{\beta}}(\lambda) = \arg\min_{\mathbf{b}} \frac{1}{2} \|\mathbf{y} - X\mathbf{b}\|_2^2 + \lambda \|\mathbf{b}\|_1 \tag{1.1}$$

The ℓ_1 penalty tends to make the weights $\hat{\boldsymbol{\beta}}(\lambda)$ sparser as λ increases. Actually, for any feature j, there is a $\lambda_{\min}$ such that for all $\lambda \geq \lambda_{\min}$, $\hat{\boldsymbol{\beta}}_j(\lambda) = 0$. That wouldn't be the case with an ℓ_2 penalty, for which the coefficients usually tend to 0 without reaching that value. This property makes the Lasso particularly suited for feature selection; just keep the features whose weights are non-zero. However, the choice of λ seems to be arbitrary, especially as it's not possible to know ahead of time how many features will be selected.

The behavior of the mapping $\lambda \mapsto \hat{\boldsymbol{\beta}}(\lambda)$ has been extensively studied, and the algorithm LARS [**?**] was developed to efficiently compute it. It is possible to do so because this path is piecewise linear. It allows to compute $\hat{\boldsymbol{\beta}}$ for all relevant values of λ at a marginal cost. In practice, the value of λ giving the highest score on cross-validation is picked. The ℓ_1 regularization can easily be extended to many other estimators than the least squares, as for example logistic regression [**?**] or SVMs [**?**].

Sparse center classifiers

Nearest centroid classification [**?**] is a very simple classification scheme. When there are only two labels, it consists in computing the averages $\boldsymbol{\theta}^{\pm} = \sum_{j \in \mathcal{I}^{\pm}} \mathbf{x}_i \in \mathbb{R}^p$ of the data points from the positive and negative classes $\mathcal{C}^{\pm}$, where $\mathcal{I}^{\pm}$ contain the positive and the negative points. A new data point $\mathbf{x} \in \mathbb{R}^p$ is classified positive or negative depending on its closest average $\boldsymbol{\theta}^+$ or $\boldsymbol{\theta}^-$. Finding the class averages can be formulated as the following optimization problem.

$$\arg\min_{\boldsymbol{\theta}^+, \boldsymbol{\theta}^-} \frac{1}{n^+} \sum_{i \in \mathcal{I}^+} \left\|\mathbf{x}_i - \boldsymbol{\theta}^+\right\|_2^2 + \frac{1}{n^-} \sum_{i \in \mathcal{I}^-} \left\|\mathbf{x}_i - \boldsymbol{\theta}^-\right\|_2^2 \tag{1.2}$$

Note now Δ the decision boundary, such that $\mathbf{x}$ is classified $\mathcal{C}^+$ if $\Delta(\mathbf{x}) > 0$ and $\mathcal{C}^-$ otherwise.

$$\Delta(\mathbf{x}) = \left\|\mathbf{x} - \boldsymbol{\theta}^-\right\|_2^2 - \left\|\mathbf{x} - \boldsymbol{\theta}^+\right\|_2^2 \tag{1.3}$$

The expression 1.3 can be expanded into $\Delta(\mathbf{x}) = \|\boldsymbol{\theta}^+\|_2^2 + \|\boldsymbol{\theta}^-\|_2^2 + 2\mathbf{x}^\top(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-)$, making it clear that the decision depends on the feature j if and only if $\boldsymbol{\theta}_j^+ \neq \boldsymbol{\theta}_j^-$. Using this observation **?** introduced in 1.2 an additional ℓ_0-norm constraint $\|\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-\|_0 \leq k$, such that at most k entries of $\boldsymbol{\theta}^+$ and $\boldsymbol{\theta}^-$ are different. They show that this version can be solved very efficiently in closed-form. The authors obtain accuracy scores on par with the ones of the Lasso on the datasets they experiment. This a sparse centroid classification allows notably to perform feature selection by keeping the features j such that $\boldsymbol{\theta}_j^+ \neq \boldsymbol{\theta}_j^-$.

None of the selection scheme presented above offer strong guarantees regarding the number of false positives that are detected. The selected subset $\hat{\mathcal{S}}$ could potentially be disjoint from $\mathcal{S}$ if the selection criterion is not adapted to the problem.

1.2 False discovery rate control

When performing feature selection one is usually interested in two quantities, namely the *false discovery rate* (FDR) and the *power*. Intuitively, the former (resp. the latter) assesses the expected proportion of false discoveries (resp. true discoveries) of a selection procedure.

1.2.1 Definitions

Suppose the conditional probability distribution $\mathcal{Y} \mid \mathcal{X}$ depends only on the subset of features $\mathcal{S} \subset \{1, \ldots, p\}$. A feature selection algorithm outputs a subset $\hat{\mathcal{S}} \subset \{1, \ldots, p\}$ (which is potentially random)

that it judges to be relevant. The false discovery proportion (FDP), the false discovery rate (FDR), and the power are defined as follows

$$\mathrm{FDP} = \frac{\left|\hat{\mathcal{S}} \setminus \mathcal{S}\right|}{|\hat{\mathcal{S}}|}, \qquad \mathrm{FDR} = \mathbb{E}\left[\mathrm{FDP}\right], \qquad \mathrm{power} = \mathbb{E}\,\frac{\left|\hat{\mathcal{S}} \cap \mathcal{S}\right|}{|\mathcal{S}|} \tag{1.4}$$

The FDP merely measures the proportion of features in $\hat{\mathcal{S}}$ that are not in $\mathcal{S}$. It depends on the samples X and $\mathbf{y}$, and possibly on the inherent randomness of the selection algorithm. That is why the FDR assesses the expectation of this value. Finally, the power is the fraction of the important features that were actually discovered, in average.

Even though it is inconceivable to retrieve the whole set $\mathcal{S}$ with no error, a multitude of techniques attempt to find as many relevant features as possible (that is, maximizing the power) while maintaining the FDR under a certain threshold. The concept of FDR was first introduced by **?** and has become increasingly decisive in some sciences such has biology where medications may be developed from true positives [**?**].

1.2.2 Benjamini–Hochberg-Yekutieli procedures

The BJ [**?**] and BY [**?**] schemes are two methods controlling the FDR that are widely used in practice. They are closely related to each other but offer different guarantees regarding the effective control of the FDR. BH is less conservative than BY but makes stronger assumptions on the p-values it manipulates.

For each feature $j \in \{1, \dots, p\}$, let $\mathcal{H}_j$ be the null hypothesis (j does not belong to $\mathcal{S}$), and p_j the corresponding p-value. Let $q \in [0, 1]$ be some FDR target that should not be exceeded. We note $(p_{(j)})_j$ the sequence of p-values in increasing order; $p_{(1)} \leq \cdots \leq p_{(p)}$. Let m_0 be the number of true null hypotheses.

Benjamini–Hochberg

The Benjamini–Hochberg (BH) procedure was the first method proposed to control the FDR. It consists in the following steps:

1. Sorting the p-values such that $p_{(1)} \leq \ldots \leq p_{(p)}$
2. Finding the largest $k \in \mathbb{N}$ such that $p_{(k)} \leq \frac{k \cdot q}{p}$
3. Rejecting all the hypotheses $\mathcal{H}_{(j)}$ such that $j \in \{1, \ldots, k\}$

In the end, all the features j such that $p_j \leq p_{(k)}$ are selected. It guarantees that $\mathrm{FDR} \leq \frac{m_0}{p} q$ under the assumption that the p-values were computed independently, which can be very restrictive in many situations.

Benjamini–Yekutieli

Similarly, the Benjamini–Yekutieli procedure controls the FDR and does not need the p-values independence assumption, at the cost of a more conservative selection. It follows the same scheme, namely: ordering the p-values, finding the largest $k \in \mathbb{N}$ such that $p_{(k)} \leq \frac{k}{p \cdot c(p)} q$, and rejecting $\mathcal{H}_{(j)}$ if $j \leq k$. Note the additional factor $c(p) \geq 1$, defined in 1.5.

$$c(p) = \sum_{j=1}^{p} \frac{1}{j}, \qquad \mathrm{FDR} \leq \frac{m_0}{p} q \tag{1.5}$$

Both procedures are illustrated in Figure 1.2. They require p-values to be computed, which is not always feasible, especially when $p > n$. Furthermore, BH and BY have stronger guarantees than desired; for a desired FDR level q, they ensure that $\mathrm{FDR} \leq \frac{m_0}{p} q$, and the factor $\frac{m_0}{p}$ might be much smaller than 1. This is likely to affect the eventual statistical power of the procedure.

Many other FDR control techniques exist [**??**] but we won't focus on them.

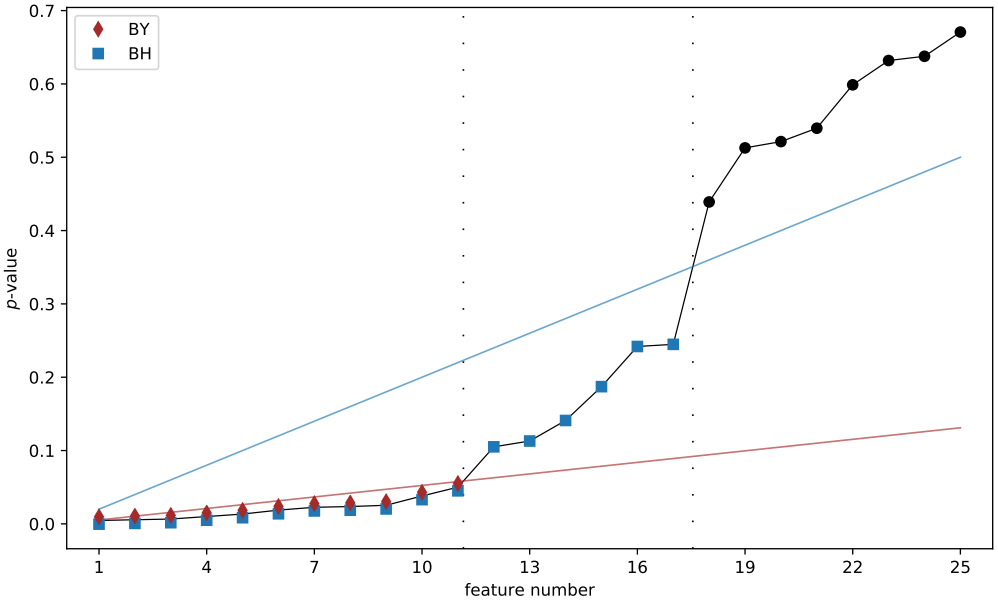

FIGURE 1.2: Illustration of the BH and the BY procedures. Once the p-values are ordered, they are geometrically equivalent to drawing a line of slope β going through the origin, identifying the last p-value under the line, and keeping features on its left. For BH, $\beta = \frac{q}{p}$ (in blue), while for BY, $\beta = \frac{q}{p \cdot c(p)}$ (in brown). In this toy example, $q = 0.5$, and it shows that BY is way more conservative than BH.

Chapter 2

The knockoffs filter framework

Recently **?** introduced the knockoffs framework to perform feature selection. The goal is to control the false discovery rate as defined in Section 1.2, that is, maintaining it under some threshold (while maximizing the power as much as possible). More formally, let $X \in \mathbb{R}^{n \times p}$ be a feature matrix and $\mathbf{y} \in \mathbb{R}^n$ the associated target vector. Suppose $\mathbf{y}$ depends only on the subset of features $\mathcal{S} \subseteq \{1, \dots, p\}$. Given some FDR target $q \in [0, 1]$, we wish to find a procedure such as, in average, the false discovery proportion is smaller that q, i.e. FDR $\leq q$. To do so, the key idea is to construct for each original feature X_j, $j \in \{1, \dots, p\}$, a knockoff (that is to say, fake) feature $\tilde{X}_j$ which is known to be out of the model. Original features are then selected only if they prove to be more significant than their knockoff counterparts.

To compare a feature and its knockoff, several quantities will be computed by interchanging matrix columns. For this reason, we define the swap operator.

Definition 1. *(*swap *operator) Given two matrices $A, B \in \mathbb{R}^{n \times p}$ with the same size, we define the* swap *operator on the concatenated matrix $[A, B]$ as follows. For a any subset of indices $S \subseteq \{1, \dots, p\}$, $[A, B]_{\text{swap}(S)}$ is the transformed matrix where the columns A_j and B_j were swapped for all $j \in S$.*

In the following sections, we are going to detail principal aspects of the construction of the knockoff features, the computation of statistics for both original and knockoff features, and the feature selection itself, based on those statistics.

2.1 Knockoffs construction

In this section, we focus on the construction of the knockoff matrix. At first **?** introduced what they called the *fixed*-X knockoffs variable selection procedure. It relies on the creation of fake features satisfying some correlation constraints with the original features. Unfortunately, it can only perform adequately when $n \geq 2p$, even though it can be partially adapted to the case where $n \geq p$. Later **?** proposed an extension of the framework called *model*-X knockoffs, in which knockoff features are sampled from a learned distribution. Despite the restriction of *fixed*-X knockoffs, this method is still appealing as the construction of the knockoff variables is straightforward (and moderately costly). On the other hand, *model*-X knockoffs work in higher dimensions but need an estimation of the distribution that generated the data, which is a hard problem in general. We will focus on the construction of *model*-X knockoffs as we are principally interested in the high dimensional setting.

In this section, let $(\mathcal{X}, \mathcal{Y})$ be a pair of random variables, $\mathcal{X}$ being composed of p features, and $\mathcal{Y}$ the associated target in $\mathbb{R}$. We suppose that the feature matrix and the target vector $(X, \mathbf{y})$ are composed of independent samples from $(\mathcal{X}, \mathcal{Y})$.

2.1.1 General case

We wish to build knockoff features $\tilde{X} \in \mathbb{R}^{n \times p}$, and to do so we are going to sample from a random variable $\tilde{\mathcal{X}}$ built such that $[\mathcal{X}; \tilde{\mathcal{X}}]$ follows the properties S.1 and S.2 defined thereafter.

Definition 2. *(model-X knockoffs) Given random vector $\mathcal{X} \in \mathbb{R}^p$ of features, a random vector $\tilde{\mathcal{X}} \in \mathbb{R}^p$ is said to be model-$\mathcal{X}$ knockoffs with respect to $\mathcal{Y}$ if it satisfies the two following properties:*

S.1 *For any $S \subseteq \{1, \dots, p\}$, $[\mathcal{X}; \tilde{\mathcal{X}}]^\top_{\text{swap}(S)} \overset{d}{=} [\mathcal{X}; \tilde{\mathcal{X}}]^\top$*

S.2 $\tilde{\mathcal{X}} \perp\!\!\!\perp \mathcal{Y} \mid \mathcal{X}$

Intuitively, the condition S.1 ensures that a knockoff feature is sufficiently close to its associated original feature so that swapping them doesn't change the distribution of the concatenated random vector. The independence condition S.2 states that the knockoff features carry no additional information on $\mathcal{Y}$, given $\mathcal{X}$. It is trivially satisfied if $\tilde{X}$ is built without exploiting $\mathbf{y}$. However, constructing knockoffs meeting the first distribution equality is practically infeasible in general.

Remark 2. *By comparison, constructing fixed-X knockoffs $\tilde{X} \in \mathbb{R}^{n \times p}$ is a deterministic process. Knockoff features must satisfy the two following covariance equalities.*

$$\begin{cases} \tilde{X}^\top \tilde{X} = \Sigma \\ X^\top \tilde{X} = \Sigma - \operatorname{diag} \mathbf{s} \quad \textit{where } \mathbf{s} \textit{ is a non negative vector.} \end{cases}$$

It ensures that $\tilde{X}$ has the same covariance as the original matrix X and that the correlation between distinct original and knockoff variables is the same as the correlation between the two originals.

A solution to these two equations is guaranteed to exist only if $2p \leq n$. Moreover, fewer test statistics yield theoretical guarantees regarding the FDR control of the procedure, and model-X knockoffs tend to give a higher statistical power experimentally. In the coming sections, we will only consider model-X knockoffs as we are interested in the high dimensional setting.

2.1.2 Gaussian case

In the particular case where $\mathcal{X}$ is multivariate gaussian, the exact distribution of $\tilde{\mathcal{X}}$ can be derived.

Proposition 1. *Suppose that $\mathcal{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$. If $\tilde{\mathcal{X}}$ is a random variable such that*

$$\begin{bmatrix} \mathcal{X} \\ \tilde{\mathcal{X}} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu} \end{bmatrix}, \Omega \right), \quad \textit{where} \quad \Omega = \begin{bmatrix} \Sigma & \Sigma - \operatorname{diag} \mathbf{s} \\ \Sigma - \operatorname{diag} \mathbf{s} & \Sigma \end{bmatrix} \quad \textit{for some } \mathbf{s} \in \mathbb{R}^p,$$

then $[\mathcal{X}; \tilde{\mathcal{X}}]$ satisfies the swap *property S.1, provided that Ω is positive semidefinite (so that it is indeed a covariance matrix).*

In Proposition 1, $\mathbf{s} \in \mathbb{R}^p$ can be any vector such that $\Omega \succeq \mathbf{0}$. We will come back later to the choice of $\mathbf{s}$ which is actually crucial. For now, assume that $\mathbf{s}$ satisfies this assumption. This result gives a way of constructing the knockoff features from X. Indeed, as $[\mathcal{X}; \tilde{\mathcal{X}}]$ is multivariate normal, we may compute the exact distribution of $\tilde{\mathcal{X}} \mid \mathcal{X}$ with classical conditional formulas [?] as shown in 2.1.

$$\tilde{\mathcal{X}} \mid \mathcal{X} \sim \mathcal{N}(\boldsymbol{v}, \Upsilon), \qquad \text{where} \quad \begin{cases} \boldsymbol{v} = \mathcal{X} - \mathcal{X} \Sigma^{-1} \operatorname{diag}\{\mathbf{s}\} \\ \Upsilon = \operatorname{diag}\{\mathbf{s}\} \left(2 I_{p \times p} - \Sigma^{-1} \operatorname{diag}\{\mathbf{s}\}\right) \end{cases} \tag{2.1}$$

To put this into practice, one would compute the empirical mean $\hat{\boldsymbol{\mu}} \in \mathbb{R}^p$ and covariance $\hat{\Sigma} \in \mathbb{R}^{p \times p}$ of $\mathcal{X}$ using the observed feature matrix X. Then, each row of $\tilde{X}$ is sampled according to a gaussian distribution $\mathcal{N}(\boldsymbol{v}, \Upsilon)$ whose parameters are described in 2.1. Note in particular that the construction process is random; if it is repeated, it may very well return different knockoffs, and thus different selected features in the end. Because of this instability, several people attempted to aggregate knockoffs and reduce the variance in the selection [???]. Depending on the available computing power, various algorithms may be used to estimate the covariance matrix Σ. The empirical estimator is cheap but known to be imprecise when $p > n$. Shrunk estimators like the one proposed by ? provide better results in high dimension.

The gaussian hypothesis is obviously rarely verified in practice but yields acceptable results even when $\mathcal{X}$ is far from gaussian. It partly comes from the fact that, rather than constructing $\tilde{\mathcal{X}}$ to respect S.1, a weaker condition would be to enforce $[\mathcal{X}; \tilde{\mathcal{X}}]$ and $[\mathcal{X}; \tilde{\mathcal{X}}]_{\text{swap}(S)}$ to have the same first two moments (mean and covariance). It turns out to be the case if $\tilde{\mathcal{X}} \mid \mathcal{X}$ is constructed as described in 2.1. In the remaining of this master thesis, we will restrain ourselves to the gaussian hypothesis. Note that generating knockoffs requires an estimation of the distribution of $\mathcal{X}$ only, and not $\mathcal{Y} \mid \mathcal{X}$ as most methods would. It is particularly appealing because labeling data is often the most costly part, while acquiring samples $\mathbf{x} \sim \mathcal{X}$ is easier.

2.2 Statistics computation

We suppose now that the distribution of $\mathcal{X}$ was determined and that we are able to sample from it. It gives a knockoff matrix $\tilde{X} \in \mathbb{R}^{n \times p}$ of the same size as X.

2.2.1 General principle

Given the original feature matrix and the sampled knockoffs $X, \tilde{X} \in \mathbb{R}^{n \times p}$, statistics w_j are computed for all $j \in \{1, \dots, p\}$. Each w_j represents how more significant the original feature X_j is compared to $\tilde{X}_j$. These statistics must satisfy the *flip-sign* technical condition 3 for the FDR control to carry out, but a wide variety of choices is possible as will be shown.

Definition 3. *(flip-sign property) A statistics function $\omega\colon \mathbb{R}^{n \times 2p} \times \mathbb{R}^n \to \mathbb{R}^p$ is said to follow the flip-sign property if for any $S \subseteq \{1, \dots, p\}$ and any $j \in \{1, \dots, p\}$,*

$$\omega_j\big([X, \tilde{X}]_{\mathrm{swap}(S)}, \mathbf{y}\big) = \begin{cases} -\omega_j\big([X, \tilde{X}], \mathbf{y}\big) & \text{if } j \in S. \\ \omega_j\big([X, \tilde{X}], \mathbf{y}\big) & \text{otherwise.} \end{cases}$$

This property is very natural, as it is simply asking the original and the knockoff features to play antisymmetric roles.

2.2.2 Statistics aggregation

Constructing statistics satisfying the *flip-sign* property 3 is actually straightforward as an elementary scheme in two steps leads to such statistics. The idea is to build statistics for each original and each knockoff feature, and then aggregate them.

1. First construct statistics $[\mathbf{z}; \tilde{\mathbf{z}}] = \zeta\big([X, \tilde{X}], \mathbf{y}\big)$ with some function $\zeta\colon \mathbb{R}^{n \times 2p} \times \mathbb{R}^n \to \mathbb{R}^{2p}$ satisfying

$$[\mathbf{z}; \tilde{\mathbf{z}}]_{\mathrm{swap}(S)} = \zeta\big([X, \tilde{X}]_{\mathrm{swap}(S)}, \mathbf{y}\big), \quad \forall S \subseteq \{1, \dots, p\} \tag{2.2}$$

   The statistics z_j (resp. $\tilde{z}_j$) indicates how significant the original (resp. knockoff) feature j is.

2. Then aggregate for each $j \in \{1, \dots, p\}$ the statistics of the original feature z_j and the one of the corresponding knockoff $\tilde{z}_j$ with an antisymmetric function $a_j\colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. That is, set $w_j = a_j(z_j, \tilde{z}_j)$.

It is easy to show that such constructed statistics will satisfy the *flip-sign* property 3.

Basically any antisymmetric function a_j could work, but some choices lead to a better power experimentally. Here are a few examples of mappings:

- $w_j = z_j - \tilde{z}_j$ (experimentally gives highest power in many cases)
- $w_j = \max(z_j, \tilde{z}_j) \cdot \mathrm{sign}(z_j - \tilde{z}_j)$ (first proposed by **?**)
- $w_j = \log \frac{z_j}{\tilde{z}_j}$

As for the function ζ, it only needs to satisfy the *swap* property 2.2. This condition may seem restrictive but a large number of choices are actually valid.

2.2.3 Examples

We illustrate here a few examples of valid mappings ζ to build aggregated knockoff statistics. After empirical observations **?** suggest to use the Lasso Signed Max (LSM) statistics defined as follows.

$$z_j = \sup\{\lambda \mid \hat{\boldsymbol{\beta}}_j(\lambda) \neq 0\}, \qquad \hat{\boldsymbol{\beta}}(\lambda) = \arg\min_{\mathbf{b}} \frac{1}{2}\big\|\mathbf{y} - [X, \tilde{X}]\mathbf{b}\big\|_2^2 + \lambda \|\mathbf{b}\|_1 \tag{2.3}$$

The vector $\hat{\boldsymbol{\beta}}(\lambda) \in \mathbb{R}^{2p}$ contains the coefficients of a Lasso model with penalty coefficient $\lambda > 0$. As mentioned in Subsection 1.1.3, all the coefficients are null starting from $\lambda = +\infty$, and are likely to shift (and to grow in absolute value) as $\lambda \to 0$. It makes sense to use the first point where the coefficient

β_j becomes non-null as a significance metric for each individual feature j. In addition, the algorithm LARS is able to compute that value pretty efficiently [?].

Another possibility proposed by ? is to train a plain Lasso estimator and keep its coefficient in absolute value. Here again, the LARS algorithm allows to cross-validate the penalty λ efficiently. Experimentally, these statistics often give remarkable results in terms of power.

More generally, the coefficients in absolute value of any reasonable regressor (or classifier, depending on the task) constitute a judicious option of statistics. The fact that the *flip-sign* property isn't restrictive makes the knockoff framework very robust. An adapted model can be trained depending on the data, be it logistic regression, SVMs, or random forests ?. When the model has hyper-parameters, those are typically tuned using cross-validation.

2.3 Selection thresholds

The selection itself requires the constitution of a data-dependent threshold τ conditioned by the target FDR $q \in [0, 1]$. In the end, only the features j whose statistics w_j are above the threshold will be selected.

$$\hat{\mathcal{S}} = \{j \mid w_j \geq \tau\} \tag{2.4}$$

The threshold τ can be adapted depending on how restrictive the procedure ought to be. It leads to different guarantees regarding the control of the FDR. Two selection procedures are established by ?, namely *knockoff* and *knockoff+*. They control the modified FDR (as defined below) and the FDR respectively.

Definition 4. *Given an estimate $\hat{\mathcal{S}}$ of $\mathcal{S}$ and a desired false discovery rate target $q \in [0, 1]$, the modified FDR is defined as follows:*

$$mFDR = \mathbb{E}\,\frac{|\hat{\mathcal{S}} \setminus \mathcal{S}|}{|\hat{\mathcal{S}}| + 1/q}$$

Since $0 \leq q \leq 1$, mFDR as described in 4 is always smaller than the actual FDR. It means that controlling the mFDR is less restrictive than controlling the FDR, and the FDR could potentially be much larger than the mFDR. But if the target threshold q is not too small, and if many features are selected by the procedure, the modified version of the FDR is close enough to the actual FDR. Being less restrictive by controlling only the mFDR can greatly improve the power of the selection.

The only difference between the *knockoff* and *knockoff+* procedures is the selection threshold τ.

Definition 5. *(knockoff and knockoff+ thresholds) Given the statistics $\mathbf{w} \in \mathbb{R}^p$ computed from X and $\tilde{X}$, we define the knockoff and knockoff+ thresholds respectively as follows:*

$$\tau = \min\left\{t > 0 \mid \frac{\#\{j \mid w_j \leq -t\}}{\#\{j \mid w_j \geq t\}} \leq q\right\} \tag{2.5}$$

$$\tau^+ = \min\left\{t > 0 \mid \frac{1 + \#\{j \mid w_j \leq -t\}}{\#\{j \mid w_j \geq t\}} \leq q\right\} \tag{2.6}$$

These thresholds only differ by their numerator, where τ^+ has an additional $+1$. The main result of ?? regarding the control of the mFDR and FDR is stated in Theorem 1.

Theorem 1. *(guarantees of the knockoff procedures) Construct $\hat{\mathcal{S}} = \{j \mid w_j \geq \tau\}$ and $\hat{\mathcal{S}}^+ = \{j \mid w_j \geq \tau^+\}$. These selections ensure the following FDR controls:*

$$mFDR\big[\hat{\mathcal{S}}\big] \leq q, \qquad FDR\big[\hat{\mathcal{S}}^+\big] \leq q \tag{2.7}$$

Even if only the *knockoff+* method truly control the FDR, using the threshold τ improves the power and gives reasonable FDR, in the same way the BH procedure usually controls the FDR even when the tests are not independent.

2.4 Bottlenecks

Despite the nice theoretical guarantees on the FDR control that the knockoff procedure proposes, two bottlenecks hurt its performances in the high-dimensional context.

2.4.1 Knockoffs construction

In the gaussian setting, knockoff features are sampled according to the distribution 2.1. The control guarantees hold for any $\mathbf{s} \in \mathbb{R}^p$ such that the covariance matrix Ω is semidefinite positive.

Proposition 2. *Let* $\Omega = \begin{bmatrix} \Sigma & \Sigma - \operatorname{diag} \mathbf{s} \\ \Sigma - \operatorname{diag} \mathbf{s} & \Sigma \end{bmatrix}$. *Then* $\Omega \succeq \mathbf{0}_{p \times p}$ *if and only if* $2\Sigma \succeq \operatorname{diag} \mathbf{s} \succeq \mathbf{0}$.

Proof. Note $D = \operatorname{diag} \mathbf{s}$. The Schur complement $\Omega_{/\Sigma}$ (see Appendix B.1) is given by $\Omega_{/\Sigma} = 2D - D\Sigma^{-1}D$. From this, we get that $\Omega \succeq \mathbf{0}$ is equivalent to $\Omega_{/\Sigma} \succeq \mathbf{0}$. Define M and its Schur complements as follows

$$M = \begin{bmatrix} 2D & D \\ D & \Sigma \end{bmatrix}, \qquad \begin{cases} M_{/2D} = \Sigma - \frac{1}{2}D \\ M_{/\Sigma} = 2D - D\Sigma^{-1}D \end{cases}$$

Finally, $\Omega_{/\Sigma} = M_{/\Sigma}$ is p.s.d. if and only if both $\Sigma - \frac{1}{2}D$ and D are p.s.d. □

Even if we could use *any* $\mathbf{s}$ satisfying this constraint, but some solutions lead to a better statistical power. As will be shown in Chapter 3, finding good solutions amounts to solving a SDP which becomes intractable when p is large.

2.4.2 Statistics computation

The computation of the statistics presented in Section 2.2 on the concatenated feature matrix $[X, \tilde{X}]$ may also become an obstacle when n and p grow. This comes from the fact that most estimators have hyper-parameters that need to be cross-validated. The cross-validation step may multiply the training time by several order of magnitudes. On top of that, most estimators have a training time complexity that is more than linear in the number of features.

These two bottlenecks make the use of the knockoff procedure virtually impossible when p is more than a few thousands. The following chapters tackle these two issues by proposing a coordinate ascent algorithm, low-rank approximations and statistics that are fast to compute.

2.5 Python implementation

Implementations of the knockoff filter framework were provided by **??** (along with other coauthors) in the languages R and MATLAB[1]. To the best of our knowledge, no public and unified Python implementation is presently available. As Python is a very popular language in the machine learning community, and as it keeps growing year after year, we decided to make a Python implementation[2] that contains fundamental components and that may be easily extended depending on the needs of potential users.

The implementation is deliberately compatible with the Scikit-Learn ecosystem [**?**], so that estimators may be combined and used in a pipeline object. It provides 3 main components, namely a knockoffs generator `KnockoffsGenerator`, statistics computations utility functions, and a selector `BaseKnockoffsSelector`. The constituent `KnockoffsGenerator` subclasses `BaseEstimator` and `TransformerMixin` from Scikit-Learn, and `BaseKnockoffsSelector` subclasses and `BaseEstimator` and `SelectorMixin`. They implement `fit` and `transform` methods which are intuitive and follow Scikit-Learn semantics.

Components whose speed is crucial are written in Cython [**?**] and take advantage of the highly optimized libraries BLAS and LAPACK, that are available in Cython through a SciPy wrapper. It is for instance the case of the coordinate ascent algorithm detailed in Chapter 3, allowing to construct knockoffs efficiently.

In the following chapters, we present the ideas and algorithms that we implemented to make this Python toolkit scalable to large dimensions.

[1] The R package page can be found at this address. Sources for both R and MATLAB are stored in this GitHub repository.

[2] The implementation can be found at this address

Code 1 Example of knockoffs usage

```
selector = SimpleKnockoffsSelector(
    GaussianXKnockoffs(),
    z_to_w_statistics(estimator_statistics(estimator=LassoCV())),
    alpha=0.2,
)

selector.fit(X, y)
print(f'Selected features: {selector.mask_}')
```

Chapter 3

Efficient knockoffs construction

As shown in Chapter 2, in the case of gaussian knockoffs, $\tilde{X}$ can be sampled from a normal distribution $\mathcal{N}(\boldsymbol{\upsilon}, \Upsilon)$ whose parameters $\boldsymbol{\upsilon}$, Υ are formulated in 3.1.

$$\tilde{\mathcal{X}} \mid \mathcal{X} \sim \mathcal{N}(\boldsymbol{\upsilon}, \Upsilon), \qquad \text{where} \quad \begin{cases} \boldsymbol{\upsilon} = \mathcal{X} - \mathcal{X}\Sigma^{-1}\operatorname{diag}\{\mathbf{s}\} \\ \Upsilon = \operatorname{diag}\{\mathbf{s}\}\left(2I_{p\times p} - \Sigma^{-1}\operatorname{diag}\{\mathbf{s}\}\right) \end{cases} \tag{3.1}$$

More generally, the same parameters are derived by only imposing $\left[X;\tilde{X}\right]_{\text{swap}(S)}^{\top}$ and $\left[X;\tilde{X}\right]^{\top}$ to have equal first two moments for any subset $S \subseteq \{1,\dots,p\}$, rather than the same distribution. These formulas are valid for any vector $\mathbf{s} \in \mathbb{R}^p$ such that Ω is indeed a covariance matrix (semidefinite positive).

$$\Omega = \begin{bmatrix} \Sigma & \Sigma - \operatorname{diag}\mathbf{s} \\ \Sigma - \operatorname{diag}\mathbf{s} & \Sigma \end{bmatrix} \succeq \mathbf{0}_{2p\times 2p}$$

As shown in Proposition 2, this matrix is semidefinite positive if and only if $\mathbf{0}_{p\times p} \preceq \operatorname{diag}\mathbf{s} \preceq 2\Sigma$. The first inequality clearly holds if all the entries of $\mathbf{s}$ are positive. It is however trickier to identify all vectors $\mathbf{s}$ satisfying the second one.

In this chapter, we show why the choice of $\mathbf{s}$ is important and how to efficiently find an appropriate one in the high-dimensional setting. We also form low-rank covariance approximations and procedures to quickly sample $\tilde{\mathcal{X}}$. In the remaining, we note Σ the true covariance and $\hat{\Sigma}$ its estimation from the samples X (be it empirical or derived from more sophisticated methods).

3.1 Equi-correlated knockoffs

A cheap solution

For any psd matrix $A \in \mathbb{R}^p$, $A+\alpha I_{p\times p}$ has the same eigenvalues as A, but shifted by α. It gives a fast and simple way to find a feasible $\mathbf{s}$: setting all the entries to the smallest eigenvalue of $2\hat{\Sigma}$ (*equi*-correlated knockoffs).

$$s_j = 2\lambda_{\min}(\hat{\Sigma}) \quad \text{for all} \quad 0 \le j \le p \tag{3.2}$$

The problem of finding the smallest eigenvalue $\lambda_{\min}(\hat{\Sigma})$ can be solved efficiently, using for instance a singular value decomposition which runs in $\mathcal{O}(p^3)$ steps [**?**].

Why it is not desirable

For large values of p, the minimum eigenvalue of $\hat{\Sigma}$ is likely to be very small, unless $\hat{\Sigma}$ has a substantially special structure. Let us analyze briefly the covariance of $\left[\mathcal{X};\tilde{\mathcal{X}}\right]$ to understand why it is unprofitable for the selection procedure. If $\tilde{\mathcal{X}}$ is built according to 3.1, it satisfies

$$\begin{cases} \operatorname{cov}(\tilde{\mathcal{X}}_j, \tilde{\mathcal{X}}_{j'}) = \operatorname{cov}(\mathcal{X}_j, \mathcal{X}_{j'}) \text{ for all } j, j' \\ \operatorname{cov}(\mathcal{X}_j, \tilde{\mathcal{X}}_{j'}) = \operatorname{cov}(\mathcal{X}_j, \mathcal{X}_{j'}) \text{ for all } j \neq j' \\ \operatorname{cov}(\mathcal{X}_j, \tilde{\mathcal{X}}_j) = \operatorname{cov}(\mathcal{X}_j, \mathcal{X}_j) - s_j \text{ for all } j \end{cases}$$

First, $\tilde{\mathcal{X}}$ has the same internal covariance has $\mathcal{X}$, and two distinct original and knockoff features have the same covariance as the one of the two corresponding original features. It makes the knockoff features sufficiently close to the original features to fool the estimator computing the statistics. However, an

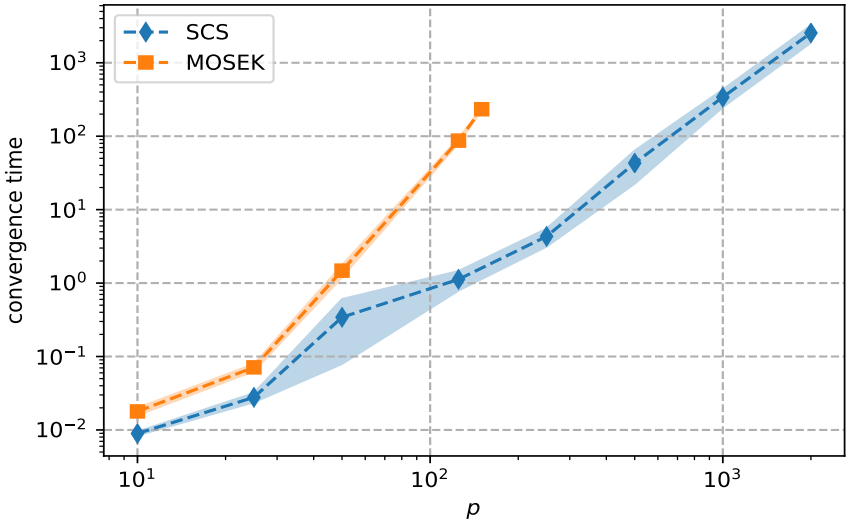

FIGURE 3.1: Time (in seconds) required to converge for SCS (blue) and MOSEK (orange) solvers as a function of p in log-log scale. Rapidly, MOSEK needs too much memory and can't be run for $p > 150$. Both match the theoretical $\mathcal{O}(p^3)$ rate and SCS already needs 15 minutes for $p = 1000$. See Appendix A.1 for details regarding the random generation of covariance matrices $\hat{\Sigma}$.

original feature j and its corresponding knockoff have a covariance that is smaller the larger s_j is. If s_j is close to 0, $\mathcal{X}_j$ cannot be distinguished from $\tilde{\mathcal{X}}_j$ and the selection is prone to suffer from a low power. This fact is even more detrimental when p is large as $\lambda_{\min}$ is likely to be very close to 0.

3.2 SDP knockoffs

The observation of the previous Section 3.1 motivates us to maximize the entries of $\mathbf{s}$, while maintaining the inequality $\operatorname{diag} \mathbf{s} \preceq 2\Sigma$. This can be formulated in the optimization problem 3.3.

$$\underset{\mathbf{s}\in\mathbb{R}^p}{\arg\max} \sum_{j=1}^{p} s_j \qquad \text{subject to} \quad \begin{cases} s_j \geq 0 \text{ for all } j \\ \operatorname{diag} \mathbf{s} \preceq 2\hat{\Sigma} \end{cases} \tag{3.3}$$

This problem is a structured semidefinite program (SDP) and can be solved efficiently for small values of p by interior point methods [??] for example. However, it quickly becomes intractable for large values of p (one thousand or more). In memory, these schemes need to store roughly $\mathcal{O}(m^2)$ floats and the time complexity is approximately $\mathcal{O}\left(m^3\right)$, where m is the number of constraints (p is this case). Experimentally, alternating direction [?] and first order methods like SCS [?] also fail to scale satisfactorily in high dimension. Figure 3.1 shows the convergence time of the solvers SCS and MOSEK as a function of p. It makes it clear that different approaches have to be considered when p is more than a few thousands. Moreover, most solutions returned by these algorithms are actually infeasible because of numerical approximations.

In order to reduce the computation time ? suggest to solve an approximated problem of 3.3 in 2 steps that we describe below.

Step 1. Pick a block-diagonal approximation $\hat{\Sigma}_{\text{approx}}$ of $\hat{\Sigma}$ and solve

$$\underset{\hat{\mathbf{s}}\in\mathbb{R}^p}{\arg\max} \sum_{j=1}^{p} \hat{s}_j \qquad \text{subject to} \quad \begin{cases} \hat{s}_j \geq 0 \text{ for all } j \\ \operatorname{diag} \hat{\mathbf{s}} \preceq 2\hat{\Sigma}_{\text{approx}} \end{cases} \tag{3.4}$$

Step 2. Solve the one dimensional maximization

$$\arg\max_{\gamma \in \mathbb{R}} \gamma \qquad \text{subject to} \quad \operatorname{diag}(\gamma \cdot \hat{\mathbf{s}}) \preceq 2\hat{\Sigma} \tag{3.5}$$

Finally, pick $\mathbf{s} = \gamma \cdot \hat{\mathbf{s}}$.

Remark 3. *Note that the two extreme options $\hat{\Sigma}_{approx} = I$ and $\hat{\Sigma}_{approx} = \hat{\Sigma}$ yield the same solutions as solving equi-correlated knockoffs 3.2 and full SDP knockoffs 3.3 respectively.*

Solving 3.5 in Step 2 can be done very efficiently using bisection as it is a one-dimensional SDP. The optimization problem 3.4 is the same as the one in 3.3, except for the approximation $\hat{\Sigma}_{\text{approx}}$. To speed up the computations, $\hat{\Sigma}_{\text{approx}}$ can be chosen to be a block-diagonal matrix of k blocks. The maximization then reduces to k smaller SDPs for which solutions can be found more efficiently. All the sub-problems being independent, they could even be distributed in several computation nodes. Picking $\hat{\Sigma}_{\text{approx}}$ is a compromise between available computation time and quality of the approximation (hence ultimate statistical power of the procedure). The block-diagonal structure can be obtained by reordering columns of $\hat{\Sigma}$ with a clustering step grouping similar features (i.e. highly correlated features).

In practice, the block-diagonal structure is unrealistic. On top of that, the problem remains intractable when p is a few dozens of thousands, even if ten groups of features are identified. The clustering step may also be costly, and the covariance matrix $\hat{\Sigma}$ itself might not even be storable in memory.

3.3 A coordinate ascent approach

We propose here a coordinate ascent algorithm which is proven to converge when coupled with a log-barrier penalty. It may very well be executed in the block-diagonal arrangement depicted in the previous section.

3.3.1 Notations and preliminaries

Notations. Let $M \in \mathbb{R}^{p \times p}$. Given two sets of indices $\mathcal{I}, \mathcal{J} \in \{1, \dots, p\}$, we note $M_{\mathcal{I},\mathcal{J}}$ the $|\mathcal{I}| \times |\mathcal{J}|$ matrix obtained by keeping the $|\mathcal{I}|$ rows and $|\mathcal{J}|$ columns indexed by $\mathcal{I}$ and $\mathcal{J}$ respectively. By convenience, an integer j denotes the set $\{j\}$ and j^c $\{1, \dots, p\} \setminus \{j\}$ in the matrix subscripting context. For example, if $M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, then $M_{1^c,1^c} = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$ and $M_{1^c,1} = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$.

Positiveness characterisation. Suppose M is structured as

$$M = \begin{bmatrix} \xi & \mathbf{y}^\top \\ \mathbf{y} & B \end{bmatrix}$$

where B is symmetric and invertible.

Lemma 1. *Using Schur complements (see Appendix B.1), it holds that M is psd if and only if*

$$B \succeq \mathbf{0} \quad \text{and} \quad \xi - \mathbf{y}^\top B^{-1} \mathbf{y} \geq 0$$

In subscript notations, this is equivalent to $M_{1^c,1^c} \succeq \mathbf{0}$ and $M_{1,1} - M_{1^c,1}^\top M_{1^c,1^c} M_{1^c,1} \geq 0$. This statement can be generalized as follows. For any $j \in \{1, \dots, p\}$, M is psd if and only if both conditions in 3.6 are satisfied

$$\begin{cases} M_{j^c,j^c} \succeq \mathbf{0} \\ M_{j,j} - M_{j^c,j}^\top M_{j^c,j^c} M_{j^c,j} \geq 0 \end{cases} \tag{3.6}$$

Proof. Let $j \in \{1, \dots, p\}$. Let P be the permutation matrix swapping columns $1 \leftrightarrow j$ and letting other columns unchanged. In two-line form, it is written

$$P = \begin{pmatrix} 1 & 2 & \dots & j-1 & j & j+1 & \dots & p \\ j & 2 & \dots & j-1 & 1 & j+1 & \dots & p \end{pmatrix}$$

M is psd if and only if $P^\top MP$ is psd. $P^\top MP$ is M with lines and columns 1 and j swapped. By applying Lemma 1 on it, we get the result. □

3.3.2 Coordinate ascent

Using the characterization 3.6, it appears that the feasibility of $\mathbf{s}$ in SDP 3.3 is equivalent to the three following conditions, for any $j \in \{1, \dots, p\}$

$$2\hat{\Sigma} \succeq \operatorname{diag} \mathbf{s} \succeq \mathbf{0} \iff \begin{cases} \mathbf{s} \geq \mathbf{0} \\ 2\hat{\Sigma}_{j,j} - s_j - 4\hat{\Sigma}_{j^c,j}^\top \left(2\hat{\Sigma}_{j^c,j^c} - \operatorname{diag} \mathbf{s}_{j^c}\right)^{-1} \hat{\Sigma}_{j^c,j} \geq 0 \\ 2\hat{\Sigma}_{j^c,j^c} - \operatorname{diag} \mathbf{s}_{j^c} \succeq \mathbf{0} \end{cases} \tag{3.7}$$

This observation motivates the following coordinate approach, as described by **?**. Start with a feasible solution $\mathbf{s}_0$, for example $\mathbf{s}_0 = \mathbf{0}_p$. At each iteration, only one coordinate j of $\mathbf{s}$ will be updated (to optimize a sub-objective), and the constraints remain satisfied if we pick s_j such that

$$\begin{cases} s_j \geq 0 \\ s_j \leq 2\hat{\Sigma}_{j,j} - 4\hat{\Sigma}_{j^c,j}^\top \left(2\hat{\Sigma}_{j^c,j^c} - \operatorname{diag} \mathbf{s}_{j^c}\right)^{-1} \hat{\Sigma}_{j^c,j} \end{cases} \tag{3.8}$$

Consequently, we set s_j to the maximum value keeping the constraints valid, that is

$$s_j = \max\left(2\hat{\Sigma}_{j,j} - 4\hat{\Sigma}_{j^c,j}^\top \left(2\hat{\Sigma}_{j^c,j^c} - \operatorname{diag} \mathbf{s}_{j^c}\right)^{-1} \hat{\Sigma}_{j^c,j},\, 0\right)$$

Unfortunately, this method is not guaranteed to converge to the global maximum, even when the problem is concave. A slight modification depicted in the next section will correct that.

3.3.3 Log-barrier

Instead of optimizing 3.3, we absorb the constraint $2\hat{\Sigma} \succeq \mathbf{s}$ into the objective by penalizing solutions too close to the feasibility frontier, as described by **?**, §11.3. It depends on the additional term $\lambda \cdot \log\det\left(2\hat{\Sigma} - \operatorname{diag} \mathbf{s}\right)$ for some coefficient $\lambda > 0$:

$$\underset{\mathbf{s} \in \mathbb{R}^p}{\arg\max} \sum_{j=1}^{p} s_j + \lambda \cdot \log\det\left(2\hat{\Sigma} - \operatorname{diag} \mathbf{s}\right) \qquad \text{subject to} \quad s_j \geq 0 \text{ for all } j \tag{3.9}$$

where we consider that $\log 0 = -\infty$. Intuitively, a solution $\mathbf{s}$ such that the minimum eigenvalue of $2\hat{\Sigma} - \operatorname{diag} \mathbf{s}$ gets too close to 0 will be penalized by this log term.

The following lemma will be helpful to compute the determinant.

Lemma 2. *If* $M = \begin{bmatrix} \xi & \mathbf{y}^\top \\ \mathbf{y} & B \end{bmatrix}$, *then*

$$\begin{aligned} \det(M) &= \det\left(\xi - \mathbf{y}^\top B^{-1} \mathbf{y}\right) \cdot \det(B) \\ &= \left(\xi - \mathbf{y}^\top B^{-1} \mathbf{y}\right) \cdot \det(B) \end{aligned}$$

Proof. It is an immediate consequence of the formula giving the determinant of a block matrix reported in Appendix B.3. □

With subscript notations, this identity can be noted as $\det(M) = \left(M_{1,1} - M_{1^c,1}^\top M_{1^c,1^c}^{-1} M_{1^c,1}\right) \cdot \det(M_{1^c,1^c})$. Employing the same idea as in 3.6, it can further be generalized to

$$\det(M) = \left(M_{j,j} - M_{j^c,j}^\top M_{j^c,j^c}^{-1} M_{j^c,j}\right) \cdot \det(M_{j^c,j^c})$$

for any $j \in \{1, \dots, p\}$. By applying this to $\log\det\left(2\hat{\Sigma} - \operatorname{diag} \mathbf{s}\right)$, we get that for any j,

$$\log\det\left(2\hat{\Sigma} - \operatorname{diag} \mathbf{s}\right) = \log\left(2\hat{\Sigma}_{j,j} - s_j - 4\hat{\Sigma}_{j^c,j}^\top Q_j^{-1} \hat{\Sigma}_{j^c,j}\right) + \log\det(Q_j)$$

where $Q_j = 2\hat{\Sigma}_{j^c,j^c} - \operatorname{diag} \mathbf{s}_{j^c}$ does not depend on s_j.

Algorithm 2 Coordinate ascent with log-barrier

1: **Input:** $\hat{\Sigma}$, barrier coefficient λ, decay μ, $\mathbf{s}^{(0)} = \mathbf{0}_p$
2: $\mathbf{s} = \mathbf{s}^{(0)}$
3: **repeat**
4: **for** $j = 1, \ldots, p$ **do**
5: $Q_j = 2\hat{\Sigma}_{j^c, j^c} - \operatorname{diag} \mathbf{s}_{j^c}$
6: $s_j = \max\left(2\hat{\Sigma}_{j,j} - 4\hat{\Sigma}_{j^c,j}^\top Q_j^{-1} \hat{\Sigma}_{j^c,j} - \lambda,\, 0\right)$
7: **end for**
8: $\lambda = \mu \cdot \lambda$
9: **until** stopping criteria

Again, we start from a feasible solution $\mathbf{s}^{(0)} = \mathbf{0}_p$ and perform coordinate ascent. At each iteration, a new coordinate j is updated and we take advantage of the fact that Q_j doesn't depend on s_j. The function

$$\alpha \mapsto \alpha + \lambda \log\left(2\hat{\Sigma}_{j,j} - \alpha - 4\hat{\Sigma}_{j^c,j}^\top Q_j^{-1} \hat{\Sigma}_{j^c,j}\right)$$

is concave and setting its derivative to 0 yields that its maximum is

$$\alpha^\star = 2\hat{\Sigma}_{j,j} - 4\hat{\Sigma}_{j^c,j}^\top Q_j^{-1} \hat{\Sigma}_{j^c,j} - \lambda$$

The jth coordinate is therefore updated as $s_j \leftarrow \max(\alpha^\star, 0)$. Note in particular that this update maintains the feasibility conditions 3.8 for any value of λ.

?, §11.3 suggest that picking $\lambda = \epsilon/p$ will yield an ϵ-optimal solution. In practice, an initial λ_0 is picked and it is multiplied by a decay rate $0 < \mu < 1$ after each coordinate cycle to refine the solution. This scheme can be proven to converge to the maximum of the modified objective 3.9. It is indeed a particular case of **?**, Theorem 3, which applies in this case as the constraints are simple, i.e. $\mathbf{s} \geq \mathbf{0}$ is of the form $L \leq \operatorname{diag} \mathbf{s} \leq U$.

This log-barrier algorithm is summarized in pseudo-code in Algorithm 2. The bottleneck is the inversion of the matrix $Q_j \in \mathbb{R}^{(p-1)\times(p-1)}$ in line 6 which takes $\mathcal{O}\left(p^3\right)$ steps. As it is done in every inner iteration, the time complexity of Algorithm 2 is $\mathcal{O}\left(n_{\text{iters}} \cdot p^4\right)$ when implemented naively. It is possible to design a version running in $\mathcal{O}\left(n_{\text{iters}} \cdot p^3\right)$ steps instead. To see this, note $A = 2\hat{\Sigma} - \operatorname{diag} \mathbf{s} \in \mathbb{R}^{p\times p}$. At each iteration j, Q_j is a principal sub-matrix of A and its inverse can efficiently be computed (in $\mathcal{O}\left(p^2\right)$ steps) from the one of A [**?**]. On top of that, when a coordinate s_j changes, A is only modified with a rank-1 update. Its new inverse can efficiently be computed ($\mathcal{O}\left(p^2\right)$ steps) using for example Sherman-Morrison formula (see Appendix B) or Cholesky rank-1 updates [**?**] which are more stable numerically. In Appendix C we give a Cython implementation example.

3.4 Low-rank covariance approximation

Covariance estimation in the high-dimensional setting ($p > n$ where n is the number of samples) is challenging as the sample (empirical) covariance is not accurate. On top of that, if p is larger than 20 000 it becomes challenging to solely store the full matrix in the memory of a standard computer. Hopefully, big data matrices and especially correlation matrices often have an underlying low-rank structure, or at least can be approximated adequately by such a low-rank estimate [**?**]. An intuitive explanation is that only a small number of latent features explain most of the data.

3.4.1 Factor model

We approximate the covariance $\hat{\Sigma}$ with the following structure

$$\hat{\Sigma} = D + U\Lambda U^\top \tag{3.10}$$

where $U \in \mathbb{R}^{p\times k}$ has orthogonal columns ($U^\top U = I_k$), and both $D \in \mathbb{R}^{p\times p}$, $\Lambda \in \mathbb{R}^{k\times k}$ are diagonal and psd. k is typically much lower than p to save has much memory and computations as possible.

In the case where X is reasonably large, the decomposition 3.10 for the empirical covariance can be obtained as follows. First factorize $X = P\Delta Q^T$ with SVD [?]. Then (assuming columns are centered),

$$\begin{aligned}\hat{\Sigma} &= \frac{1}{n} X^\top X \\ &= \frac{1}{n} Q\Delta^2 Q^\top\end{aligned}$$

From this, building a rank-k approximation of $\hat{\Sigma}$ is easily done by keeping its top k eigenvalues, that is

$$\Lambda = \Delta^2_{:k,\,:k} \qquad\qquad U = Q_{:,\,:k}$$

Then set $D = \operatorname{diag}^2 \max\left(\hat{\Sigma} - U\Lambda U^\top, \mathbf{0}_{p\times p}\right)$ to be the diagonal psd matrix containing the difference between $\hat{\Sigma}$ and the low-rank approximation. None of these steps requires to form the full covariance matrix $\hat{\Sigma}$, which might not fit in memory.

As pointed out earlier, the empirical covariance estimator performs poorly in high dimension. Shrunk estimators like the one proposed by **?** are proven to be often superior in that situation. Suppose again that $\hat{\Sigma} = \frac{1}{n} X^\top X$ is the empirical estimator. Note $\mu = \operatorname{Tr}(\hat{\Sigma})/p$ and δ a shrinkage coefficient. The shrunk covariance estimator is then given by

$$\hat{\Sigma}_\delta = (1-\delta)\hat{\Sigma} + \delta\mu I$$

The same decomposition strategy can be employed, but this time $\hat{\Sigma}_\delta = Q\big[(1-\delta)\Delta^2/n + \delta\mu I\big]Q^\top$. Finding the optimal coefficient δ can be done in $\mathcal{O}\left(n \cdot p\right)$ steps and without extra memory, provided that we know the decomposition of X.

Randomized SVD algorithms [?] may be employed in the case where computing the exact SVD decomposition of X is too costly, when X doesn't even fit in memory, or simply to speed-up the factorization process. Without the full SVD, computing the optimal shrinkage coefficient δ is more costly but can be done in $\mathcal{O}\left(n^2 \cdot p\right)$ steps and without extra memory. Rather than simply computing a low-rank approximation $U\Lambda U^\top$ and then D, these randomized estimations allow to achieve alternating descent on U and D without constructing $\hat{\Sigma}$. Nonetheless, one single step seems to be often very close to a local minimum. We implement such algorithms and find that they scale moderately well with n and p.

In the following sections, we suppose that the decomposition $\hat{\Sigma} = D + U\Lambda U^\top$ was computed. Even in the (likely) case where the covariance matrix is not exactly diagonal plus low-rank, this approximating structure is general enough to yield satisfying results.

3.4.2 Low-rank coordinate ascent

The complexity of Algorithm 2 can be drastically reduced to $\mathcal{O}\left(n_{\text{iters}} \cdot p \cdot k^2\right)$ by taking advantage of the special structure of $\hat{\Sigma}$. Remember that at each iteration j, only the coordinate j of $\mathbf{s}$ is updated

$$s_j \leftarrow \max(\alpha^\star,\, 0) \quad \text{with} \quad \alpha^\star = 2\hat{\Sigma}_{j,\,j} - 4\hat{\Sigma}^\top_{j^c,\,j} Q_j^{-1} \hat{\Sigma}_{j^c,\,j} - \lambda$$

where $Q_j = 2\hat{\Sigma}_{j^c,\,j^c} - \operatorname{diag}\mathbf{s}_{j^c}$. Note $V = U\sqrt{\Lambda}$ so that $\hat{\Sigma} = D + VV^\top$. Then we get that

$$Q_j = 2D_{j^c,\,j^c} - \operatorname{diag}\mathbf{s}_{j^c} + 2V_{j^c,:}V^\top_{j^c,:}$$

where the subscript ':' denotes that all columns (or rows) are kept. By writing $F_j = 2D_{j^c,\,j^c} - \operatorname{diag}\mathbf{s}_{j^c}$ and $A_j = V^\top_{j^c,:}F_j^{-1}V_{j^c,:}$, the Woodbury formula (see Appendix B.2) gives that

$$\begin{aligned}Q_j^{-1} &= F_j^{-1} - 2F_j^{-1}V_{j^c,:}\big(I_k + 2V^\top_{j^c,:}F_j^{-1}V_{j^c,:}\big)^{-1}V^\top_{j^c,:}F_j^{-1} \\ &= F_j^{-1} - 2F_j^{-1}V_{j^c,:}\big(I_k + 2A_j\big)^{-1}V^\top_{j^c,:}F_j^{-1}\end{aligned}$$

Using this, the inner-loop update $\alpha^\star$ can be written as

$$\begin{aligned}\alpha^\star &= 2\hat{\Sigma}_{j,j} - 4\hat{\Sigma}_{j^c,j}^\top F_j^{-1}\hat{\Sigma}_{j^c,j} - \lambda + 8\hat{\Sigma}_{j^c,j}^\top F_j^{-1} V_{j^c,:}\big(I_k + 2A_j\big)^{-1} V_{j^c,:}^\top F_j^{-1}\hat{\Sigma}_{j^c,j} \\ &= \underbrace{2\hat{\Sigma}_{j,j} - 4V_{j,:}A_jV_{j,:}^\top - \lambda}_{(*)} + \underbrace{8V_{j,:}A_j\big(I_k + 2A_j\big)^{-1}A_jV_{j,:}^\top}_{(**)} \end{aligned} \tag{3.11}$$

where we used the fact that $\hat{\Sigma}_{j^c,j} = D_{j^c,j} + V_{j^c,:}V_{j,:}^\top = V_{j^c,:}V_{j,:}^\top$.

Forming the matrix A_j at each iteration would be very costly. But if we note $F = 2D - \operatorname{diag}\mathbf{s}$ and $A = V^\top F^{-1}V$, it appears that A_j is a rank-1 update of A

$$\begin{aligned} A_j &= V_{j^c,:}^\top F_j^{-1} V_{j^c,:} \\ &= A - F_{j,j}\cdot V_{j,:}^\top V_{j,:} \end{aligned}$$

Therefore, we can simply build A at the beginning and update it efficiently to make A_j. Using this, the term $(*)$ in 3.11 can be easily evaluated in $\mathcal{O}\left(k^2\right)$ steps. The term $(**)$ involves the computation of the inverse of $B_j = I_k + 2A_j$. But again, B_j is a rank-1 update of $B = I_k + 2A$, which can be computed once and updated with Sherman-Morrison (Appendix B) or Cholesky rank-1 updates [**?**]. Finally, when s_j is updated, the coordinate (j, j) of F is changed, which induces a rank-1 update of A. All these operations can be done in $\mathcal{O}\left(k^2\right)$ steps. The matrix F being diagonal, it requires only p floats to be stored and its inversion is trivial.

Algorithm 3 Low-rank coordinate ascent

Input: approximation $\hat{\Sigma} = D + U\Lambda U^\top$, barrier coefficient λ, decay μ, $\mathbf{s}^{(0)} = \mathbf{0}_p$
$\mathbf{s} = \mathbf{s}^{(0)}$
$V = U\sqrt{\Lambda}$
repeat
  for $j = 1, \ldots, p$ **do**
    $F_j = 2D_{j^c,j^c} - \operatorname{diag}\mathbf{s}_{j^c}$
    $A_j = V_{j^c,:}^\top F_j^{-1} V_{j^c,:}$
    $\alpha^\star = 2\hat{\Sigma}_{j,j} - 4V_{j,:}^\top A_j V_{j,:} - \lambda + 8V_{j,:}^\top A_j\big(I_k + 2A_j\big)^{-1}A_jV_{j,:}$
    $s_j = \max\left(\alpha^\star, 0\right)$
  end for
  $\lambda = \mu\cdot\lambda$
until stopping criteria

This scheme is summarized in Algorithm 3. It uses at most $\mathcal{O}\left(p\cdot k\right)$ memory and the time complexity is $\mathcal{O}\left(n_{\text{iters}}\cdot p\cdot k^2\right)$, as stated above. We make a Cython implementation which is available in the released Python package. Because of numerical instabilities, we completed experiments with a version inverting the matrix $I_k + 2A_j$ at each iteration, leading to a higher time complexity of $\mathcal{O}\left(n_{\text{iters}}\cdot p\cdot k^3\right)$. Figure 3.2 shows the convergence times obtained in these experiments, as a function of p and k. It demonstrates that the scalability is much better than with SCS, thanks to the low-rank approximation and the fact that coordinate ascent is particularly well-suited to this optimization problem. Even with 100 000 features, the convergence could be obtained in a few minutes. We hope to prevent numerical instabilities in future versions and attain the $\mathcal{O}\left(n_{\text{iters}}\cdot p\cdot k^2\right)$ time complexity.

3.4.3 Efficient low-rank sampling

In this section, we detail briefly how to sample knockoffs $\tilde{\mathcal{X}}$ once a feasible solution $\mathbf{s}$ is computed, and in the special case where $\hat{\Sigma} = D + U\Lambda U^\top$. As shown in Section 2.1.2, $\tilde{\mathcal{X}}$ is sampled from the following conditional distribution

$$\tilde{\mathcal{X}} \mid \mathcal{X} \sim \mathcal{N}\left(\boldsymbol{\upsilon}, \Upsilon\right), \qquad \text{where} \quad \begin{cases} \boldsymbol{\upsilon} = \mathcal{X} - \mathcal{X}\Sigma^{-1}\operatorname{diag}\{\mathbf{s}\} \\ \Upsilon = \operatorname{diag}\{\mathbf{s}\}\left(2I_{p\times p} - \Sigma^{-1}\operatorname{diag}\{\mathbf{s}\}\right) \end{cases} \tag{3.12}$$

A classical approach to sample $\mathbf{z} \sim \mathcal{N}\left(\boldsymbol{\upsilon}, \Upsilon\right)$ from a multivariate normal distribution is to sample first a vector $\tilde{\mathbf{z}} \in \mathbb{R}^p$ from $\mathcal{N}\left(0, 1\right)$, and then pose $\mathbf{z} = \boldsymbol{\upsilon} + L\tilde{\mathbf{z}}$, where L is a lower Cholesky factorization of

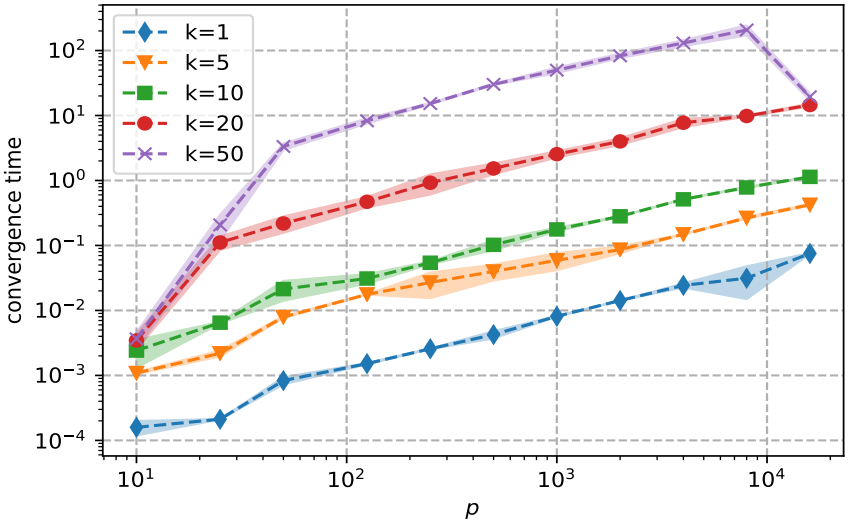

FIGURE 3.2: Convergence times of the low-rank coordinate ascent algorithm 3 as a function of p and k. It confirms the theoretical $\mathcal{O}(n_{\text{iters}} \cdot p \cdot k^3)$ rate. In practice, the algorithm converges much faster to an appropriate solution when using a larger tolerance threshold. See Appendix A.2 for details regarding the random data generation.

Υ. This uses the fact that if $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$, then

$$A\mathbf{x} + \mathbf{b} \sim \mathcal{N}\left(A\boldsymbol{\mu} + \mathbf{b},\, A\Sigma A^\top\right) \tag{3.13}$$

Note that this scheme works even if L is not lower-triangular, but simply satisfies $LL^\top = \Upsilon$.

As Υ is a $p \times p$ matrix we may not want to store it in memory, nor compute a Cholesky factorization (which takes $\mathcal{O}\left(p^3\right)$ steps). We show here how to factorize Υ in a cheap way, requiring only $\mathcal{O}\left(k \cdot p\right)$ memory and $\mathcal{O}\left(k \cdot p^2\right)$ operations. We note $S = \operatorname{diag} \mathbf{s}$, so that $\Upsilon = 2S - S\hat{\Sigma}^{-1}S$. Using the low-rank structure $\hat{\Sigma} = D + U\Lambda U^\top$, the Woodbury formula (see Appendix B.2) gives

$$\begin{aligned}\hat{\Sigma}^{-1} &= (D + UU^\top)^{-1} \\ &= D^{-1} - D^{-1}U(I_k + U^\top D^{-1}U)^{-1}U^\top D^{-1}\end{aligned}$$

Note $L \in \mathbb{R}^{k\times k}$ the lower Cholesky factorization of $(I_k + U^\top D^{-1}U)^{-1}$ (which can be computed efficiently if k is small), $V = SD^{-1}UL \in \mathbb{R}^{p\times k}$, and $C = 2S - SD^{-1}S$. Then the covariance reduces to

$$\Upsilon = C + VV^\top$$

where C is diagonal and $VV^\top$ has rank at most k. C is not necessarily psd, which will force us to sample from a complex normal distribution. Let H be a complex square-root of C (thus, potentially with imaginary numbers on the diagonal), and $P = H^{-1}V \in \mathbb{R}^{p\times k}$. Then,

$$\Upsilon = H\left(I_{p\times p} + PP^\top\right)H^\top$$

$\left(I_{p\times p} + PP^\top\right)$ can be factorized in the following way. Note

$$W = \left(I_{k\times k} + \sqrt{I_{k\times k} + P^\top P}\right)^{-1} \in \mathbb{R}^{k\times k}$$

where the square-root is a $k \times k$ matrix root (which can again be computed efficiently if k is small). Then

$$I_{p\times p} + PP^\top = BB^\top, \qquad \text{where} \quad B = I_{p\times p} + PWP^\top$$

Finally, we note $M = HB$ and we have that $\Upsilon = MM^\top$. B is $p \times p$ but we never have to fully evaluate it; instead only W and P can be stored and matrix multiplications are then at most $p \times k$.

M is a complex matrix so we cannot directly use the property 3.13. But if $\mathcal{X} \sim \mathcal{N}(\mathbf{0}, I)$ and $\mathcal{Y} = i\mathcal{X}$, then $\mathrm{Im}\,(M\mathcal{Y}) \sim \mathcal{N}(\mathbf{0}, MM^\top = \Upsilon)$. Using this scheme, we can draw dozens of thousands knockoff samples in a few seconds without explicitely building $\hat{\Sigma}$ nor Υ.

Chapter 4

Fast statistics computation

As described in Chapter 2, the computation of statistics on the concatenated feature matrix is a key step of the knockoff selection process. We discussed in Section 2.2 the fact that a large variety of statistics control the FDR, provided that the *flip-sign* property 3 is satisfied. However, the power is highly impacted depending on how suitable for the data the statistics are.

In this chapter, we note $X \in \mathbb{R}^{n \times p}$ the original design matrix, and we suppose that we sampled valid knockoff features $\tilde{X} \in \mathbb{R}^{n \times p}$ (respecting conditions S.1 and S.2). The target vector is noted $\mathbf{y} \in \mathbb{R}$.

4.1 Lasso is good but slow

Experimentally, cross-validating a regressor (or classifier) against the data $\left[X, \tilde{X}\right]$, keeping its coefficients in absolute value, and aggregating them as described in Subsection 2.2.2 yields excellent results. Most classical regression models scale nicely as p increases; $\mathcal{O}(n \cdot p)$ for Lasso [?] and logistic regression [?], $\mathcal{O}\left(n \cdot p^2\right)$ for support vector machines (SVM) [?]. In practice, the Lasso is observed to produce the highest statistical power in many cases. But all of these models have hyper-parameters that need to be tuned, usually through cross-validation, and with a naive grid search. Tuning the estimator tends to lead to a higher power than picking ad hoc hyper-parameters, but it increases the time complexity of the training phase by a lot. Figure 4.1 shows the time required a tune the penalty coefficient of a Lasso model, using the algorithm LARS [?]. For 5 000 features (in total, with the knockoffs), it already needs ≈ 20 minutes to compute the statistics.

Many datasets, like the ones we will consider for experiments in Chapter 5, contain dozens of thousands of features. For this reason, we try to develop faster techniques and statistics in the following sections (potentially at the cost of a lower power).

4.2 Multi-stage procedures

A natural way to address the high number of features is to split the selection into several steps. If p is high, you may want to reduce the number of features by cutting a large proportion with a cheap and fast procedure first (for example univariate regression), and use more sophisticated methods afterwards on the remaining set of features. However, methods that control the FDR like BH, BY, and the knockoffs are not guaranteed to work in this feature truncation approach anymore. A modification of the BH procedure is proposed by **?** to address the issue of this multi-stage evaluation. They prove that their process effectively controls the FDR on the entire set of features, as desired.

In our Python implementation mentioned in Section 2.5, we propose a `MultiStepsSelector` class to do that kind of multi-steps selections. We are well-aware that no theoretical guarantee exists in the case of the knockoff procedure. But we observed on synthetic data that in practice the false discovery proportions obtained were not too far away from the target value. Making a theoretical analysis will be the subject of future work.

4.3 Sparse naive Bayes

In this section, we present a sparse version of naive Bayes elaborated by **?**. It allows to perform very fast and accurate feature selection. We restrict ourselves to the specific case of binary classification, so $\mathbf{y} \in \{-1, 1\}^n$, but most key results can naturally be extended to the multiple classes setting. The

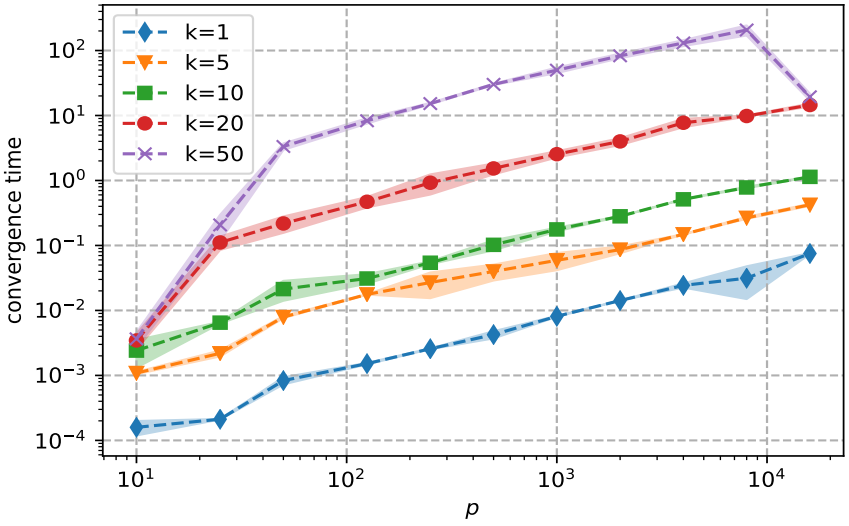

FIGURE 4.1: Convergence time to cross-validate the parameter λ of a Lasso model, as a function of the feature space dimension p (original + knockoff features). Benchmark done in Python using the class `LassoCV` from `sklearn.linear_model`. The data is the one described in Section 5.1.1, and contains 2 695 samples.

negative and the positive classes are noted $\mathcal{C}_-$ and $\mathcal{C}_+$ respectively, and will be indifferently substituted with their respective labels.

4.3.1 Reminders on vanilla naive Bayes

Naive Bayes was first introduced by **?** for text documents classification. It is a simple model assuming the independence of the features, given the label. Despite this simple presupposition (that is very likely to fail in practice), naive Bayes remains an engaging model for large scale datasets because of its low complexity. The time complexity to train a model is asymptotically $\mathcal{O}(n \cdot p)$, where n and p are the number of samples and the number of features respectively. Another appealing propriety is that naive Bayes can be trained in an online fashion, as new data points come in a sequential order. This aspect can be particularly helpful if the dataset doesn't fit in memory. Furthermore, distributed implementations could even be considered in order to speed up the learning process.

General settings. Note $(\mathcal{X}, \mathcal{Y})$ the pair of random variables that generated the samples and targets X and $\mathbf{y}$. The goal is to explain $\mathbf{y}$ given X, that is, finding the posterior probabilities $\Pr(\mathcal{C}_\pm \mid \mathcal{X})$. From these probabilities, a new observation $\mathbf{x} \in \mathbb{R}^p$ is classified according to the highest posterior probability between the two classes

$$y(\mathbf{x}) = \underset{\mathcal{C} \in \{\mathcal{C}^-, \mathcal{C}^+\}}{\arg\max} \Pr(\mathcal{C} \mid \mathbf{x}) \tag{4.1}$$

To do so, we combine the use of Bayes rule and make the (very) naive assumption that all the features are independent given the class, that is

$$\Pr(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm) = \prod_{j=1}^{p} \Pr(\mathcal{X}_j = x_j \mid \mathcal{C}_\pm), \qquad \Pr(\mathcal{C}_\pm \mid \mathcal{X} = \mathbf{x}) = \frac{\Pr(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm) \cdot \Pr(\mathcal{C}_\pm)}{\Pr(\mathcal{X} = \mathbf{x})} \tag{4.2}$$

On the right hand side, the denominator $\Pr(\mathcal{X} = \mathbf{x})$ does not depend on the class $\mathcal{C}_\pm$. It is therefore not required to evaluate it in order to perform the inference described in 4.1. Thus, we only need to estimate the probabilities $\Pr(\mathcal{C}_\pm)$ and $\Pr(\mathbf{x} \mid \mathcal{C}_\pm)$. The former are simply data averages, i.e. the frequencies of the positive and negative classes in the observed data. As for the latter probabilities $\Pr(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm) = \prod_{j=1}^{p} \Pr(\mathcal{X}_j = x_j \mid \mathcal{C}_\pm)$, they can be modeled by a plethora of distribution families, depending on the prior knowledge we have on the data. The distribution is typically parametrized by some vector $\boldsymbol{\theta} \in \mathbb{R}^m$. which allows the probabilities to be computed by maximizing the likelihood $\mathcal{L}$ of

the observed data. Equivalently we may also maximize the log-likelihood $\mathcal{LL} = \log \mathcal{L}$.

$$\mathcal{LL}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \Pr(\mathcal{X} = \mathbf{x}_i \mid y_i ; \boldsymbol{\theta})$$

We present now 3 meaningful cases, where the prior distributions of $\mathcal{X} \mid \mathcal{C}_\pm$ are either gaussian, bernoulli or multinomial. Let $\mathcal{I}_\pm = \{i \in \{1, \dots, n\} \mid y_i = \mathcal{C}_\pm\}$ be the sets containing the indexes of the positive and negative data points. We also note for each class $\mathcal{C}_\pm$ its cardinality and empirical sum respectively, as follows:

$$n^\pm = |\mathcal{I}_\pm|, \qquad \mathbf{f}^\pm = \sum_{i \in \mathcal{I}_\pm} \mathbf{x}_i$$

Gaussian naive Bayes In this case the observed data conditioned on its label is modeled by a gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^\pm, \Sigma^\pm)$. It is the most common configuration because it can account for continuous data and the normal distribution constitutes a decent prior in many cases by virtue of its high entropy. Note that the covariance $\Sigma^\pm \in \mathbb{R}^{p \times p}$ is diagonal as a result of the independence assumption that we made.

$$\Pr(\mathcal{X} = \mathbf{x} \mid C_\pm) = \frac{1}{\sqrt{(2\pi)^p \det(\Sigma_\pm)}} \exp\Big(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_\pm)^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_\pm)\Big)$$

By denoting $\sigma_j = \Sigma_{jj}$, the log-likelihood can be written

$$\begin{aligned}\mathcal{LL}_g(\boldsymbol{\mu}_+, \boldsymbol{\sigma}_+, \boldsymbol{\mu}_-, \boldsymbol{\sigma}_-) = \sum_{j=1}^{p} \Bigg[&\sum_{i \in \mathcal{I}_+} -\frac{1}{2}\log(2\pi) - \log \sigma_j^+ - \frac{(x_j - \mu_j^+)^2}{2{\sigma_j^+}^2} \\ &+ \sum_{i \in \mathcal{I}_-} -\frac{1}{2}\log(2\pi) - \log \sigma_j^- - \frac{(x_j - \mu_j^-)^2}{2{\sigma_j^-}^2} \Bigg]\end{aligned}$$

Even if $\mathcal{LL}_g$ is not concave, it maximizer admits a closed-form solution (readily shown to be the point where the gradient is null)

$$\boldsymbol{\mu}^\pm = \frac{\mathbf{f}^\pm}{n^\pm}, \qquad \boldsymbol{\sigma}^\pm = \sqrt{\frac{1}{n^\pm} \sum_{i \in \mathcal{I}^\pm} (\mathbf{x}_i - \boldsymbol{\mu}^\pm)^2}$$

Bernoulli naive Bayes The Bernoulli distribution assumes that the design matrix is binary, that is $X \in \{0, 1\}^{n \times p}$. Even though this situation isn't very prevalent in practice, it has a simple and elegant solution. More formally, we assume the existence of $\boldsymbol{\theta}^+, \boldsymbol{\theta}^- \in (0, 1)^p$ such that for any data point $\mathbf{x} \in \mathbb{R}^p$ the conditional probabilities are given by

$$\Pr(\mathcal{X}_j = x_j \mid \mathcal{C}_\pm) = (\theta_j^\pm)^{x_j} \cdot (1 - \theta_j^\pm)^{1 - x_j}$$

It yields that $\log \Pr(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm) = \mathbf{x}^\top \log \boldsymbol{\theta}^\pm + (\mathbf{1} - \mathbf{x})^\top \log(\mathbf{1} - \boldsymbol{\theta}^\pm)$ and finally

$$\begin{aligned}\mathcal{LL}_b(\boldsymbol{\theta}^+, \boldsymbol{\theta}^-) = \mathbf{f}_+^\top \log \boldsymbol{\theta}^+ + (n_+ \mathbf{1} - \mathbf{f}_+)^\top \log\big(\mathbf{1} - \boldsymbol{\theta}^+\big) \\ + \mathbf{f}_-^\top \log \boldsymbol{\theta}^- + (n_- \mathbf{1} - \mathbf{f}_-)^\top \log\big(\mathbf{1} - \boldsymbol{\theta}^-\big)\end{aligned} \tag{4.3}$$

The independence assumption makes the optimization problem decomposable across features; it reduces to p simpler maximizations, each of them being concave and admitting a closed-form solution. Finally, we find that

$$\theta_\pm^\star = \frac{f^\pm}{n^\pm}, \qquad \text{which are simply the averages of each class.}$$

Multinomial naive Bayes Multinomial naive Bayes generalizes the Bernoulli version as it supposes that $X \in \mathbb{N}^{n \times p}$ is generated by the following underlying distribution

$$\Pr(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm) = \frac{\big(\sum_{j=1}^p x_j\big)!}{\prod_{j=1}^p x_j!} \cdot \prod_{j=1}^{p} (\theta_j^\pm)^{x_j} \tag{4.4}$$

In the special case where $X \in \{0, 1\}^{n \times p}$ we recover the above Bernoulli formulation. It is parametrized by $\boldsymbol{\theta}^+$, $\boldsymbol{\theta}^- \in (0, 1)^p$, and they must satisfy $\mathbf{1}^\top \boldsymbol{\theta}^+ = \mathbf{1}^\top \boldsymbol{\theta}^- = 1$ for 4.4 to be a proper distribution. Note that this model is still valid in the more general case where we only assume the data to be non-negative, $X \in \mathbb{R}_+^{n \times p}$. Hence, it is not as restrictive as it may appear at first sight, as it is applicable to a large number of datasets. The log probability is given by

$$\log \Pr\left(\mathcal{X} = \mathbf{x} \mid \mathcal{C}_\pm\right) = \mathbf{x}^\top \log \boldsymbol{\theta}_\pm + \log \frac{\left(\sum_{j=1}^p x_j\right)!}{\prod_{j=1}^p x_j!}$$

and the log-likelihood reduces to (after removing the constant terms)

$$\mathcal{LL}_m\left(\boldsymbol{\theta}^+, \boldsymbol{\theta}^-\right) = \mathbf{f}_+^\top \log \boldsymbol{\theta}^+ + \mathbf{f}_-^\top \log \boldsymbol{\theta}^- \tag{4.5}$$

which is again decomposable across features. It turns out that the optimums are

$$\boldsymbol{\theta}_\pm^\star = \frac{\mathbf{f}^\pm}{\mathbf{1}^\top \mathbf{f}^\pm}$$

In the models presented above, the time complexity to train the naive Bayes classifier is $\mathcal{O}(n \cdot p)$, and the solutions can be computed in closed-form, which makes the effective computation cost very low. In comparison, no closed-form solution exist for the Lasso [**?**], for logistic regression [**?**], and for SVMs [**?**].

Decision boundary Given a new data point $\mathbf{x} \in \mathbb{R}^p$, we wish to attribute it the most probable label $y \in \{\mathcal{C}^-, \mathcal{C}^+\}$. No matter what model parametrized by $\boldsymbol{\theta}$ was chosen, 4.1 reduces to

$$\begin{aligned} y &= \underset{\mathcal{C} \in \{\mathcal{C}^-, \mathcal{C}^+\}}{\arg\max} \Pr\left(\mathcal{C} \mid \mathbf{x}\right) \\ &= \operatorname{sign}\left[\log \frac{\Pr(\mathcal{C}^+)}{\Pr(\mathcal{C}^-)} + \log \frac{\Pr(\mathbf{x} \mid \mathcal{C}^+)}{\Pr(\mathbf{x} \mid \mathcal{C}^-)}\right] \end{aligned}$$

In the cases of *bernoulli* (4.3.1) and *multinomial* (4.3.1) naive Bayes, there exist $v \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^p$ such that $y = \operatorname{sign}(v + \mathbf{w}^\top \mathbf{x})$. In these two special cases, the decision boundary is a hyperplane (which doesn't happen for gaussian naive Bayes). For both of them v has the same value, and by noting $\mathbf{w}_b$ and $\mathbf{w}_m$ the weights for the bernoulli and the multinomial case respectively, we have

$$v = \log \frac{\Pr(\mathcal{C}^+)}{\Pr(\mathcal{C}^-)}, \qquad \begin{cases} \mathbf{w}_b = \log(\boldsymbol{\theta}^+ \odot (\mathbf{1} - \boldsymbol{\theta}^-)) - \log(\boldsymbol{\theta}^- \odot (\mathbf{1} - \boldsymbol{\theta}^+)) \\ \mathbf{w}_m = \log \boldsymbol{\theta}^+ - \log \boldsymbol{\theta}^- \end{cases}$$

4.3.2 Sparse naive Bayes (SNB)

By adding a sparsity constraint in the Bernoulli and the multinomial optimization problems portrayed above **?** introduced a version of naive Bayes that imposes the weight vector $\mathbf{w}$ to have a number of non-zero entries under a certain threshold $k \in \mathbb{N}$. This property makes naive Bayes employable to perform feature selection, by keeping only the features whose weights are non-zero.

Problem statement. Let $0 \leq k \leq p$ be a desired level of sparsity. We wish to train a naive Bayes classifier whose decision boundary depends on at most k features. As shown in the previous section 4.3.1, for both bernoulli and multinomial naive Bayes there exist $v \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^p$ such that $y(\mathbf{x}) = \operatorname{sign}(v + \mathbf{w}^\top \mathbf{x})$ for a data point $\mathbf{x} \in \mathbb{R}^p$. Furthermore, the jth entry of the decision vector $\mathbf{w}$ is null if and only if $\boldsymbol{\theta}_j^- = \boldsymbol{\theta}_j^+$, where $\boldsymbol{\theta}^-$ and $\boldsymbol{\theta}^+$ are the parameters of the loss functions $\mathcal{LL}_b$ and $\mathcal{LL}_m$. Naturally, imposing the constraint $\left\|\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-\right\|_0 \leq k$ in the optimization problems will yield a weight vector $\mathbf{w}$ with the desired sparsity. The optimization problems for Bernoulli and multinomial SNB (shortened BSNB and MSNB respectively) can be phrased as follows:

$$\begin{aligned} \underset{\boldsymbol{\theta}^+, \boldsymbol{\theta}^-}{\text{maximize}} \quad & \mathcal{LL}_\text{b}\left(\boldsymbol{\theta}^+, \boldsymbol{\theta}^-\right) = \mathbf{f}_+^\top \log \boldsymbol{\theta}^+ + (n_+ \mathbf{1} - \mathbf{f}_+)^\top \log\left(\mathbf{1} - \boldsymbol{\theta}^+\right) \\ & \qquad\qquad + \mathbf{f}_-^\top \log \boldsymbol{\theta}^- + (n_- \mathbf{1} - \mathbf{f}_-)^\top \log\left(\mathbf{1} - \boldsymbol{\theta}^-\right) \\ \text{subject to} \quad & \left\|\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-\right\|_0 \leq k. \end{aligned} \tag{BSNB}$$

$$\begin{aligned} \underset{\boldsymbol{\theta}^+,\,\boldsymbol{\theta}^-}{\text{maximize}} \quad & \mathcal{LL}_{\mathrm{m}}\left(\boldsymbol{\theta}^+,\,\boldsymbol{\theta}^-\right) = \mathbf{f}_+^\top \log \boldsymbol{\theta}^+ + \mathbf{f}_-^\top \log \boldsymbol{\theta}^- \\ \text{subject to} \quad & \left\|\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-\right\|_0 \leq k \quad \text{and} \quad \mathbf{1}^\top \boldsymbol{\theta}^+ = \mathbf{1}^\top \boldsymbol{\theta}^- = 1. \end{aligned} \tag{MSNB}$$

Main results and resolution. Surprisingly, despite the combinatorial constraints these optimizations problems can be (approximately) solved very efficiently, with an additional minor cost compared to vanilla naive Bayes. Especially for the Bernoulli case, an optimal solution can be computed in closed-form as shown in Theorem 2.

Theorem 2. *Suppose that $X \in \{0,1\}^{n\times p}$ is modeled by the Bernoulli distribution. Then, the exact solution to the problem BSNB can be computed. First, define $\mathbf{t}$ and $\mathbf{u}$ as follows*

$$\mathbf{t} = (\mathbf{f}^+ + \mathbf{f}^-) \odot \log\left(\frac{\mathbf{f}^+ + \mathbf{f}^-}{n}\right) + (n\mathbf{1} - \mathbf{f}^+ - \mathbf{f}^-) \odot \log\left(\mathbf{1} - \frac{\mathbf{f}^+ + \mathbf{f}^-}{n}\right)$$
$$\mathbf{u} = \mathbf{f}^+ \odot \log\frac{\mathbf{f}^+}{n^+} + (n^+\mathbf{1} - \mathbf{f}^+) \odot \log(\mathbf{1} - \frac{\mathbf{f}^+}{n^+}) + \mathbf{f}^- \odot \log\frac{\mathbf{f}^-}{n^-} + (n^-\mathbf{1} - \mathbf{f}^-) \odot \log(\mathbf{1} - \frac{\mathbf{f}^-}{n^-})$$

Let $\mathcal{I}$ be the set of $p-k$ smallest elements of $\mathbf{u}-\mathbf{t}$. The optimal $\boldsymbol{\theta}^-$, $\boldsymbol{\theta}^+$ are given by

$$\theta_j^{-\star} = \theta_j^{+\star} = \frac{1}{n}(f_j^+ + f_j^-)\ \forall j \in \mathcal{I}, \qquad \theta_j^{\pm\star} = \frac{f_j^\pm}{n^\pm}\ \forall j \notin \mathcal{I}$$

In total, the maximizer can be found in $\mathcal{O}\left(n \cdot p + p \cdot \log k\right)$ steps, which is very close to the cost $\mathcal{O}\left(n \cdot p\right)$ of naive Bayes.

In the multinomial case, there is no closed-form solution, but a near-optimal one can be obtained as stated in Theorem 3.

Theorem 3. *Suppose that $X \in \mathbb{R}_+^{n\times p}$ is generated by the multinomial distribution. Define $\phi_k : \alpha \mapsto s_k(\mathbf{h}(\alpha)) + C$ where C is some constant, s_k the sum of the k largest values of a vector, and*

$$\begin{aligned} \mathbf{h}(\alpha) = \mathbf{f}_+ \odot \log \mathbf{f}_+ + \mathbf{f}_- \odot \log \mathbf{f}_- - (\mathbf{f}_+ + \mathbf{f}_-) \odot \log(\mathbf{f}_+ + \mathbf{f}_-) \\ - \mathbf{f}_+ \log \alpha - \mathbf{f}_- \log(1-\alpha) \end{aligned}$$

ϕ_k is the dual of MSNB and can be minimized very quickly (with bisection for example) as it is a one-dimensional convex function. Let $\alpha^\star$ be its minimizer, $\mathcal{I}$ the set of the $p-k$ smallest entries of $\mathbf{h}(\alpha^\star)$, and $B_\pm = \sum_{j\notin\mathcal{I}} f_i^\pm$. A primal point can be reconstructed as follows:

$$\theta_j^{+\star} = \theta_j^{-\star} = \frac{f_j^+ + f_j^-}{\mathbf{1}^\top(\mathbf{f}^+ + \mathbf{f}^-)}\ \forall j \in \mathcal{I}, \qquad \theta_j^{\pm\star} = \frac{B_+ + B_-}{B_\pm}\frac{f_j^\pm}{\mathbf{1}^\top(\mathbf{f}^+ + \mathbf{f}^-)}\ \forall j \notin \mathcal{I}$$

Furthermore, it holds that $\psi(k-4) \leq \phi(k) \leq \psi(k) \leq \phi(k+4)$, implying that the duality gap is small if $\psi(k) - \psi(k-4)$ is small.

Experimentally, the duality gap quickly converges to 0 as k increases, and the reconstructed primal point is near-optimal. The time complexity is once again $\mathcal{O}(n\cdot p + p\cdot\log k)$, which is a minor additional cost compared to plain naive Bayes.

The authors experiment SNB on several text datasets. They obtain competitive test accuracies, while training their models several order of magnitude faster than the Lasso. For these reasons, SNB appears as an attractive alternative to the Lasso for the computation of statistics in the knockoff procedure 2.2.

4.4 Other fast statistics

4.4.1 Based on the Lasso Signed Max

Remember the Lasso Signed Max (LSM) statistics defined in 4.6.

$$z_j = \sup\{\lambda \mid \hat{\boldsymbol{\beta}}_j(\lambda) \neq 0\}, \qquad \hat{\boldsymbol{\beta}}(\lambda) = \underset{\mathbf{b}}{\arg\min}\ \frac{1}{2}\left\|\mathbf{y} - \left[X, \tilde{X}\right]\mathbf{b}\right\|_2^2 + \lambda\left\|\mathbf{b}\right\|_1 \tag{4.6}$$

Chapter 5

Experiments

In this chapter, we show experiments performed on genetic data. That kind of data is usually high dimensional. For this reason, the tools developed in the previous chapters are particularly suited to solve it.

5.1 Setup

5.1.1 Genetic data

[1]

# of samples	# of features	N
2 695	35 238	13

Table 5.1: ARCHS4 dataset characteristics.

5.2 Results

5.2.1 Applications

The apparent low complexity of sparse naive Bayes compared to ℓ_1-penalized methods such as Lasso, logistic regression or SVMs makes is appealing for very large scale datasets. We mention here a few applications to which we will come back later[2].

5.3 Criteo dataset

This experiments is not related to the knockoffs framework, but shows the very good scalability of sparse naive Bayes presented in Section 4.3. As part of a Kaggle competition *Display Advertising Challenge*[3] in mid-2014, CriteoLabs shared log data collected over one week[4] whose features were undisclosed for confidentiality purposes. Main characteristics of this dataset can be found in Table 5.2. It consists in

Samples	Total features	Numerical features	Categorical features	Features after encoding
45 840 617	39	13	26	33 762 590

Table 5.2: Criteo dataset characteristics. Even though the number of features is small, most categorical features have millions of categories. It makes the training of predicting models particularly challenging as it requires several dozens of GB of memory.

≈ 45 millions of display ads with 39 features, and a boolean label describing whether or not the ad was clicked by a customer. Among these 39 features, 26 are categorical and a classical one-hot encoding would end up in millions of features. This makes the Criteo dataset challenging, as it doesn't fit in the random-access memory after encoding, and potentially not in the mass storage of a standard computer

[1] Data available at this address, under the filename `human_matrix.h5 v8`.
[2] An implementation of sparse naive Bayes can be found here.
[3] The Kaggle competition can be found at this address.
[4] The competition's dataset can be downloaded at this address.

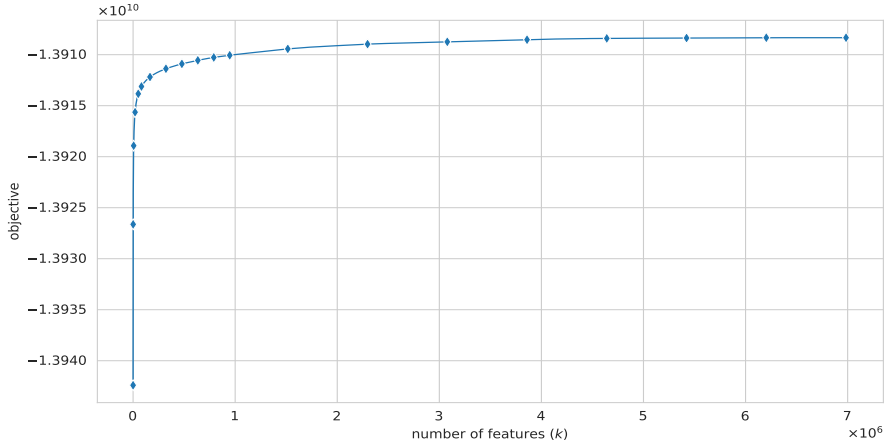

FIGURE 5.1: Optimal value of the MSNB optimization problem on the Criteo dataset as a function of the sparsity parameter k. Only 1–2 millions of the features explain most of the target vector (elbow heuristic).

either. Even on a small subset of the features, say 10%, selecting important features using Lasso or ℓ_1-penalized logistic regression isn't realistic. One-hot encoding isn't adapted to that situation. Another approach would be to one-hot encode for each categorical feature only the most frequent categories, and put the rest in a category *other*. The winners of the Kaggle competition used the hashing trick. It consists in choosing an encoding space size m, for example $m = 2^{20}$, and defining some hash function h: Categories $\to \{0, \dots, m-1\}$. This approach has several notable advantages. First, the final encoded feature space m can be adapted depending on the needs and on the computing power. It may be used in an online fashion without a first pass (that would be required for one-hot encoding in order to figure out all the existing categories). Lastly, it naturally handles new labels in the test set that were unseen in the train set (which would typically need a special *other* category in the one-hot encoding setting). Collisions between categories are likely to happen, and even collisions between categories from different features.

We present here in Figure 5.1. $m = 2^{24} = 16777216$ All the computations are done on a standard workstation (16GB, Intel Core i7 3.60GHz × 8). Sparse naive Bayes requires data averages to run, i.e. the sums of the negative and of the positive points. This part is time consuming but once these sums are computed they can be reused for any sparsity level k. Using a light hash function and PyPy, they were obtained in around 20 minutes. Only roughly half of the 2^{24} hash features were hit by the hash function. Then, SNB optimum are computed for 1200 log-spaced points using a Python and NumPy implementation of SNB in 1 hour. In this situation, what makes SNB particularly appealing is the fact that at no moment we need to load the full dataset in memory. We are only computing data averages whose shape are much smaller than the full matrix. Note also that most tasks could even be distributed to speed up the computations.

Genetic and fMRI data

Conclusion

In this master thesis, we investigated high-dimensional feature selection in the gaussian knockoffs framework setting. In high dimension, nearly all the steps of the selection procedure become challenging, and in particular the construction of the knockoff features, and the computation of statistics associated to each aggregated feature.

Building knockoffs inducing a high statistical power relies on the optimization of a semidefinite program. Hopefully, the structure and the simple bounds of this SDP allow us to perform coordinate ascent which is guaranteed to converge in this case. On top of that, a low-rank approximation structure speeds up the computations by several order of magnitude. Experiments show that this approach keeps a reasonably high power.

Applications to a wider range of datasets, such as fMRI or will be explored in future work. GPU acceleration

When computing statistics with sparse naive Bayes, the sparsity level k has to be tuned, and the calculations have to be done from scratch for every single value. For the Lasso, the penalty coefficient λ can efficiently be tuned with the LARS algorithm **?**. We would like to find a similar algorithm allowing t

Appendices

Appendix A

Data generation details

A.1 SCS and MOSEK convergence times

Experiments were run in Python using CVX [?] through the Python wrapper CVXPY [?]. Table A.1 shows the values of p that were used. We restrained the parameter p for MOSEK as it didn't scale as well as SCS. For each p, 5 replications were performed with matrices of the form $M = D + U\Lambda U^\top$ where

SCS	MOSEK
10	10
25	25
50	50
125	125
250	150
500	
1000	
2000	

Table A.1: Orders of matrices used.

D is diagonal, $\operatorname{diag} D \sim \mathcal{U}[0, 1]$, $U \in \mathbb{R}^{p\times k}$ for $k = 1, 5, 10, 20, 50$ are orthonormal random matrices, and $\Lambda \in \mathbb{R}^{k\times k}$ is diagonal and $\operatorname{diag} \Lambda^2 \sim \mathcal{U}[0, 1]$. No significant difference was measured when changing k, even for large values. The convergence is however much slower if D is scaled up.

A.2 Coordinate ascent

The random data is generated using the same scheme presented in Section A.1 for SCS. As the method scales better, additional sizes $p = 4000$, 8000 and 16 000 are tested. We use a tolerance threshold of 10^{-6} such that the obtained solution is very close to the one returned by SCS and MOSEK. In practice, very good solutions are attained much faster with a larger tolerance.

Appendix B

Linear algebra notes

In this section, we state and prove some useful linear algebra facts.

B.1 Schur complement and positive definiteness

Let $X = \begin{bmatrix} A & B^\top \\ B & C \end{bmatrix}$ be a symmetric matrix where A and C are invertible.

Definition 6. *The Schur complements $X_{/A}$ and $X_{/C}$ are defined as follows:*

$$X_{/A} = C - BA^{-1}B^\top, \qquad X_{/C} = A - B^\top C^{-1}B$$

Schur complements give a way to characterize the fact that X is positive definite and positive semidefinite.

Lemma 3. *The following three properties are equivalent:*

1. $X \succ \mathbf{0}$
2. $A \succ \mathbf{0}$ *and* $X_{/A} \succ \mathbf{0}$
3. $C \succ \mathbf{0}$ *and* $X_{/C} \succ \mathbf{0}$

As well as the three following ones:

1. $X \succeq \mathbf{0}$
2. $A \succ \mathbf{0}$ *and* $X_{/A} \succeq \mathbf{0}$
3. $C \succ \mathbf{0}$ *and* $X_{/C} \succeq \mathbf{0}$

Proof. X can be factorized as follows

$$X = \begin{bmatrix} I & \mathbf{0} \\ B^\top C^{-1} & I \end{bmatrix}^\top \begin{bmatrix} A - B^\top C^{-1}B & \mathbf{0} \\ \mathbf{0} & C \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ B^\top C^{-1} & I \end{bmatrix}$$

which means that $X = Q^\top DQ$ where D is a block-diagonal matrix. Therefore $X \succ \mathbf{0}$ (resp. $\succeq \mathbf{0}$) if and only if $D \succ \mathbf{0}$ (resp. $\succeq \mathbf{0}$), which is equivalent to its diagonal blocks to be $\succ \mathbf{0}$ (resp. $\succeq \mathbf{0}$). To get the same result with the complement $X_{/A} = C - BA^{-1}B^\top$, we use the factorization

$$X = \begin{bmatrix} I & \mathbf{0} \\ BA^{-1} & I \end{bmatrix} \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & C - BA^{-1}B^\top \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ BA^{-1} & I \end{bmatrix}^\top$$

□

Consider the particular partition $X = \begin{bmatrix} \xi & \mathbf{y}^\top \\ \mathbf{y} & B \end{bmatrix}$ where B is invertible. Then a direct application of Lemma 3 gives that

$$\begin{cases} X \succ \mathbf{0} \iff B \succ \mathbf{0} \text{ and } \xi > \mathbf{y}^\top B^{-1}\mathbf{y} \\ X \succeq \mathbf{0} \iff B \succ \mathbf{0} \text{ and } \xi \geq \mathbf{y}^\top B^{-1}\mathbf{y} \end{cases}$$

More results regarding Schur complements can be found in **?**. A generalization involving pseudo-inverses of A and C when they are singular is possible.

B.2 Sherman–Morrison–Woodbury formula

The Sherman-Morrison formula gives a way to calculate the inverse of the sum of an invertible matrix $M \in \mathbb{R}^{p\times p}$ and a rank-1 update $uv^\top$.

Lemma 4. *(Sherman-Morrison) Let $u, v \in \mathbb{R}^p$. Then,*

$$\left(M + uv^\top\right)^{-1} = M^{-1} - \frac{M^{-1}uv^\top M^{-1}}{1 + v^\top M^{-1}u}$$

It can be generalized to a rank-k update UV as follows.

Lemma 5. *(Woodbury) Let $U, V \in \mathbb{R}^{p\times k}$. Then,*

$$\left(M + UV^\top\right)^{-1} = M^{-1} - M^{-1}U\left(I + V^\top M^{-1}U\right)^{-1}V^\top M^{-1}$$

Proofs and further details regarding these formulas can be found in **?**.

B.3 Block matrix determinant

Lemma 6. *Let M be partitioned as $M = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}$ where P and S are nonsingular. Then,*

$$\begin{cases} \det M = \det(S)\det(P - QS^{-1}R) \\ \det M = \det(P)\det(R - RP^{-1}Q) \end{cases}$$

Proof. Note the following block-matrix identity

$$\begin{bmatrix} P & Q \\ R & S \end{bmatrix}\begin{bmatrix} I & \mathbf{0} \\ -S^{-1}R & I \end{bmatrix} = \begin{bmatrix} P - QS^{-1}R & Q \\ \mathbf{0} & S \end{bmatrix}$$

The fact that the determinant of a block-triangular matrix is the product of the determinants of the diagonal blocks gives the result. A similar factorization holds for the second identity. □

Appendix C

Cython acceleration

Because of the overhead of Python, we opted to implement SDP solvers in Cython [?], using SciPy's BLAS and LAPACK interfaces for Cython. Code 4 is an example of implementation of Algorithm 2.

Code 4 Full coordinate ascent implementation with Cython, BLAS and LAPACK

```
import numpy as np
cimport cython
cimport numpy as np
from scipy.linalg.cython_blas cimport dsyr, dsymv, ddot

@cython.boundscheck(False)
@cython.wraparound(False)
@cython.nonecheck(False)
cdef _full_rank(int p, double[:, ::1] Sigma, int max_iterations,
                double lam, double mu, double tol):
cdef double[::1] s = np.zeros(p)
cdef double[:, ::1] A = np.linalg.inv(Sigma) / 2
cdef double[::1] temp = np.zeros(p)
cdef double[::1] A_entry = np.zeros(p)
cdef double q, r, c, z, delta, kappa, alpha = 1, beta = 0
cdef char* up = 'U'
cdef int inc = 1

for i in range(max_iterations):
    for j in range(p):
        dsymv(up, &p, &alpha, &A[0, 0], &p, &Sigma[j, 0],
            &inc, &beta, &temp[0], &inc)
        q = (ddot(&p, &temp[0], &inc, &Sigma[j, 0], &inc)
            - 2 * temp[j] * Sigma[j, j] + A[j, j] * Sigma[j, j] * Sigma[j, j])
        r = (temp[j] - Sigma[j, j] * A[j, j]) ** 2 / A[j, j]
        c = 4 * (q - r)
        z = 2 * Sigma[j, j] - lam - c
        if z < 0:
            z = 0
        delta = s[j] - z
        kappa = -delta / (1 + delta * A[j, j])
        for k in range(p):
            if k > j:
                A_entry[k] = A[k, j]
            else:
                A_entry[k] = A[j, k]
        dsyr(up, &p, &kappa, &A_entry[0], &inc, &A[0, 0], &p)
        s[j] = z
    lam = mu * lam
return s
```