Contents

1 Introduction

Multi-view learning is a method which consists in combining information of different sets of data which describe the same objects [1, 2]. In this type of data, different features are partitioned into groups according to some common characteristics and each group of features is referred to as a *view*. This occurs naturally, for example, in biomedical research, where different measurements are often employed. In other words, instead of receiving one single table of data, we are given several tables, each containing different features. Traditional machine learning algorithms combine all views together into one single dataframe, suitable for the conventional learning setting. This naive approach can however lead to overfitting, resulting in losing meaningful statistical information of each view, and it can be very slow. For this reason, multi-view learning has been introduced, with the goal of finding the optimal way of learning from different views.

The relevance of this framework is proportional to the increasing frequency of multi-view data. Suppose for example we are interested in training an algorithm to predict the occurring of a certain disease or to diagnose some medical condition through computer vision. The algorithm would have access, for each patient, to data coming from different scans (MRI, CT, X Rays, Ultrasounds...), different tests results (biopsy, blood test...) and some generic data (personal information, lifestyle...). These kinds of datasets would naturally form different views, with several statistical properties within each view. It is clear that understanding how to combine these datasets together plays a fundamental role in shaping the accuracy of the final prediction. Furthermore, multi-view learning yields extremely useful results also in the matter of feature selection, which is a fundamental problem, as collecting large amounts of data can be very expensive.

The aim of this analysis will be to first introduce an overview of the techniques which are nowadays employed in multi-view learning [1, 2], then I will focus on one example of multi-view learning method, called Stacked Penalized Logistic Regression [3], which has been proven to be very accurate in selecting views containing information and discarding noisy views. Then, I will run a simulation on Python and show, in practice, the results of the mentioned model. Finally I will present several applications of this approach and of similar techniques.

2 Multi-view Learning Overview

2.1 General Framework

As mentioned above, the idea behind multi-view learning is to find the optimal way to combine information coming from different sets of data, as creating one single view would cause a loss of information or statistical properties. This often happens for data coming from different measurements, or for image data where we might have heterogeneous features (such as color and texture).

Multi-view learning methods can be split into three major categories, **co-training** style algorithms, **multiple kernel learning** (MKL) and **subspace learning**. The main idea behind the first group is to maximize the agreement on distinct views, with applications mainly in the field of semi-supervised learning [4].

MKL, on the other hand, was initially designed to address the challenge of choosing the best kernel function for kernel-based methods like Support Vector Machines[1]. However, this approach perfectly fits in the framework of multi-view learning by letting different kernels represent different views of the data. Then, choosing the optimal combination of the kernels can be thought of as finding the most efficient way of combining distinct views.

Finally, subspace learning-based approaches assume the existence of a lower-dimensional subspace generating the data. The goal is to infer this latent subspace, shared by multiple views, overcoming the *curse of dimensionality*.

[1] See Section 2.4.1

2.2 Principles of Multi-view Learning

The main characteristic of multi-view learning is the availability of several views for the same input data, yielding an abundance of information, which is in principle useful to obtain better predictions. However, it is of fundamental importance to appropriately combine these views, by fully considering their relationship. Otherwise, the abundance of data may even lead to worse performances. In particular, there are two significant principles which ensure the success of multi-view learning approaches: *consensus* and *complementary* principles.

2.2.1 Consensus Principle

Consensus principle mainly consists in maximizing the agreement on distinct views. Consider a semi-supervised learning framework, where a label y_i is associated to some data points x_i, which will be called labeled data, while the majority of the dataset is unlabeled. For simplicity, restrict to binary labels, i.e. $y_i = \pm 1$. Suppose that the dataset consists of two single views X^1, X^2: a sample (x_i, y_i) will be thus seen as (x_i^1, x_i^2, y_i). Let f^1 and f^2 be two hypotheses, i.e. two specific models or classifiers, one for the first view and one for the second view. Let $P_{err}(f^1), P_{err}(f^2)$ be, respectively, the error rates of hypothesis f^1 and f^2, where the error rate is the probability of mis-classifying a sample.

It has been proven by Dasgupta et al. [5] that, under some mild assumptions, the probability that two independent hypotheses disagree sets an upper bound for the error rate of either hypothesis, i.e.

$$P(f^1 \neq f^2) \geq \max\{P_{err}(f^1), P_{err}(f^2)\} \tag{2.1}$$

This can be applied to the multi-view learning framework as follows: in co-training style algorithms, training aims to both maximize the mutual agreement on different views of the unlabeled data and minimize the error on labeled examples, achieving an accurate classifier on each view.

Consensus principle can be formulated for co-regularization algorithms, which can be seen as a regularized version of co-training algorithms, as follows[2]:

$$\min \sum_{i \in U} [f^1(x_i) - f^2(x_i)]^2 + \sum_{i \in L} V(y_i, f(x_i)), \tag{2.2}$$

where $V(\cdot, \cdot)$ is a given loss function, U is the set of unlabeled samples and L is the set of labeled samples. The first term enforces the agreement on two different views on unlabeled examples, while the second term evaluates the empirical loss on the labeled examples with respect to the predefined loss function V. In addition, one could add a term regarding the complexity of the hypotheses. The result of this optimization will yield two optimal hypotheses.

Finally, in Kernel Canonical Correlation Analysis (KCCA), which we will see as an example of MKL method, the consensus principle can be applied to obtain a supervised learning algorithm called SVM-2K [6], based on the Support Vector Machine (SVM) architecture, which will be further discussed later on (see Section 2.4.1). The idea behind SVM-2K is to first project the features to a one-dimensional space (using a SVM) and then force the constraint of consensus of two views on this one-dimensional space.

2.2.2 Complementary Principle

The complementary principle states that in the multi-view framework each view might contain knowledge that other views do not have. Thus, multiple views should be employed together to accurately describe the data. The idea behind this principle is to employ the complementary information held by different views to improve performance and accuracy of the model.

Wang and Zhou [7] used the complementary principle to show how co-training style algorithms perform better when there are no redundant views. Further applications of this principle can be seen in multiple kernel learning and subspace learning as well.

[2]Note that we are still considering the semi-supervised learning framework defined above.

2.3 Co-training and Co-regularization Methods

Co-training style algorithms, first proposed by Blum and Mitchell [4], are often used in *semi-supervised learning* [8]. Recall that in this context the labeled data points serve as *supervision* for the algorithm.

In co-training, the algorithms usually train different but correlated models on each view and their outputs are forced to be similar on some validation points, by the *consensus principle*. It can be therefore seen as a late combination of multiple views. These models, which are trained separately on different views, are known as **base-learners**, while the final model, obtained by combining these base-learners together, is called **meta learner**. Each iteration of the procedure aims at maximizing the consistency of the predictions on the labeled data set, while minimizing the disagreement on the unlabeled data: this way, classifiers learn from each other and reach the optimal solution. In particular, in each iteration, the learner on one view labels unlabeled data which are then added to the training pool of the other learner, so that underlying information can be exchanged.

Unlike co-training, co-regularization algorithms directly involve an objective function such as in (2.2), which specifically measures and enforces the agreement between the classifiers trained on different views. The regularization term penalizes the models if their outputs for the same instances are inconsistent. Moreover, in co-regularization, different views are used simultaneously in a single integrated learning process. As seen before, the optimization considers both the prediction accuracy on labeled data and the consistency between views on unlabeled data.

While co-training leverages the independent capabilities of each view through a semi-supervised learning cycle, co-regularization focuses on synthesizing the views by enforcing a mathematical constraint of agreement during training. These methods can be extended to unsupervised learning setting, i.e. when no validation set is provided. In this case, the classifiers are trained on each view and the combination of views is validated on the same training set.

In order for co-training style algorithms to work, the following assumptions have to be satisfied:

a) **sufficiency** - each view is sufficient for classification on its own;

b) **compatibility** - target functions in different views predict the same labels for co-occurring features with high probability;

c) **conditional independence** - views are conditionally independent given the class label[3];

From co-training, several different approaches have been developed, such as co-EM, co-regularization, co-regression, co-clustering, multi-learner algorithms... It is, however, beyond the scope of this analysis to go in depth of each one of the mentioned methods.

2.4 Multiple Kernel Learning

2.4.1 Remarks on Support Vector Machines

Support Vector Machines (SVMs) are an important architecture which is tightly connected to the framework of Multiple Kernel learning, as briefly mentioned in Section 2.2.1.

Consider the supervised learning setting where data points x, contained in some d-dimensional space X, are associated with a label y, which can be either $+1$ or -1. When data points are linearly separable, the goal of SVMs is to find the hyperplane which correctly separates the data points in two classes with the largest margin. In other words, the idea is to maximize the distance between this hyperplane, or classification boundary, and the *support vectors*, which are the data points closest to this boundary. The parameters of the optimization problem are therefore the slope of the separating plane w and the intercept b.

[3]This assumption is often too strong to be satisfied in practice; for this reason, several alternatives have been considered [9].

SVMs can be expanded to non-linearly separable sets of data by simply mapping the data to a higher dimensional dataset. In other words, for any input space X, there exists a mapping $\varphi : X \to \mathcal{H}$, where $\mathcal{H}$ is a D-dimensional space, $D > d$, such that the projected data points $\varphi(x)$ are linearly separable in $\mathcal{H}$. In this case, the optimization problem would require to compute a dot product in the D-dimensional space $\mathcal{H}$, in the form $\langle \varphi(x^i), \varphi(x^j) \rangle$, where x^i, x^j are different data points in X.
This operation can be computationally expensive. For this reason, a particular type of functions, called **kernels**, is employed. These functions have the following fundamental property:

$$k(x^i, x^j) = \langle \varphi(x^i), \varphi(x^j) \rangle_{\mathcal{H}} \tag{2.3}$$

Equation (2.3) is known as **kernel trick**, which is extremely useful as it can capture complex and nonlinear patterns within the data without explicitly computing the mappings φ, saving a lot of computational time and memory.

2.4.2 From SVMs to Multiple Kernel Learning

The kernel approach is applied to a vast range of different problems. In particular, in the context of multiple kernel learning, the idea is to let different kernels represent different views [10]: the goal is to find the optimal method of combining these kernels under the complementary principle, as a way of integrating multiple sources of information and improving learning performance.

Among the various kernel types used in machine learning, literature suggests that the linear, polynomial, and Gaussian kernels stand out as the most frequently employed. The methods of combinations of kernels can be grouped into two categories:

i) **Linear combination methods** - direct summation kernel and weighted summation kernel [11];

ii) **Nonlinear combination methods** - exponentiation and power [12], kernel regression and polynomial combination [13];

An example of MKL algorithms is the Multiple Additive Regression Kernels (MARK) pro-

posed by N. Duffy and D. Helmbold [14], which is inspired by ensemble and boosting methods. Other approaches involve learning the kernel matrix from data with semi-definite programming techniques [11] or with a semi-infinite linear program [15][16]. Again, several other approaches have been employed, but it is beyond the goal of this analysis to mention them all.

2.5 Subspace Learning

Subspace learning techniques focus on finding a latent subspace shared by multiple views, based on the assumption that a lower-dimensional representation underlies the generation of the data. It is an instance of unsupervised learning. In single-view learning, the most relevant example of subspace learning is Principal Component Analysis (PCA): PCA is a dimensionality reduction technique, which finds a way of projecting a data points to a lower dimensional space, without losing important information.

This approach can be generalized to multiple views as **Canonical Correlation Analysis** (CCA). CCA finds an optimal projection on each view by maximizing correlation between different views in the subspace. In the case of two views, $X \in \mathbb{R}^{D_1 \times N}$ and $Y \in \mathbb{R}^{D_2 \times N}$, CCA computes two projection vectors $w_x \in \mathbb{R}^{D_1}$ and $w_y \in \mathbb{R}^{D_2}$ so that the following correlation coefficient:

$$\rho = \frac{w_x^T X Y^T w_y}{\sqrt{(w_x^T X X^T w_x)(w_y Y Y^T w_y)}} \tag{2.4}$$

is maximized.

The main limitation of CCA, however, is that the subspace is thought to be linear, thus it can be seen as a linear feature extraction algorithm. In order to deal with the non-linearities of the data, kernel canonical correlation analysis (KCCA) has been proposed. The idea is to first project the data in higher dimensions, then apply CCA on the higher dimensional representation.

As mentioned in Section 2.2.1, KCCA can be combined with classification algorithms such as Support Vector Machine: this method is called SVM-2K [6], and it can be seen as

the optimization of two distinct Support Vector Machines, one in each of the two feature spaces.

In contrast with CCA, which ignores label information, a generalization of **Fisher's discriminant analysis** has been proposed by Diethe et al. [17], to find informative projections for multi-view data in a supervised setting.
In single-view learning, the idea behind Fisher Linear Discriminant (FLD) is to optimize a function that will yield significant separation between the projected class means, while simultaneously minimizing the variance within each class, which effectively reduces class overlap. In other words, FLD selects a projection that maximizes the class separation [18]. This approach can be extended to several views, resulting in multi-view Fisher discriminant analysis.

Other examples of subspace learning are multi-view embedding, multi-view metric learning and latent space models.

3 StaPLR for View Selection

3.1 Multi-view Stacking

Multi-view stacking (MVS) [19], [3] is a multi-view learning technique that involves training separately individual models, called **base-learners**, on different representations of the data, i.e. views. The predictions of these base-learners, each corresponding to a different view, are then used as input to a secondary model, which is called **meta-learner** and it learns how to optimally combine them. When the meta-learner is designed to produce sparse models, MVS can also be used for feature or view selection, which will be further explained in Chapter 4.

From now on, we will use the following notation: let $X^{(1)}, ..., X^{(V)}$ denote our multi-view data set, with $X^{(v)}$ corresponding to the $n \times m_v$ matrix of features in view v. Let $y = (y_1, ..., y_n)$ be the vector of labels. Clearly, n represents the sample size. In this context, we define a **learner** A to be a function which maps a labeled data set to a learned function $\hat{f}$ that associates a label to each input vector. For each view $X^{(v)}$, with $v = 1, ..., V$, several base-learners $A_{v,b}$ can be applied to all observations of that view, with B_v denoting the total number of base-learners in that view, i.e. $b = 1, ..., B_v$. In principle there can be multiple meta-learners as well, however in this case we consider only one, A_{meta}.

Each base-learner $A_{v,b}$ is applied to the n observations of view v. As a result we obtain a set of learned functions $\hat{f}_{v,1}, ..., \hat{f}_{v,B_v}$, one for each base-learner. We now need to compute cross-validated predictions for each base-learner, as they will be used to train the meta-

learner. For this reason, the data is partitioned into K groups denoted $S_1, ..., S_K$; for each group S_k, the base-learner $A_{v,b}$ is applied to the observations which do not belong to the group, denoted $X^{(v)}_{i \notin S_k}, y_{i \notin S_k}$, and the learned function $\hat{f}_{v,b,k}$ is then applied to the samples in S_k to obtain the cross-validated predictions. Here, the subscript $k = 1, ..., K$ denotes the index of the group that was excluded from training, on which the learned function is applied to obtain the predictions. For each view and for each base-learner $A_{v,b}$, we obtain a vector $z^{(v,b)} \in \mathbb{R}^n$ of cross-validated predictions. These vectors are collected in a $n \times B$ matrix denoted Z, with $B = \sum_v B_v$.

The matrix of cross-validated predictions Z is used as input for the meta-learner, to obtain the final model $\hat{f}_{meta}$. The final predictions are then $\hat{f}_{meta}(\hat{f}_{1,1}(X^{(1)}), ..., \hat{f}_{V,B_V}(X^{(V)}))$. The steps are highlighted also in the pseudo-code provided below.

Algorithm 1 Multi-View Stacking (2 levels)

Data: Views $X^{(1)}, \ldots, X^{(V)}$ and outcomes $y = (y_1, \ldots, y_n)$.
for $v = 1$ to V **do**
    for $b = 1$ to B_v **do**
        $f_{v,b} = A_{v,b}(X^{(v)}, y)$
    end for
end for
for $v = 1$ to V **do**
    for $b = 1$ to B_v **do**
        for $k = 1$ to K **do**
            $f_{v,b,k} = A_{v,b}(X^{(v)}_{i \in S_k}, y_{i \in S_k})$
            $z_i^{(v,b)} = f_{v,b,k}(X_i^{(v)})$ for all $i \in S_k$
        end for
    end for
end for
$Z = (z^{(1,1)}, z^{(1,2)}, \ldots, z^{(1,B_1)}, z^{(2,1)}, \ldots, z^{(V,B_V)})$
$\hat{f}_{\text{meta}} = A_{\text{meta}}(Z, y)$
$\hat{y} = \hat{f}_{\text{meta}}(f_{1,1}(X^{(1)}), \ldots, f_{V,B_V}(X^{(V)}))$

In the setting of multi-view-stacking, it has been proven that constraining the coefficients of the meta-learner to be non-negative yields an improvement both of the accuracy of the model and in the view selection performance.

Several models can be used as base-learners or meta-learners [19]. In this case, we restrict

our attention to the case where **penalized logistic regression** is used for both the base-learners and the meta-learner. The next section will provide further details on this topic.

3.2 Remarks on Logistic Regression

Logistic regression is a model used in classification problems [20], where the goal is to predict an output label Y based on some given input data $\mathbf{x} = (x_1, ..., x_m)$ by inferring the conditional probability

$$P(Y = y|X = \mathbf{x})$$

In particular, consider the case of binary outputs, i.e. we will restrict to the context of binomial regression with $Y = 0, 1$. Then, in the **logistic regression model**,

$$P(Y = 1|X = \mathbf{x}) = \frac{1}{1 + \exp(-\beta_0 - \sum_{j=1}^{m} \beta_j x_j)}, \tag{3.1}$$

where the coefficients $\beta_1, ..., \beta_m$ express the relative importance of the different x_j's.
In simple logistic regression, these coefficients are estimated by maximizing the likelihood, i.e. the probability that every sample in the training set is correctly classified. In practice, the log-likelihood is maximized instead, as the logarithm is a monotonic transformation and therefore it does not change the maximizers.

The model works under the following assumptions:

i) independent observations;

ii) binary dependent variables[4];

iii) linear relationship between independent variables and log odds of the dependent variable;

iv) no outliers;

v) large sample size;

[4]In practice, however, we often consider problems with multiple classes.

3.2.1 Penalized Logistic Regression

The idea behind penalized models is to introduce a regularization term that enforces the exclusion of some coefficients in the final model. In practice, this is useful because, when a large number of features is involved, one might want to have some kind of *selection*, i.e. to have some features excluded from the final model, in order to actually grasp which ones have the most relevance on the final outcome, and to reduce the computational cost of the final model.

Instead of maximizing the likelihood alone, penalized methods introduce some regularization term to the optimization problem, which shrinks the coefficients of the less relevant variables to zero.

In **penalized logistic regression**, coefficients are thus estimated as follows:

$$\hat{\beta}_0, \hat{\beta} = \underset{\beta_0,\beta}{\arg\max} \left\{ \sum_{i=1}^{n} \left[y_i(\beta_0 + \beta^T x_i) - \log(1 + \exp(\beta_0 + \beta^T x_i) \right] - P(\beta, \lambda) \right\},$$

where $P(\beta, \lambda)$ is some penalty, depending on the coefficients β and on a hyper-parameter λ, which determines the strength of the regularization.

The most commonly used penalty terms include:

- Ridge penalty;
- Lasso penalty;
- Elastic net;

Ridge penalty [21], also known as L2 regularization, is defined as

$$P_{ridge}(\lambda, \beta) = \lambda \parallel \beta \parallel_2^2 \quad \text{with} \quad \parallel \beta \parallel_2^2 = \sum_i \beta_i^2 \tag{3.2}$$

By adding this penalty term, Ridge regression shrinks the coefficients of less relevant variables towards zero, which in turn helps to stabilize model, improve accuracy and reduce

overfitting. With this type of regularization, variables with minor contribution have their coefficients significantly close to zero, but they are all still included in the model.

Lasso penalty, also known as L1 regularization, is, on the other hand, defined as

$$P_{lasso}(\beta, \lambda) = \lambda \parallel \beta \parallel_1 \quad \text{with} \quad \parallel \beta \parallel_1 = \sum_i |\beta_i| \tag{3.3}$$

The main benefit of using this penalization is that it encourages some coefficients to be exactly zero, thus it performs feature selection by eliminating unimportant predictors from the model.

Finally, **elastic net** regularization is the combination of Ridge and Lasso penalties. Some coefficients are therefore shrunk towards zero, while other are set to be exactly zero.

3.3 Stacked Penalized Logistic Regression

Stacked Penalized Logistic Regression (StaPLR) [3] is an instance of multi-view stacking, where penalized logistic regression is used both in the base-learners and in the meta-learner.

In StaPLR, the parameters for a base-learner b trained on view v are estimated as

$$\hat{\beta}_0^{(v)}, \hat{\beta}^{(v)} = \underset{\beta_0, \beta}{\arg\max} \left\{ \sum_{i=1}^{n} \left[y_i(\beta_0 + \beta^T x_i^{(v)}) - \log(1 + \exp(\beta_0 + \beta^T x_i^{(v)})) \right] - P_b(\beta, \lambda) \right\}$$

for some penalty function $P_b(\beta, \lambda)$, where λ is the tuning parameter. For simplicity, we drop here the bold notation used for vectors. The parameters of the meta-learner are estimated as:

$$\hat{\beta}_0^{meta}, \hat{\beta}^{meta} = \underset{\beta_0, \beta}{\arg\max} \left\{ \sum_{i=1}^{n} \left[y_i(\beta_0 + \beta^T z_i^{(v)}) - \log(1 + \exp(\beta_0 + \beta^T z_i^{(v)})) \right] - P_{meta}(\beta, \lambda) \right\}$$

In the case of StaPLR, the penalty of the base learner is chosen to be **Ridge regression** [21], while the penalty chosen for the meta-learner is **Lasso**. Since we are interested in

view selection, it is clear why using Lasso penalty on the meta-learner is a reasonable choice.

Parameters in the model are estimated with coordinate descent [22], while the tuning parameters λ, which determine the strength of the regularization, are chosen through cross validation.

Several papers have proposed to constrain the parameters to be non-negative [23], [24], [25]. Although this constraint does not impact much the prediction accuracy of the model, it has been proven to be fundamental in the view selection performance.

4 Synthetic Data Analysis

The goal of this chapter is to present a simple simulation done on Python to assess the performance of StaPLR. I will first mention how to construct synthetic data, then I will explain the functions that I have created and finally discuss the results obtained by StaPLR, comparing them with those obtained by a Group Lasso instance. Note that the entire code is available at this link: `https://github.com/beatricecitterio/multi-view-learning`.

4.1 Setting and Methods

4.1.1 Data Generation

The first step of this analysis is to build the data. To do so, I created a function `generate_data` which takes as argument a list, called `condition`, containing the following parameters:

- `sample_size`: the number of samples we want to generate;
- `view_size`: the number of features in each view;
- `intra_view_correlation`: the correlation between features within the same view;
- `inter_view_correlation`: the correlation across different views;

By default, the number of views is set to 30 as in [3]. In principle, however, this can be easily changed.

To create the dataset, the function draws samples from a multivariate normal distribution, ensuring that the generated features adhere to some specific correlation structure determined by a correlation matrix, which is the output of another function, `generate_correlation_matrix`. This function constructs a block correlation matrix such that within each view, the correlation between features is controlled by the `intra_view_correlation` parameter, while correlation between features belonging to different views is governed by the `inter_view_correlation` parameter.

Since generating the data can be computationally expensive when the number of features per view is large, I decided to use the following combinations of parameters as different experimental conditions:

- `sample_size` can be equal to 50, 100 or 200;
- `view_size` can be 50 or 100, so that we can see how the performance changes when the number of features per view is either smaller than the size of the sample or larger; more features could be used, but this slows down the training considerably;
- three couples of correlation values are employed, namely $[0.1, 0]$, $[0.4, 0]$ and $[0.4, 0.2]$, as in [3];

After generating features according to the correlation parameters, *signal* is injected into the data by assigning non-zero coefficients to some features. The signal is distributed across the views so that in views 1-5 all features have probability 1 of having a true relation with the response. For views 6-10 the signal probability of each feature is 50%, while it is zero for the remaining views (i.e. 11 to 30).

Finally, a binary outcome variable is generated using a logistic function applied to a linear combination of the features and their associated regression weights. This simulates a binary classification problem where the goal is to predict y based on the multi-view features X, as in the logistic regression model (Section 3.2).

4.1.2 Model

After generating the dataset, two classes are created, `BaseLearner` and `MetaLearner`, to implement the Stacked Penalized Logistic Regression introduced in Chapter 3. As mentioned before, this technique leverages the predictions of multiple base models trained on each view separately as input for a second-level meta-model.

The `BaseLearner` class is designed to initialize and manage a logistic regression model. The constructor takes as parameters the regularization type, which in this case is set to L2, the inverse regularization strength, here $C = 0.1$, the solver (`saga`) and the maximal number of iterations, which in this case we set to be 1000. Then, a `LogisticRegression` model, imported from the library `scikit-learn`, with the parameters defined above, is initialized. The methods `fit` and `predict` allow to complete the training and evaluation of the base learner.

The `MetaLearner` integrates multiple `BaseLearner` instances and uses their predictions as features for a final logistic regression model. In this case, the penalty chosen is L1, for the reasons mentioned in Section 3.3. Beyond the usual `fit` and `predict` methods, I defined an additional method, `get_inclusion_probabilities`, which determines the significance of each base learner's predictions in the final decision of the meta-model by checking which coefficients in the meta-model's logistic regression are significantly different from zero (threshold $= 1e - 5$). Therefore, this function can be used to understand the relevance of each view in the final prediction.

4.1.3 Inclusion Probabilities

For each view, the inclusion probability is computed as the number of times that a view is included in the final model, i.e. it has a coefficient larger than zero, or larger than a given threshold, out of the total number of times that we run the simulation. I created a function `plot_inclusion_prob` which computes these probabilities and then constructs box plots to visualize how the inclusion probabilities change with the type of view (i.e. from views with signal probability 1 to views with signal probability 0). We can plot this

for different conditions and see how the results change.

4.1.4 Performance

The performance of the model is computed both with the accuracy and with the **AUC score**. The first returns the fraction of correctly classified samples, while the latter is derived from the ROC curve, which plots the False Positive Rate (**FPR**) against the True Positive Rate (**TPR**) at various threshold settings.

FPR is defined as the proportion of negative instances which are incorrectly classified as positive. Formally,

$$FPR = \frac{FP}{TN + FP}, \tag{4.1}$$

where FP is the number of false positives and TN is the number of true negatives.

TPR, on the other hand, is computed as the proportion of positive labels which are correctly classified. This is also known as *sensitivity* or *recall*. Formally,

$$TPR = \frac{TP}{TP + FN} \tag{4.2}$$

where TP is the number of true positives and FN is the number of false negatives.

The AUC score is simply the area under the ROC curve. It is used as an accuracy metric as it quantifies the classifier's ability to distinguish between the positive and negative classes. AUC score of 1 would mean that the model perfectly distinguishes between positive and negative classes, while if AUC = 0.5 then model performs no better than random guessing.

In this analysis, I computed 50 values of accuracy and AUC score for each given experimental condition. The results are then shown with a series of box plots.

4.1.5 Group Lasso

Group Lasso is an extension of the Lasso penalization (see Section 3.2.1) which allows to perform feature selection on some groups of features. Here, I created a function `fit_group_lasso` which first concatenates all the views together into one single data frame, then divides the features into groups, where group i corresponds to the features belonging to view i. Then, the `GroupLasso` model is initialized, with `group_reg` = 0.05, `l1_reg` = 0.1 and `scale_reg` = `"inverse_group_size"`. Here, `group_reg` and `l1_reg` represent, respectively, the regularisation coefficient for the group sparsity penalty, and for the coefficient sparsity penalty. The first encourages entire groups of features to become zero, performing feature selection at group level. The second encourages individual coefficients within a group to become zero, leading to sparsity within a group. Finally, `scale_reg` determines how to scale the group-wise regularisation coefficients.

Note that the choice of parameters, in particular `group_reg` and `l1_reg`, deeply influences the behavior of the model. Therefore, it could be interesting and useful to experiment with different settings of parameters.

4.2 Results

4.2.1 View Selection

View selection probabilities are computed as the proportion of times that a view was included in the model: in practice, for each condition, the experiment was carried out 50 times.

In Figure 4.1, we see the performance of the model for different experimental conditions. Note that, in general, what seems to shape the results the most is the relationship between the first two parameters, i.e. `sample_size` and `view_size`. As a matter of fact, within rows, i.e. when these two parameters remain unchanged, we can see a similar pattern.

Another trend which emerges in particular when the number of features per view is lower

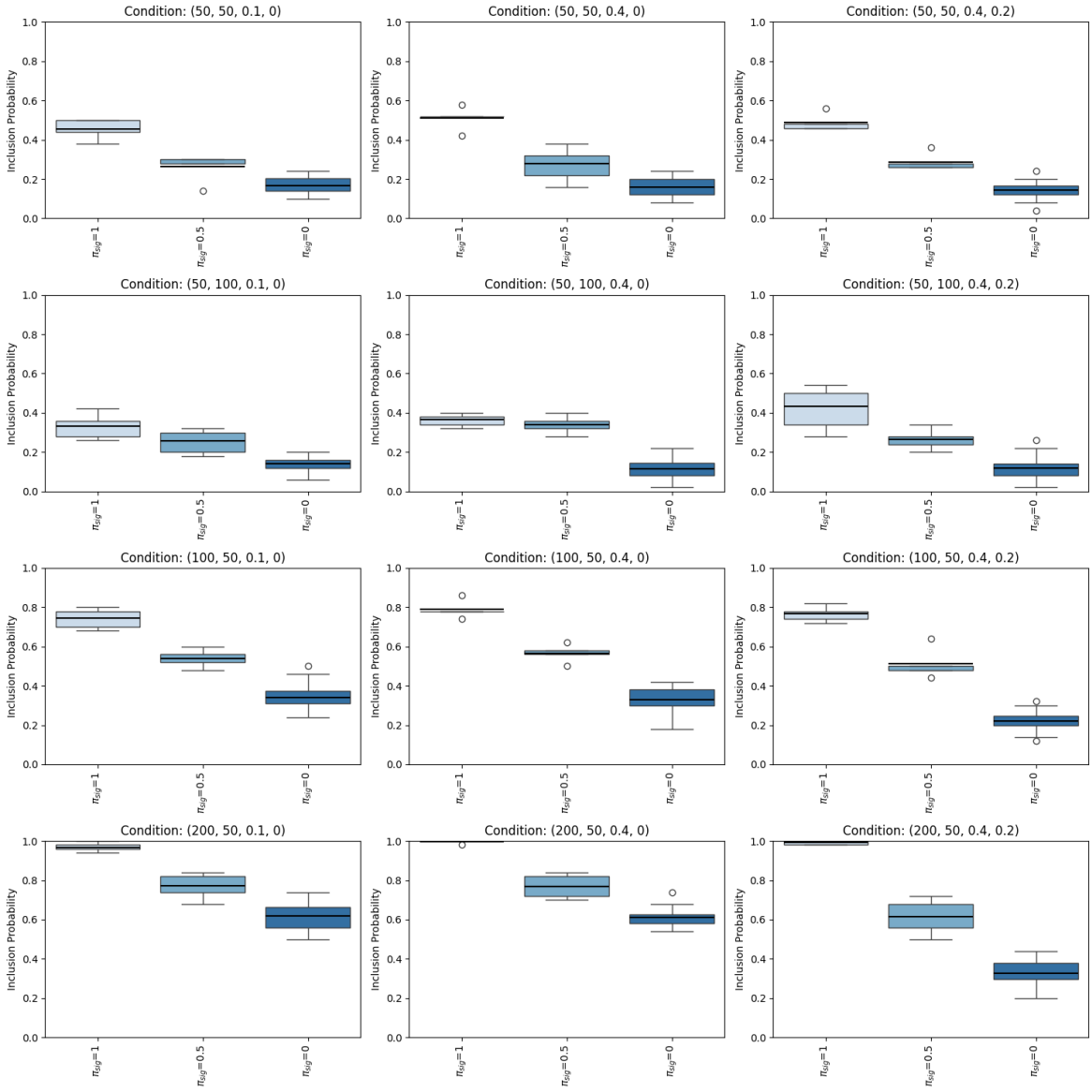

Figure 4.1: Inclusion Probabilities by View Type (StaPLR)

than the number of samples is that the model performs the best when correlation is [0.4, 0.2]. As a matter of fact, for these values, not only the probability of including signaling views is high, but also the probability of including non-signaling views decreases, which is our ultimate goal.

It is clear to see that, when the number of samples is lower than the number of features in each view, the performance of the model is very conservative, in the sense that it tends to include only a few views at each iteration. This can be seen in particular in the second row of Figure 4.1, where inclusion probabilities tend to be low for each view type.

If the number of samples remains the same but we decrease the number of features in each view, i.e. in first row of Figure 4.1, we see that the inclusion probabilities are higher for relevant views and lower for non-relevant views. In other words, the boxes in the plot are further away from each other, which shows that the view selection performance of the model is getting better. For this reason, we then decided to plot the probabilities only by increasing the number of samples and keeping the number of features within each view constant.

Comparing the results of 50 samples (first row), 100 samples (third row) and 200 samples (fourth row), we see that as the number of samples increases, overall more views are included in the model. This is good because relevant views are almost always included (their inclusion probability is close to 1 in the last row of the figure). The drawback is, however, that the model becomes less selective, so it tends to include more views in general, even when they are not relevant. In particular, with 200 samples and `inter_view_correlation` = 0, the probability of including noisy views is around 60%, which is clearly undesirable.

The best performance is obtained when `intra_view_correlation` is 0.4 and `inter_view_correlation` is 0.2: in the corresponding boxplots we see that signaling views are always included, while non-signaling ones are included less than 40% percent of the times. Ideally, one would like to obtain inclusion probability equal to 1 for views that have high

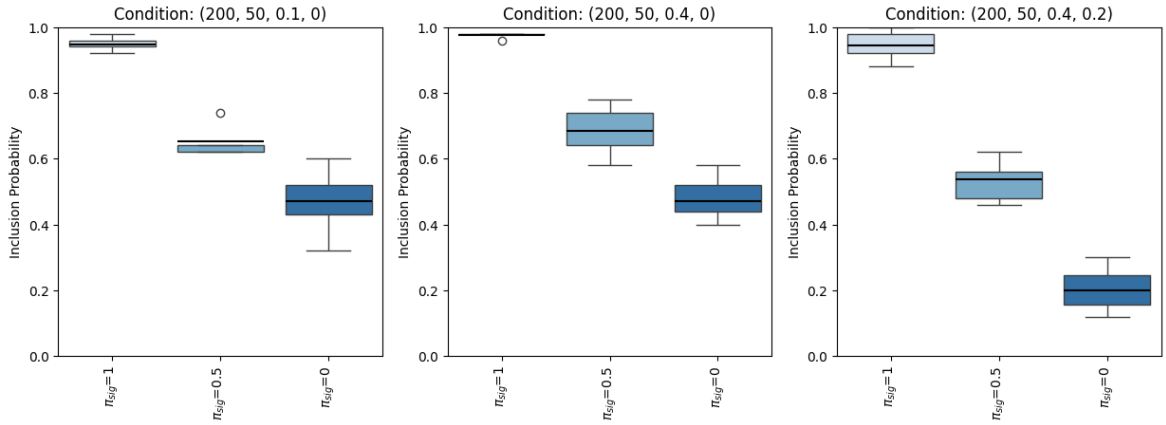

Figure 4.2: Inclusion Probabilities with threshold = 0.4

signal probability and inclusion probability 0 for those with zero signal probability.

Overall, the model seems to *learn* during training the correct importance for each class of views. The only drawback is in the rate of *false positives*, i.e. number of views with no relevance on the output which are actually included in the model. As mentioned before, we would like to include fewer views with signal probability 0, in order to have more efficient view selection.

To achieve this, we could increase the threshold in the function `get_inclusion_probabilities`. I experimented with several values of this parameters, and changes started to appear only when the threshold was above 0.4: although it might seem high in principle, heuristics were in favour of this choice. I tried this approach only for `sample_size` = 200 and `view_size` = 50, as this condition was the one yielding better results in the previous setting.

Figure 4.2 shows how increasing the threshold actually improves the performance. As a matter of fact, views with signal probability of 1 are included with probability close to 1, while the probability of inclusion of non-signaling views decreases. The best result is obtained again when `intra_view_correlation` is 0.4 and `inter_view_correlation` is 0.2. Indeed in this case we obtain the lowest inclusion probability so far for non-signaling views, which is around 0.2.

4.2.2 Performance Metrics

In this section I will present the results obtained by the model in terms of accuracy and AUC scores. Both scores are computed, for each condition, across several trials.

Figure 4.3 shows how the two metrics vary with different conditions for StaPLR. It is clear to see that in this case the different correlation within and between views does shape the outcome, differently from what was observed in Section 4.2.1. Indeed, the model yields better results both in accuracy and in AUC score when `intra_view_correlation` = 0.4 and `inter_view_correlation` = 0.2. On the other hand, between the first two columns there doesn't seem to be much difference, suggesting that what actually influences the outcome the most is the `inter_view_correlation`, which changes only in the last column. Increasing the number of features per view seems to have little impact on the performance of the model. On the other hand, for `sample_size` = 200, StaPLR seems to perform the best, which is reasonable, since having access to more sample data should always yield better results.

Figure 4.4 shows the performance of the model compared to the simple Group Lasso instance defined in Section 4.1.5. Again, we see that in this case as well, when `inter_view_correlation` is higher, both models seem to perform better. Moreover, StaPLR seems to outperform Group Lasso when there is no `inter_view_correlation` and small `intra_view_correlation` (i.e. equal to 0.1). In general, however, the two models obtain similar results in terms of accuracy across all conditions. This was somehow expected, since Group Lasso is a well established model. One of the key strengths of StaPLR is that it can match the accuracy of Group Lasso while selecting fewer views and significantly decreasing computational time, as illustrated in Figure 4.5. This reduction in time becomes more pronounced as the number of samples increases.

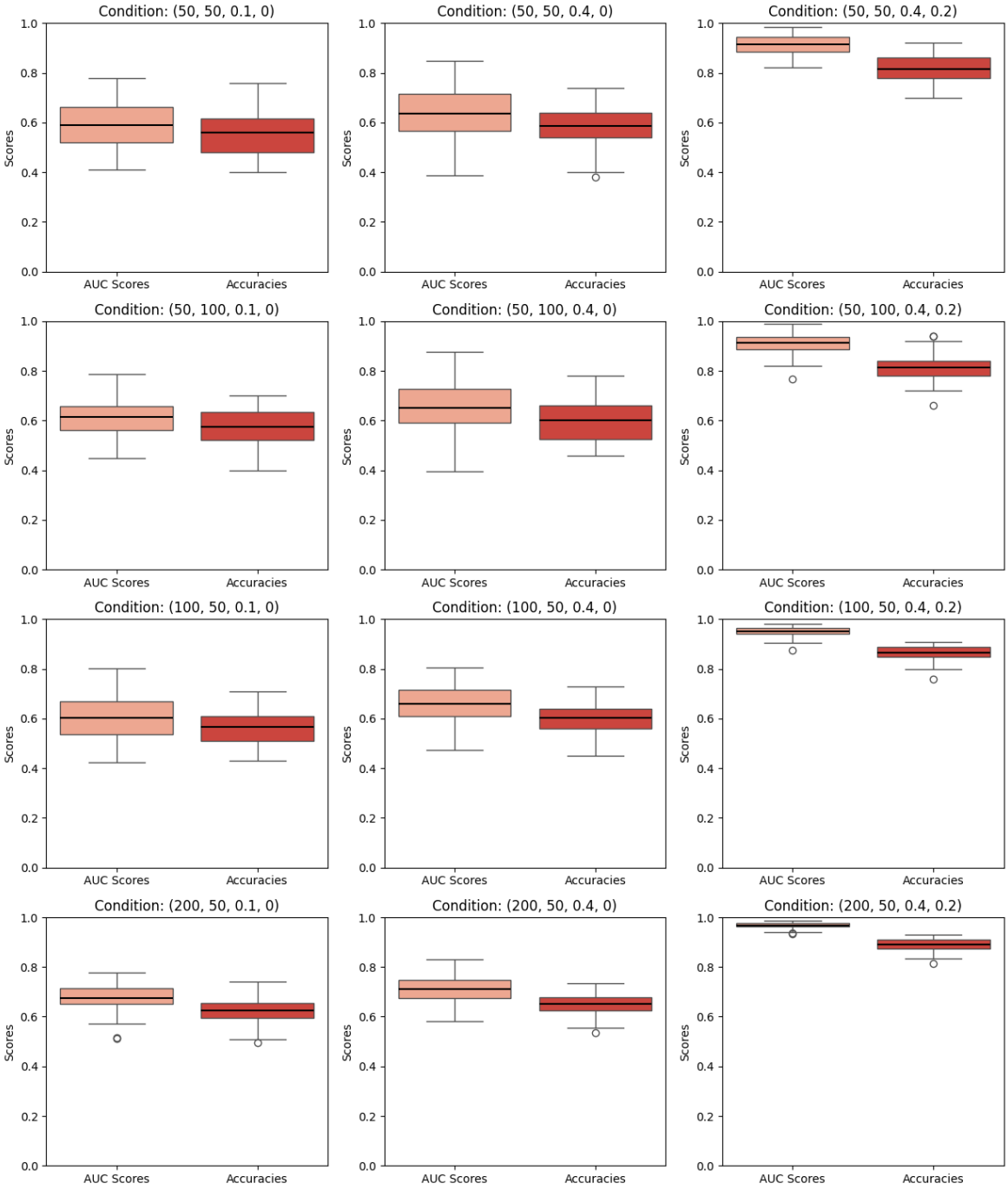

Figure 4.3: Accuracy and AUC score for StaPLR model

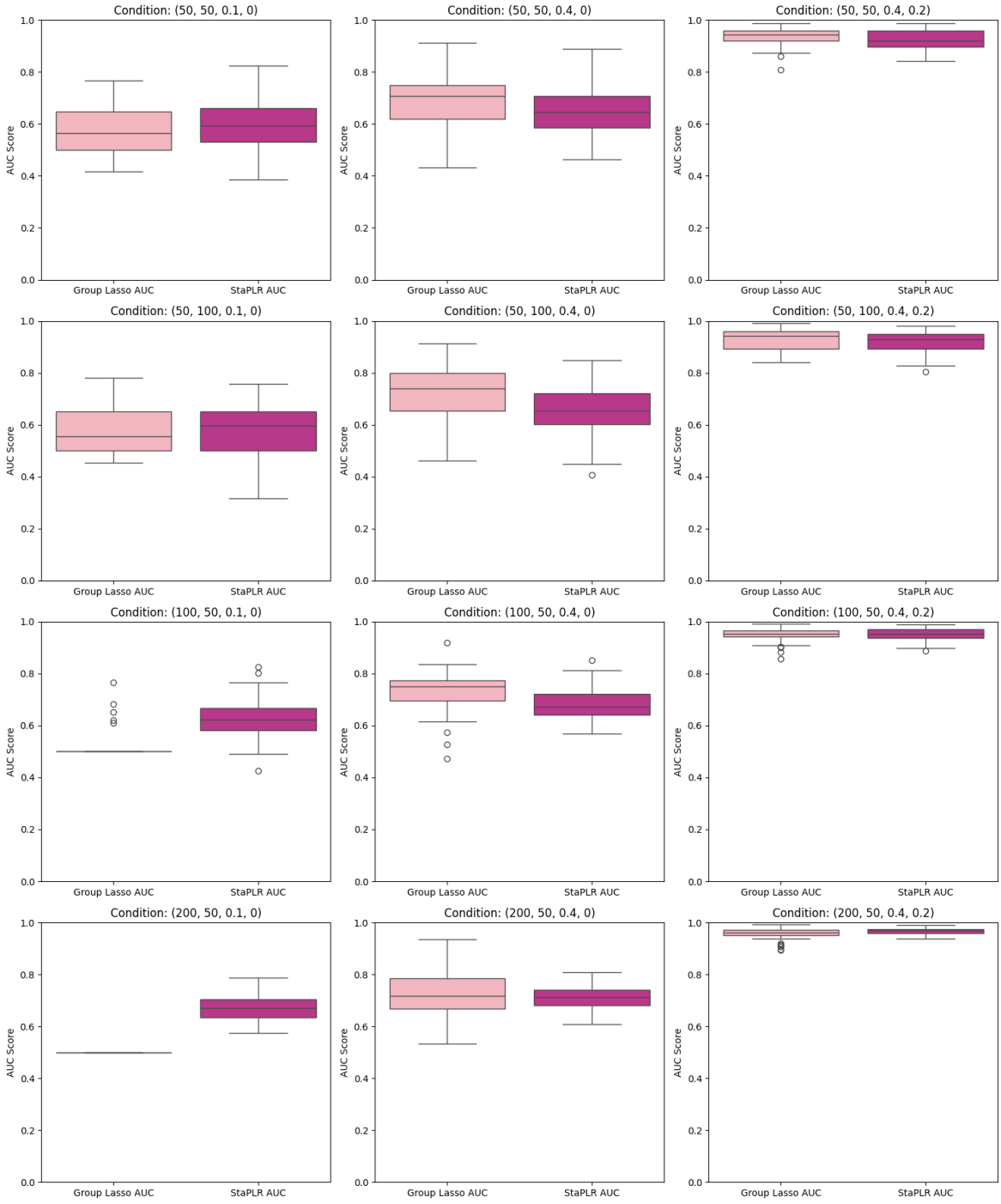

Figure 4.4: AUC scores computed over 50 iterations for Group Lasso and StaPLR

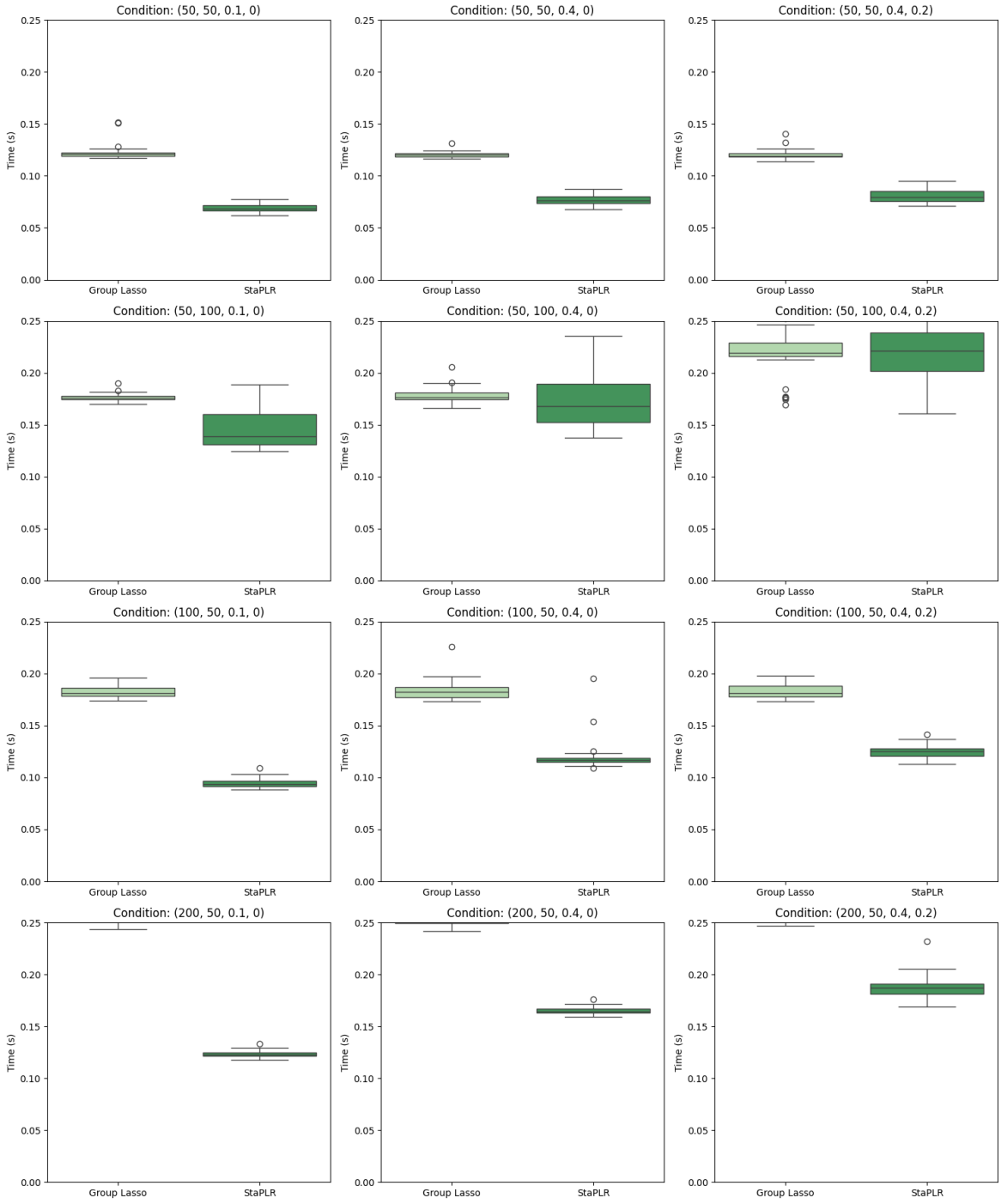

Figure 4.5: Computational time of Group Lasso and StaPLR

4.3 Discussion

Overall, the following observations can be made. Firstly, as one would expect, we see that in general having access to more samples improves the performance. This is reasonable because it means that the model has more data to train on. For this reason, one could increase the number of samples even more to improve the precision of this analysis. Furthermore, this does not impact the speed of the algorithm, which converges very quickly.

From the view selection perspective, having less features per view than samples yields a better performance. This characteristic, on the other hand, does not seem to impact the accuracy of both StaPLR and Group Lasso.

Furthermore, with respect to other methods, such as Group Lasso, StaPLR shows decreased false positive rate in view selection [3], which is of course a desirable property. In particular, it selects less views without losing classification accuracy, reducing the overall computational time.

To improve the performance even more, one could increase the threshold in view selection, to get an even more selective model. Moreover, the fact that increasing the threshold as in Section 4.2.1 does not affect the inclusion probability of signaling views suggests that the model assigns high coefficients to the relevant views, correctly understanding their impact on the output. For this reason, we can say that the performance of StaPLR is satisfactory.

5 Applications and Further Research

The goal of this chapter is to highlight the strong potential and broad range of applications of multi-view learning in general and of StaPLR as one of its representative methods. As mentioned in the Introduction, the main benefits of these procedures are optimizing the combination of data coming from different measurements and performing view selection. These two aspects are crucial in improving the performance of the models and reducing the costs of collecting and dealing with large data sets.

These methods intuitively apply to domains such as biomedical research. However, we will see in this chapter how the field of potential applications is much broader than one would think.

5.1 Applications of StaPLR

5.1.1 Alzheimer's Disease Prediction

A key advantage of StaPLR is that it is currently one of the few multi-view learning methods which can be extended to hierarchical multi-view structures with unlimited levels, while maintaining computational efficiency. Hierarchical multi-view data refers to datasets in which some views can be divided into subviews. Consider for example the setting proposed by van Loon et al. [26] in the context of Alzheimer's disease classification: data is based on three different types of MRI scans (structural MRI, diffusion-weighted MRI and resting state functional MRI). For each scan type, different measures are computed, each with different features: these measures thus represent our usual *views*. However, in this setting, they are in turn nested in different scan types.

As mentioned before, StaPLR can be easily extended to hierarchical multi-view data, allowing to compute measures of feature/view relevance at each level of the hierarchy. The results obtained by StaPLR in [26] are compared with those of a simple logistic elastic net regression: the models agree on which MRI measures are the most important for the classification of Alzheimer's disease. However, StaPLR includes less MRI measures in the final model without losing accuracy, yielding better results in terms of computational costs and view selection.

5.1.2 Gene Expression Data

Gene expression profiling offers a clear example of multi-view data. In this context, genes can be categorized into distinct sets based on various criteria, such as their involvement in signaling pathways or their cytogenetic location [27].

Van Loon et al. [3] have shown how StaPLR can be applied to different gene expression datasets (colitis dataset from Burczynski et al. [28] and breast cancer dataset from Ma et al. [29]) and they compared the results with Group Lasso. For the colitis dataset, both methods have comparable performance in terms of AUC and accuracy, but StaPLR selects fewer views.
On the breast cancer dataset, on the other hand, StaPLR outperforms the Group Lasso even in terms of AUC and accuracy. Moreover, the addition of non-negativity constraints leads to an even more selective model.

5.2 Applications of Multi-view Learning

Multi-view learning is found to be strongly linked to other topics in machine learning, such as active learning, ensemble learning and domain adaptation [1].

Co-training style algorithms and subspace learning algorithms have been applied to several different fields, from image classification to natural language processing, improving performance with respect to old-fashioned approaches. Moreover, with co-training algo-

rithms, the cost of labeling unlabeled data can be eliminated, as mentioned in Section 2.3. Multiple kernel learning, on the other hand, has been mainly applied to object classification, object detection and object recognition.

5.3 Further Relevant Work

Hou et al. [30] proposed a multi-view learning approach to reduce the computational cost of processing multiple camera views in computer vision tasks. The authors introduced a view selection module called MVSelect, which uses reinforcement learning to determine the most informative camera views for a given task. This approach aims to achieve comparable performance to using all available views while significantly reducing the computational burden. The method is evaluated on multi-view classification and detection tasks, showing promising results with only 2 or 3 out of N views. The authors also mention the potential for optimizing camera layouts based on the insights gained from the view selection process.

Cao and Xie [31] proposed a multi-view algorithm to tackle the problem of building graphs for multi-view unsupervised feature selection. As a matter of fact, current methods do not fully utilize information about different levels of neighbors in the data. Their approach by introducing multi-order neighbor information, has been proven to be extremely effective.

Multi-view learning has found numerous applications in Natural Language Processing (NLP). In particular, multimodal semantic representation models [32], [33], which incorporate diverse data sources like text and images, have demonstrated superior performance compared to unimodal models (relying solely on text) in tasks like semantic relatedness and compositional prediction.

These examples represent just a glimpse into the vast landscape of multi-view learning applications, demonstrating its potential to transform fields from computer vision and feature selection to natural language processing, with countless possibilities yet to be explored.

6 Conclusion

In this analysis, I have presented an overview of multi-view learning techniques, highlighting their significance and utility in various domains. The main benefit of multi-view learning lies in its ability to exploit the statistical properties of individual views, which conventional methods often overlook by concatenating all views into a single dataset.

A large portion of this thesis was dedicated to the Stacked Penalized Logistic Regression (StaPLR) model, a specific instance of multi-view learning. The simulation done in Chapter 4 showed that StaPLR performs robustly across various conditions, particularly in view selection, as it minimizes the inclusion of irrelevant views without losing accuracy. These findings agree with recent literature ([3] and [26]). Future improvements of this analysis could focus on enhancing model selectivity, for example by adjusting thresholds and constraints to improve view selection. Furthermore, enhancing the scalability of these models to handle larger datasets and more views would be a crucial step. Finally, investigating the choice of different base-learners could deeply impact the results.

In conclusion, multi-view learning holds significant promise to elevate the performance of machine learning models across a wide range applications. Its ability to integrate heterogeneous data sources, particularly valuable in biomedical research, has the potential to revolutionize fields far beyond. By capturing complementary information from multiple views, these methods can unlock deeper insights, leading to more accurate and robust predictions. The continuous development and refinement of multi-view learning approaches such as StaPLR will undoubtedly drive significant advancements in various domains, leading the way for innovative solutions to complex real-world problems.

Bibliography

[1] Chang Xu, Dacheng Tao, and Chao Xu. "A Survey on Multi-view Learning". In: *CoRR* abs/1304.5634 (2013). URL: http://arxiv.org/abs/1304.5634.

[2] Jing Zhao et al. "Multi-view learning overview: Recent progress and new challenges". In: *Information Fusion* 38 (2017), pp. 43–54. ISSN: 1566-2535. URL: https://www.sciencedirect.com/science/article/pii/S1566253516302032.

[3] Wouter van Loon et al. "Stacked penalized logistic regression for selecting views in multi-view learning". In: *Information Fusion* 61 (2020), pp. 113–123. ISSN: 1566-2535.

[4] Avrim Blum and Tom Mitchell. "Combining labeled and unlabeled data with co-training". In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. COLT' 98. Madison, Wisconsin, USA: Association for Computing Machinery, 1998, pp. 92–100. ISBN: 1581130570. URL: https://doi.org/10.1145/279943.279962.

[5] Sanjoy Dasgupta, Michael Littman, and David McAllester. "PAC Generalization Bounds for Co-training". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper_files/paper/2001/file/4c144c47ecba6f8318128703ca9e2601-Paper.pdf.

[6] Jason Farquhar et al. "Two view learning: SVM-2K, Theory and Practice". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press, 2005.

[7] Wei Wang and Zhi-Hua Zhou. "Analyzing Co-training Style Algorithms". In: *European Conference on Machine Learning*. 2007. URL: https://api.semanticscholar.org/CorpusID:17037938.

[8] Yassine Ouali, Céline Hudelot, and Myriam Tami. *An Overview of Deep Semi-Supervised Learning*. 2020. arXiv: 2006.05278.

[9] Steven Abney. "Bootstrapping". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 360–367. URL: https://aclanthology.org/P02-1046.

[10] Mehmet Gönen and Ethem Alpaydın. "Localized multiple kernel learning". In: Jan. 2008, pp. 352–359.

[11] Francis R. Bach, Gert R. G. Lanckriet, and Michael I. Jordan. "Multiple kernel learning, conic duality, and the SMO algorithm". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 6. ISBN: 1581138385. URL: https://doi.org/10.1145/1015330.1015424.

[12] Manik Varma and Bodla Babu. "More generality in efficient multiple kernel learning". In: June 2009, p. 134.

[13] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. "Learning Non-Linear Combinations of Kernels". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc., 2009.

[14] Nigel Duffy and David Helmbold. "Leveraging for Regression". In: (Feb. 2001).

[15] Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. "A General and Efficient Multiple Kernel Learning Algorithm". In: Jan. 2006.

[16] Sören Sonnenburg et al. "Large Scale Multiple Kernel Learning". In: *Journal of Machine Learning Research* 7 (July 2006), pp. 1531–1565.

[17] Tom Diethe, David Hardoon, and John Shawe-Taylor. "Multiview Fisher Discriminant Analysis". In: *NIPS Workshop on Learning from Multiple Sources* (Dec. 2008).

[18] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[19] Wouter van Loon et al. "View selection in multi-view stacking: choosing the meta-learner". In: *Advances in Data Analysis and Classification* (Apr. 2024). ISSN: 1862-5355. URL: http://dx.doi.org/10.1007/s11634-024-00587-5.

[20] Fetsje Bijma, Marianne Jonker, and Aad van der Vaart. *An Introduction to Mathematical Statistics*. Amsterdam University Press, 2017. ISBN: 9789462985100. URL: http://www.jstor.org/stable/j.ctt1v2xsxr.

[21] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 42.1 (2000), pp. 80–86. ISSN: 00401706. URL: http://www.jstor.org/stable/1271436.

[22] Jerome H. Friedman, Trevor Hastie, and Rob Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22. URL: https://www.jstatsoft.org/index.php/jss/article/view/v033i01.

[23] L. Breiman. "Stacked regressions". In: *Machine Learning* 24 (2004), pp. 49–64. URL: https://api.semanticscholar.org/CorpusID:11152531.

[24] Michael Leblanc and Robert Tibshirani. "Combining Estimates in Regression and Classification". In: *Journal of the American Statistical Association* 91.436 (1996), pp. 1641–1650. URL: https://doi.org/10.1080/01621459.1996.10476733.

[25] Kai Ming Ting and Ian H. Witten. "Issues in Stacked Generalization". In: *CoRR* abs/1105.5466 (2011). arXiv: 1105.5466. URL: http://arxiv.org/abs/1105.5466.

[26] Wouter van Loon et al. "Analyzing Hierarchical Multi-View MRI Data With StaPLR: An Application to Alzheimer's Disease Classification". In: *Frontiers in Neuroscience* 16 (2022). URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.830630.

[27] Aravind Subramanian et al. "Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles". In: *Proceedings of the National Academy of Sciences* 102.43 (2005), pp. 15545–15550. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0506580102.

[28] Michael E. Burczynski, Randall L. Peterson, Natalie C. Twine, et al. "Molecular classification of Crohn's disease and ulcerative colitis patients using transcriptional profiles in peripheral blood mononuclear cells". In: *The Journal of Molecular Diagnostics* 8.1 (2006), pp. 51–61. DOI: 10.2353/jmoldx.2006.050079.

[29] Xiao-Jun Ma et al. "A two-gene expression ratio predicts clinical outcome in breast cancer patients treated with tamoxifen". In: *Cancer Cell* 5.6 (2004), pp. 607–616. ISSN: 1535-6108. URL: https://www.sciencedirect.com/science/article/pii/S1535610804001412.

[30] Yunzhong Hou, Stephen Gould, and Liang Zheng. *Learning to Select Camera Views: Efficient Multiview Understanding at Few Glances*. 2023. arXiv: 2303.06145.

[31] Zhiwen Cao and Xijiong Xie. "Multi-view unsupervised complementary feature selection with multi-order similarity learning". In: *Knowledge-Based Systems* 283 (2024), p. 111172. ISSN: 0950-7051. URL: https://www.sciencedirect.com/science/article/pii/S095070512300922X.

[32] Stephen Roller and Sabine Schulte Im Walde. "A multimodal LDA model integrating textual, cognitive and visual modalities". In: Jan. 2013, pp. 1146–1157.

[33] Elia Bruni, Nam Khanh Tran, and Marco Baroni. "Multimodal distributional semantics". In: *J. Artif. Int. Res.* 49.1 (2014), pp. 1–47. ISSN: 1076-9757.