UNIVERSITÄT DES SAARLANDES

Saarland Informatics Campus SIC

Saarland University

Master's Thesis

Multivoxel Pattern Analysis for fMRI Data on Program Comprehension

---

Lina Lampel

Supervisor:
Prof. Dr. Sven Apel

Advisor:
Dr. Norman Peitek

First Reviewer:
Prof. Dr. Sven Apel

Second Reviewer:
Dr. Mariya Toneva

February 6, 2023

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, February 6, 2023

Lina Lampel

Abstract

Researchers have already shown that using multivoxel pattern analysis (MVPA) to decode information from functional magnetic resonance imaging (fMRI) data on code comprehension can further our understanding of this complex cognitive task. However, as collecting fMRI data is extremely costly and MVPA is considered a method that requires large amounts of data, so far only very few such analyses have been conducted. At the same time there are several existing fMRI datasets on the topic of code comprehension which have already been analyzed using classical statistical methods but not MVPA. In this thesis we aim to leverage the information hidden in such existing datasets by using MVPA. We have therefore developed decoding pipelines which receive fMRI data as input and output trained models for a certain target variable of interest.

Our results show that we can classify whether a programmer is comprehending code or locating syntax errors based solely on fMRI data ($68\% - 74\%$ balanced accuracy, $p < 0.001$). Furthermore, we can classify whether a programmer is comprehending a code snippet with recursion or iteration and have found that the classification accuracy is higher when the model is trained and tested only on experienced programmers (68% balanced accuracy, $p = 0.001$). We were not able to decode the complexity of a code snippet or the experience level of a programmer.

Overall, our results are promising for some of the target variables, however, in general they suggest that the datasets are not optimal for MVPA, mostly due to their small size.

Contents

List of Figures

List of Tables

List of Acronyms

ANOVA analysis of variance

BA Brodmann area

BAC balanced accuracy

BOLD blood oxygen level dependent

CV cross-validation

fMRI functional magnetic resonance imaging

GLM general linear model

HRF hemodynamic response function

ISI inter-stimulus-interval

LOO leave-one-out

LORO leave-one-run-out

LOSO leave-one-subject-out

ML machine learning

MRI magnetic resonance imaging

MVPA multivoxel pattern analysis

RSA representational similarity analysis

SVC support vector classifier

SVM support vector machine

SVR support vector regressor

TR repetition time

1

Introduction

When it comes to comprehending and writing source code, the brain is arguably the most important organ in the human body. Yet, due to its overwhelming complexity, the brain is also the least understood organ. Scientists agree that further research into how the brain functions during code comprehension could potentially have many advantages [4], [1]. It could not only help us fundamentally understand how the brain processes code, but also based on that information, guide us into how to best train new programmers. Furthermore, it could help us improve programming language and software tool design. All of this in turn, would result in generating high quality code.

A state-of-the-art method of measuring brain activity is functional magnetic resonance imaging (fMRI). fMRI allows us to non-invasively measure and analyze brain activity during the completion of different cognitive tasks. In the past, most fMRI studies have used classical statistical methods, such as general linear models (GLMs) for data analysis [5]. With the help of such methods, one can determine which brain areas are active for certain cognitive tasks. fMRI studies on code comprehension have, for example, shown that there are five brain regions associated with program comprehension [4].

The power of the above described classical statistical approach is, however, limited [6]. In 2001, Haxby et al. were the first ones to demonstrate that even more information can be extracted from neuroimaging data using pattern matching [7]. They showed that different stimuli show specific patterns in fMRI data.

The process of extracting target variables, such as stimuli, tasks, or participant characteristics, from fMRI data is called *decoding*. The set of methods for decoding information from fMRI data is called multivoxel pattern analysis (MVPA). Even though

Haxby et al. used a correlation method in their work, state-of-the-art MVPA methods are based on supervised machine learning [5]. MVPA has already made it possible to train classifiers that distinguish whether a person is viewing a picture or a sentence [8]. Another example is a classifier, that has been trained to distinguish which of multiple presented motion directions a subject is focusing on [9]. In the domain of code comprehension, researchers have been able to decode the following target variables: performed task (i.e., code comprehension vs. prose reading) [1], category of code (i.e., math, search, sort, string) [2], and several others [3].

There are many more targets of interest in the domain of code comprehension that could potentially be decoded and which would help us understand this cognitive task further. Unfortunately, the cost of fMRI studies is extremely high [1]. Therefore it is difficult to collect new datasets for this kind of analysis. In this thesis, we investigate whether MVPA methods can be used to decode information from existing fMRI data on code comprehension. More specifically, we use datasets which were originally collected with the purpose of analyzing them with a classical statistical approach, instead of MVPA. These datasets are, therefore, not optimal for MVPA analysis for several reasons, the most important one being their comparatively small size. In this thesis, we explore, whether we can extract useful information by performing MVPA on these datasets despite their shortcomings in this regard. Our goal is to build three separate machine learning pipelines, each with the aim of decoding a different target variable category - stimuli (see Figure 1.1), tasks, and participant characteristics. The data made available to us for this thesis stems from the following studies [4], [10], [11], [12].

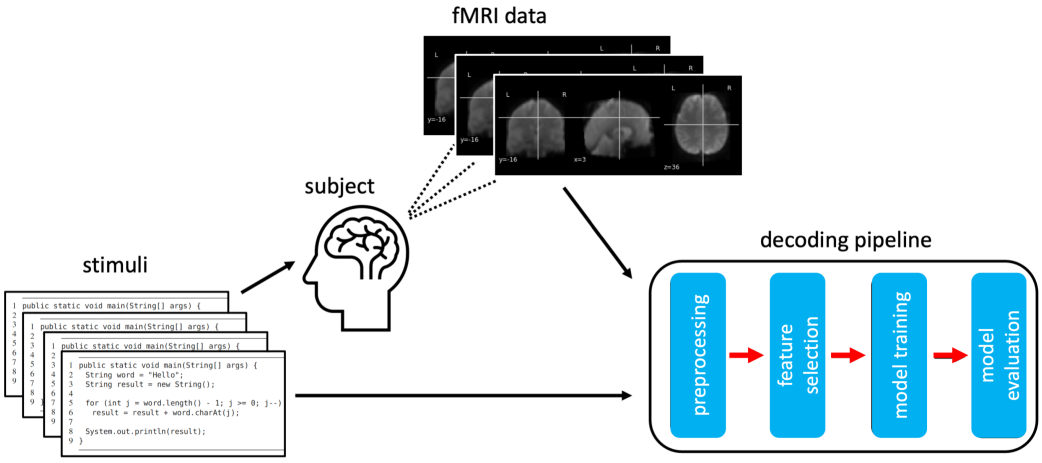


Figure 1.1.: Overview of a decoding pipeline for fMRI data on code comprehension and stimuli as target variables

Our results show that with the underlying datasets we can only predict some of the selected target variables. Furthermore, the results suggest that larger and purposefully collected datasets might be necessary to further explore whether the rest of the target variables can be predicted from neural data.

To summarize, in this thesis we make the following contributions:

- We build MVPA decoding pipelines for task and stimulus prediction of fMRI data.
- We show that the neural representations of the tasks code comprehension and syntax error finding are distinct. This claim is supported by five different models that classify the two tasks based solely on fMRI data in a statistically significant manner (i.e., $68\% - 74\%$ balanced accuracy with $p < 0.001$).
- We show that the algorithm type of a code snippet (recursion vs. iteration) can also be decoded from fMRI data. Further, we show that the resulting model is even more accurate if trained and tested solely on a group of experienced programmers (68% balanced accuracy, $p = 0.001$).
- We were not able to predict the code complexity of code snippets and the experience level of a programmer. We discuss possible reasons why these targets could not be decoded from the underlying datasets and how future research could address these potential problems.

The thesis is structured as follows. In Chapter 2, we introduce the research objectives. In the following chapter, we provide an overview of the background and explain important concepts related to this thesis. In Chapter 4, we present the datasets that have been made available to us for this work. In Chapter 5, we describe our approach in detail. In the following two chapters, we present our results and discuss potential limitations and threats to validity of the approach. In Chapter 8, we provide an overview of work related to our research objectives. Finally, in Chapters 9 and 10, we discuss future work and summarize the findings of the thesis.

2

Research Objectives

In this chapter, we introduce our research questions and discuss how answering them could improve our understanding of code comprehension.

RQ1: Can we use MVPA to predict different tasks from fMRI data on code comprehension?

Comparing other tasks to code comprehension can help us understand in what way they are similar and in what way they differ from the code comprehension task in terms of neural activity. If we can, for example, actually train a model to distinguish between program comprehension and another task, we can then analyze the parameters of the model. Thereby, we can gain information about what regions of the brain make the difference, as well as, what regions of the brain are used in both tasks. Ultimately, we could leverage such information for splitting the task of code comprehension into multiple simpler subtasks that are involved in the process and thereby develop a cognitive model for the complex task of code comprehension. Such knowledge could then guide us towards more efficient ways of training programmers, as we could explicitly train the involved subtasks.

RQ2: Can we use MVPA to predict different stimuli attributes from fMRI data on code comprehension?

There are numerous properties of the presented stimuli, that is the code snippets, that we could try to predict. Static properties, such as different complexity metrics, as well as control flow properties, such as whether a program contains a loop or recursion, or even different coding styles could potentially be predicted. Such an analysis of the

neural data could, for instance, help us investigate which control flow structures are more difficult to understand than others or which complexity metrics are most similar to the actual cognitive load. Being able to understand which control flow structures or which coding styles require more cognitive power from the programmer, could guide us towards more adequate programming practices.

RQ3: Can we use MVPA to predict different participant attributes from fMRI data on code comprehension?

Similarly, there are several different participant attributes that we could try to predict from neural data, such as the type or degree of education, the gender, the age, and the years of experience. Being able to successfully predict these characteristics from fMRI data could enable us to answer questions such as: What makes a programmer excellent? How many years of experience/training are needed to become a good programmer? Do men process code differently than women? Having the answers to these question could then again help us develop more efficient training practices for programmers.

3

Background

In this chapter, we explain the methods and tools that we use for creating our MVPA pipelines. This includes an explanation of supervised machine learning in general as well as descriptions of the specific algorithms and concepts. We also provide an introduction into the structure of fMRI data and MVPA.

3.1. Supervised Machine Learning

Generally speaking, machine learning (ML) is the process of using statistical algorithms to automatically extract information from large amounts of data. *Supervised* ML is the method of using existing data to predict a certain *target variable* (also called *dependent variable*) in new unseen data. Each data point, (also called *sample*), in the data X has one or more features (also called *independent variables*). In contrast to unsupervised ML, we also have a *label* Y - the target variable - for each sample.

Usually, one of the first steps in an ML pipeline, is to split the dataset in two parts. The larger part of the data is used to fit a mathematical model and therefore called *training set*. During the training the label Y can be seen as a "teacher" supervising the training process, thus the name supervised ML. The *predictive power* of this model can then be evaluated using the rest of the data, called *test set*.

Based on the type of the target variable Y the resulting model is either called a *classifier* or a *regressor*. If Y is discrete, the model is a classifier and the label is a *class* or *category*. A simple example for a classifier is a model that classifies emails in the categories "spam" and "non-spam". If Y is continuous, the model is a regressor. A simple example for a regressor is a model that predicts the price of a house based

on features like number of bedrooms, location, square footage etc. In this thesis, we use both classification and regression, depending on the target of interest.

In the following, we shortly explain the idea behind the different ML algorithms, techniques, and performance metrics used in this thesis.

3.1.1. Data Preprocessing

The first step of a machine learning pipeline is to preprocess the data. Depending on the underlying dataset this step can be very different. When working with fMRI data there are several typical fMRI preprocessing steps that are usually performed, such as spatial smoothing, motion correction, Talairach transformation, slice-scan-time correction, and bandpass-filtering [13]. Since the data that has been made available to us has already undergone those preprocessing steps, we do not discuss them in detail here. However, we apply one more preprocessing step called *GLM-based pattern estimation* [14]. This step is a feature extraction preprocessing step that is *not* necessary for a classical GLM analysis of fMRI data, but it is necessary when performing a multivoxel pattern analysis. We introduce this method later in this chapter, as we need some fMRI background to explain it properly.

3.1.2. Feature Selection (ANOVA)

Another step that is usually performed before training a model, is to identify which of the available features could actually have an influence on the outcome variable. Only these are then used for training. The concept of feature selection is even more important when we have an extremely high number of features, which is the case for fMRI data [15]. Some of the datasets in this thesis consist of $902\,629$ features.

There are many different feature selection methods. A popular one is *analysis of variance (ANOVA)* [16], which computes the variance of a predictor within a category of the response. If the variance is equal between the different categories of the output variable, the predictor has no impact on the output variable and is therefore not used for training.

3.1.3. Data Imbalance

We speak of an *imbalanced dataset* when some of the values of the response variables occur much more often than others, that is some classes are underrepresented in the dataset. If we train a model using an imbalanced dataset, it is likely that it favors the overrepresented classes as it has not seen enough samples from the underrepresented classes. This will result in a poor accuracy within the minority classes.

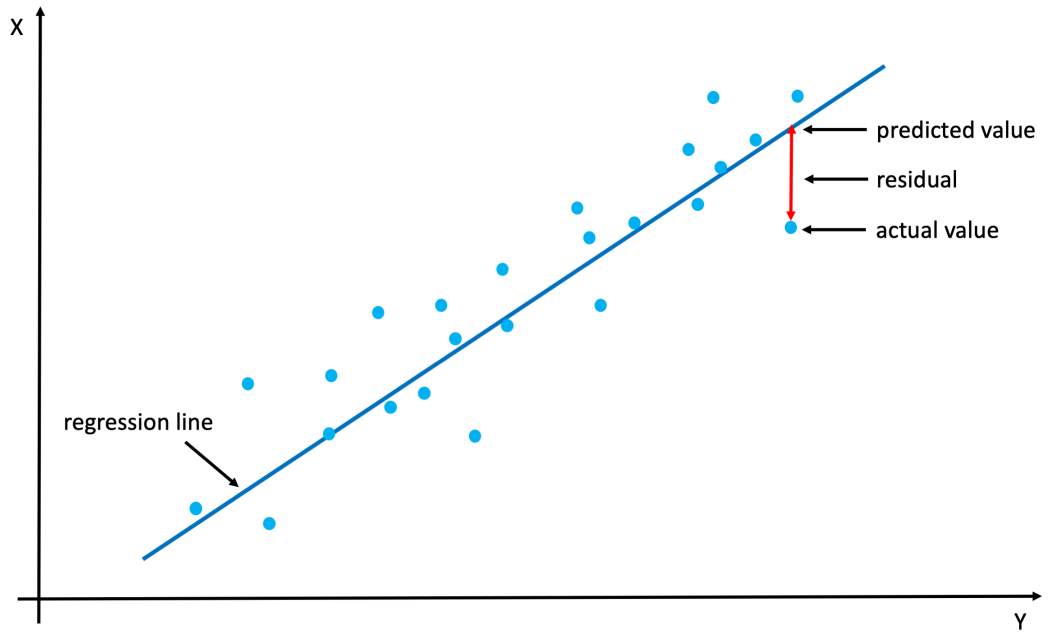


Figure 3.1.: Principle of linear regression in 2D space

3.1.4. Models

Linear Regression

After preprocessing the data and performing feature selection, the next step is to train the model. The most basic regression algorithm we could use for that is simple linear regression [17]. Its goal is to find the best fitting straight line for the data by estimating the parameters of the following equation, where y_i is the dependent variable and x_i is the independent variable:

$$y_i = \beta_0 + x_i\beta_1$$

β_0 is the intercept and β_1 is the slope of the line. The best fitting line is the one that minimizes the sum of the squared *residuals* (see Figure 3.1). Residuals are the distances between the actual and the predicted values. To make a prediction for a new data point, we simply plug its feature values in the computed optimal equation. Simple linear regression is used in the case that we have only a single feature. If we have more than one feature, the algorithm is adapted and called *multiple linear regression* [17].

Support Vector Machine (SVM)

Support vector machines (SVMs) [17] are more adequate than other models for datasets where there are many features and few samples. As this is the case for fMRI, SVMs perform very well on this type of data and are therefore the most popular choice for MVPA [6]. They are mostly used for classification but can also be applied to a regression scenario with some adaptation.

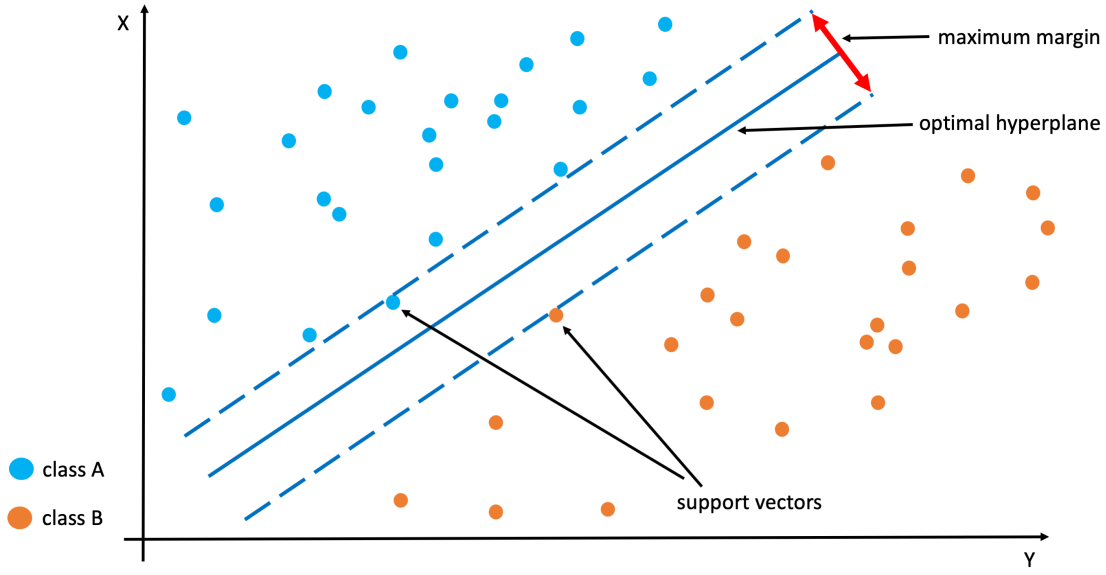


Figure 3.2.: Principle of a support vector classifier in 2D space

A support vector classifier (SVC) [17] aims to compute a *hyperplane* in space such that it separates the samples from different classes as well as possible. The algorithm tries to maximize the *margin*, that is the distance between the hyperplane and the samples from both classes. Samples that lay on the margin are called *support vectors*. Figure 3.2 depicts this principle in 2D. The hyperplane acts as a decision boundary for unseen data - the algorithm decides to which class a samples belongs, by checking on which side of the hyperplane it lays. The dimensions of the hyperplane depend on the dimensionality of the dataset. If we have two features, it is a line. If we have three - it becomes a two dimensional plane.

A support vector regressor (SVR) [17] uses the same principle, however, in this case the algorithm tries to maximize the number of samples that lie within the margin around the hyperplane (see Figure 3.3). The resulting hyperplane is the decision function we use for predicting the labels of unseen samples.

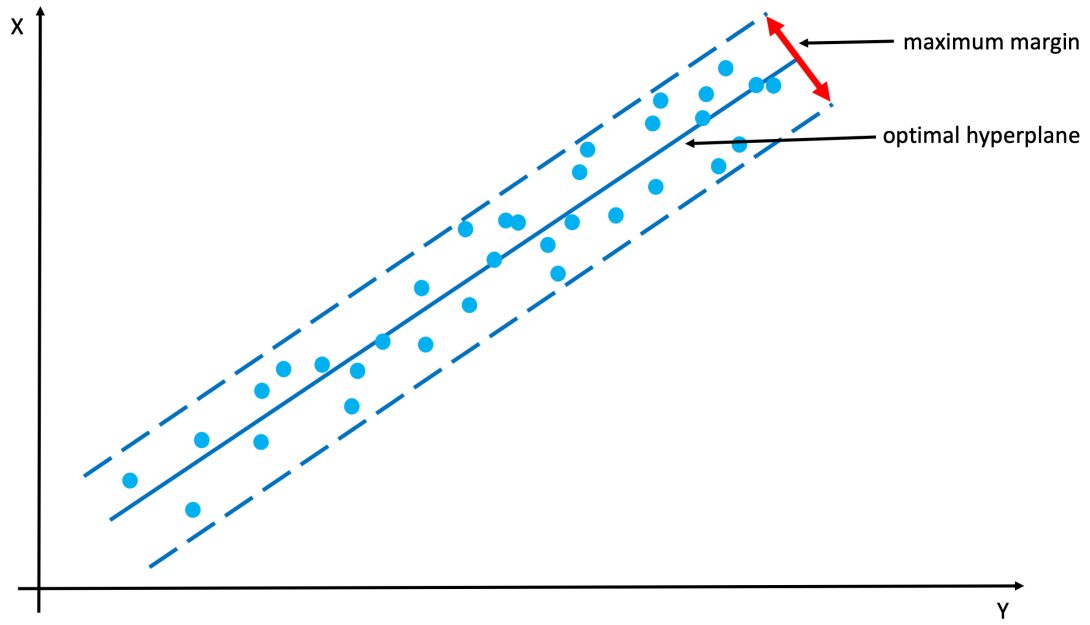


Figure 3.3.: Principle of a support vector regressor in 2D space

Logistic Regression

Logistic regression [17] can also be used for both tasks - classification and regression. However, despite its name, it is mostly used for classification. For that, in a binary scenario the labels are encoded as 0 and 1. Then the *logistic function*, also called *sigmoid function*, is fit to the data (see Figure 3.4). This is done similarly to fitting a linear regression model. However, there are two main differences. First, we do not use least squares for parameter estimation, but instead we use the *maximum likelihood*. Second, the logistic function only outputs values within 0 and 1, which can be seen as the probability of a sample belonging to a certain class. By setting a threshold, we can turn these probabilities into binary predictions. If we set the threshold at 0.5 and get a probability of 0.9, then we classify the underlying sample as belonging to class 1. On the other hand, if the model outputs a probability of 0.4, we classify the sample as belonging to class 0.

The algorithm can be adapted for a multi-class scenario by using the the *one-vs-all* approach. For n different classes, we train n different models, where each model predicts the probability of the sample belonging to some class X vs. the other $n-1$ classes. To make a prediction for an unseen data point, we feed the new sample to all n models and output the prediction of the model with the highest probability.

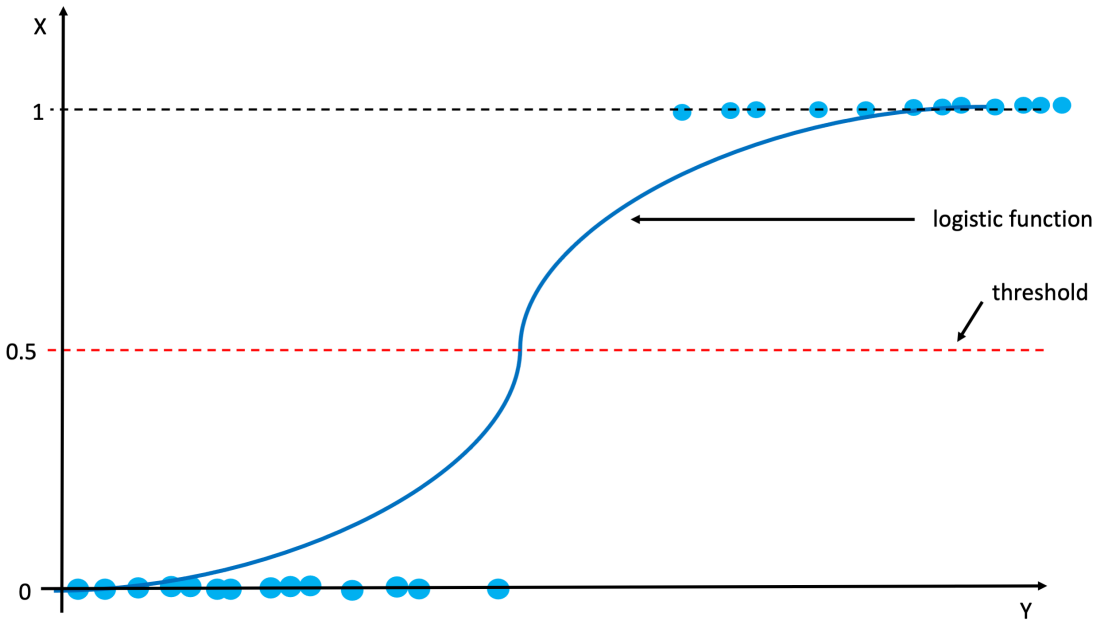


Figure 3.4.: Principle of logistic regression in 2D space

3.1.5. Performance Metrics

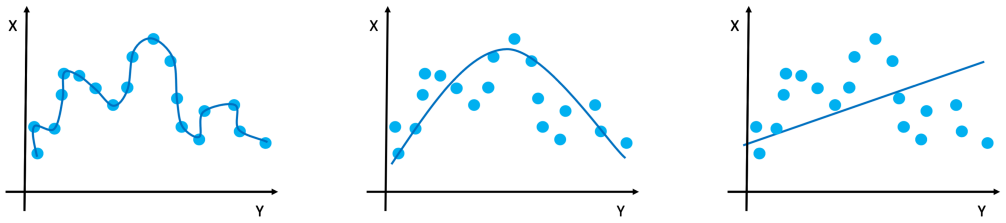
In order to estimate how well a model performs on a certain dataset, we need performance metrics. There are different performance metrics, depending on whether we are evaluating a regression or a classification model. In the following, we introduce the performance metrics used in this thesis.

Accuracy and Balanced Accuracy

Accuracy [17] is a popular metric used for MVPA classification tasks. It tells us how many of the predictions that the model made were correct in percent. An accuracy of 100% means that the model has classified all samples correctly. When dealing with an imbalanced dataset, however, it is preferable to use a slightly modified and more conservative metric called balanced accuracy (BAC) [18]. This metric accounts for the fact that some classes are less common than others and could therefore potentially be favored by the algorithm.

R^2 score

The R^2 score [17] is a performance metric for regressors. It tells us how much of the variance in the dependent variable was explained by the independent variables. The



(a) Overfitting (b) Good fit (c) Underfitting

Figure 3.5.: Models with different complexity levels fit on the same data

best possible R^2 score is 1 and means that the model explains 100% of the variance in the data. The score usually ranges from 0 to 1. It is, however, also possible to have a negative R^2 score. This means that the trained model explains the variance in the data even worse than simply using the mean of the response variables.

3.1.6. Overfitting & Underfitting

When we speak of overfitting, we mean that a model not only learns relevant patterns from the training data but also the included *noise* [17]. Visually, this means that the model tries to interpolate the training data as well as possible (see Figure 3.5(a)). This translates into the model having an excellent performance on the training data. However, this is not desirable as the resulting model usually performs poorly on the testing data. In other words, we can say that it has a high *variance* and does not generalize well. It is therefore bad at making predictions for unseen data.

Underfitting on the other hand, is when a model performs poorly on training and on testing data [17]. It is not able to capture the relationship between the independent and the dependent variables (see Figure 3.5(c)). This means, that it has high *bias* and low variance. This of course, also leads to a model that is bad at making predictions for unseen data.

3.1.7. Regularization

Regularization is an important technique in machine learning, as it prevents overfitting. It is implemented by adding a so-called *penalty* term to the model equation. It prevents the parameters of the model from taking extreme values. The most common regularization techniques are *lasso* (using L1 penalty) and *ridge* (using L2 penalty) [17]. An L1 penalty means adding the absolute value of the sum of the coefficients to the cost function. An L2 penalty means adding the squared of the model coefficients to the cost function. Both methods lead to smaller model parameter estimates, and if

the right amount of regularization is applied, to an improved test set performance. An important difference between the two penalties, however, is that with lasso, the parameters of the model, can potentially be shrunk so far, that they become zero for some features, which means that these features are not used for prediction. This phenomenon occurs when we have features, that do not influence the output. This is why, we can say that lasso regularization is also a feature selection method.

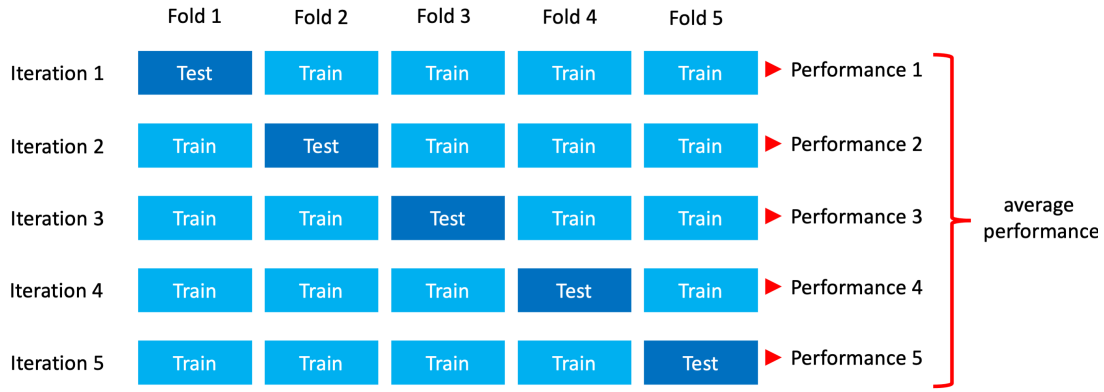


Figure 3.6.: Estimating a performance score by using 5-fold cross-validation

3.1.8. Cross-Validation (CV)

Cross-validation (CV) [17] is a method for estimating the performance of a machine learning model on unseen data . For this purpose the test data is first divided in k so-called *folds*. In each of the k iterations of the algorithm, $k-1$ folds are used to train an ML model while the remaining fold is used to test the model. For each iteration we record the performance of the model on the current fold using a performance metric of our choice. Finally, the k recorded scores are averaged and the result is the CV score. An example of CV with 5 folds can be seen in Figure 3.6. The first step of the algorithm, the division into folds, can either be executed randomly or with a strategy, depending on the type of cross-validation that is performed. The details of the different flavors of CV used in this thesis, will be explained in Chapter 5.

Usually, CV is performed only on the training set and used for hyperparameter tuning. Afterwards, the test score of the best model is reported as a final result. However, since the datasets that we use in this thesis are very small, we do not split them explicitly into train and test set. Instead, we perform cross-validation on the whole dataset and report the CV score as a final result.

3.1.9. Significance Testing (Permutation Test)

After training and evaluating a model, it is also important to test the statistical significance of the resulting performance score. As research has shown, with such small datasets it is not enough to simply compare our results to chance level [19].

A well established significance testing technique for fMRI data is *permutation testing* [20]. The idea behind this technique is the following. First we permute the target variables to randomize the data. Then we train and evaluate the model using the permuted data. We repeat these two steps for a certain number of permutation (e.g., 1000 permutation) and record the resulting scores. This generates a null distribution. Finally, we compute the probability that the original score, that is the one obtained from training the model on the original targets, results from the null distribution. If the probability (i.e., the p-value) is small, it is considered unlikely that our original result is obtained by chance alone and provides evidence that there actually is a relationship between the predictors and the response.

3.2. Functional Magnetic Resonance Imaging (fMRI)

fMRI is a type of neuroimaging developed in the 1990s. Since then it has become a well-established method for measuring brain activity in a non-invasive manner [21]. In this section, we discuss the structure of fMRI data as well as the experiment design of fMRI studies. We also introduce different analysis methods for fMRI, such as GLM-based analysis and multivoxel pattern analysis.

3.2.1. fMRI Structure

In order to explain the structure of fMRI data more in detail, we start by explaining the structure of the more commonly known (non-functional) magnetic resonance imaging (MRI) data. MRI data is a static 3D scan of the brain. These scans consist of so-called *voxels*, which are units of the image with a single value, similar to pixels in regular 2D images. The purpose of MRI is to create a detailed image of the body's tissue and organs. *Functional* MRI on the other hand, is used for measuring changes in the blood flow and thereby measuring brain activity. In order to actually measure a change, however, we need more than one scan of the brain. fMRI is therefore very similar to MRI but has an additional forth dimension - time. Thus, while a person is lying in an fMRI scanner and is shown a certain stimulus, the scanner takes not one but several 3D scans of the brain, called *volumes*. The speed at which the volumes are obtained is called *repetition time (TR)*, which is usually $1 - 2$ seconds/volume. By analyzing

the change in the resulting volumes one can, for example, detect which brain regions were activated after a certain stimulus was presented.

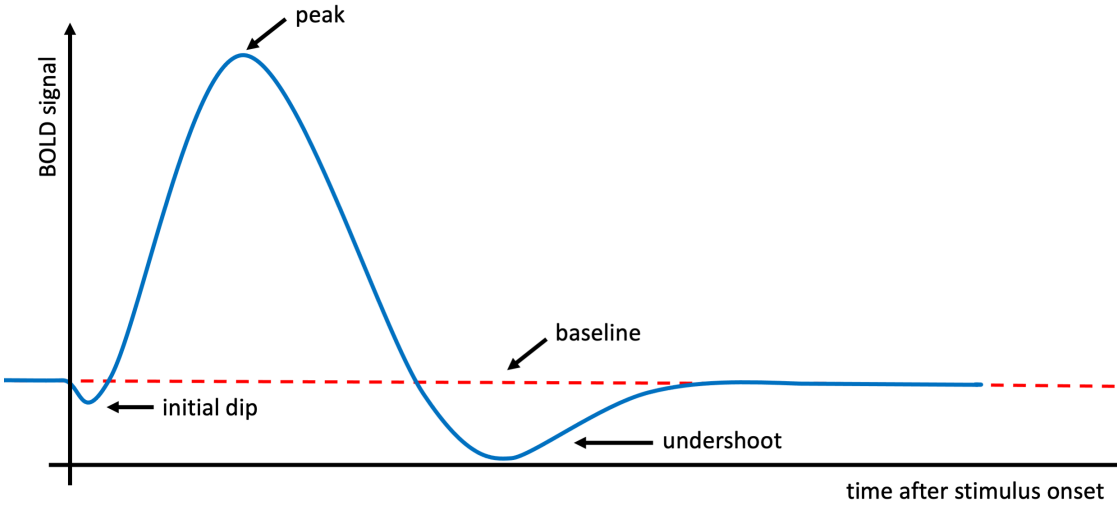


Figure 3.7.: BOLD response to a brief stimulus

3.2.2. Blood Oxygen Level Dependent (BOLD)

More specifically, fMRI measures the *blood oxygen level dependent (BOLD)* effect over time [22]. The more active a certain brain area is, that is the more neural activity there is in this area, the higher the amount of oxygenated blood flowing through it. The more oxygenated a certain brain area is, the higher the measured BOLD effect will be. Hence, the BOLD effect is used as a proxy for brain activity. It should also be noted that the BOLD effect does not appear immediately after a stimulus is presented, since it takes time for the cerebral blood flow to increase in a certain brain area. The peak of the BOLD effect can be measured $2 - 6$ seconds after a stimulus has been presented (see Figure 3.7).

3.2.3. Experiment Design

The fMRI experiments that are of interest for this thesis are typically designed as follows. While a subject is lying as motionless as possible in the fMRI scanner multiple volumes of the brain are acquired. An uninterrupted period of time in which the scanner is acquiring data is called a *session*. The studies from which our data stems, usually have only one session per participant. However, one such session consists of multiple *runs*. This allows the participants to take a short break between runs while

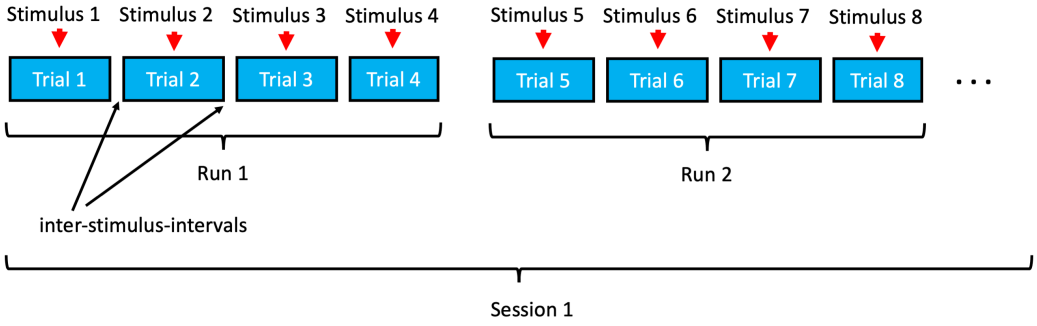
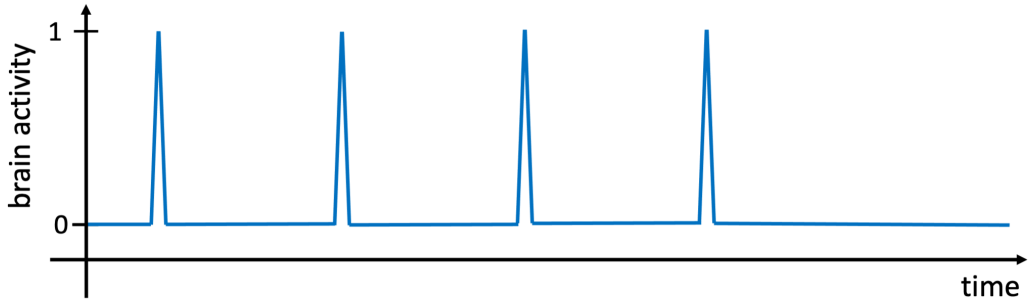


Figure 3.8.: Example of an experimental design with four trials per run

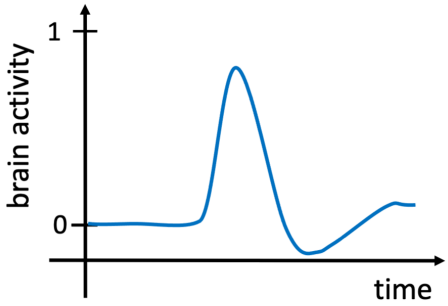
still staying in the fMRI scanner. A single run then again consists of multiple *trials*. During each trial, a single *stimulus* is presented to the subject. In our context, a stimulus is in most cases a snippet of code. The time between two stimuli within one run is called *inter-stimulus-interval (ISI)*. For each stimulus the subject is usually asked to perform a certain *task*. An example of an experimental design can be seen in Figure 3.8. In our context, a task could be to understand what the code snippet does, decide which functional category the code snippet belongs to, or decide whether there is a bug in the code snippet. There are usually at least two stimulus categories within one experiment. This is because we need to be able to distinguish between brain activity that is associated with the part of the task that we are interested in, in our case the comprehending of code, and brain activity that is associated with other parts of the task that are not of interest for the experiment, in our case, for example, merely *reading* source code. The tasks should therefore be as similar as possible but different in the one part we are interested in. One task is called the *experimental condition/task* and one is the *control condition/task*. In our context, an experimental condition could be a snippet of code and a control condition could be a snippet of prose text. The difference of the measured BOLD effect between the two conditions, shows the activation that is needed for the task of interest.

3.2.4. General Linear Model (GLM) Analysis

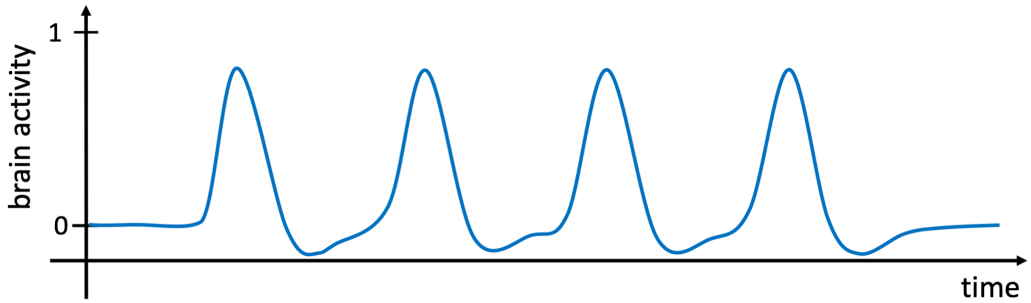
General linear model analysis [23] is the most popular fMRI analysis method. It is a univariate method that can be applied to explore the extend to which a certain brain area is associated with a certain task. To perform a GLM analysis, one needs to specify a so-called *design matrix* first. The design matrix consists of onset vectors (see Figure 3.9(a)), one for each stimulus type, as well as vectors for potential confounds. In order to also incorporate our knowledge about the shape of the BOLD signal (see Figure 3.7), we convolve the onset vectors with the so-called hemodynamic response function (HRF) (see Figures 3.9(b) and 3.9(c)).



(a) Onset vector



(b) Hemodynamic response function (HRF)



(c) Convolved onset vector

Figure 3.9.: Convolution of onset vector with hemodynamic response function

This convolved design matrix and the recorded BOLD signal are used to fit a linear model, where the design matrix is the independent variable X and the BOLD signal is the dependent variable y. The resulting predicted signal can then be further analyzed to answer questions such as "Does the brain process stimuli from category A different than stimuli from category B?" or "Which regions of the brain are involved in processing stimuli from type A?".

A GLM based analysis is, however, limited as it assumes that the covariance across neighboring voxels is not informative and either analyzes voxels individually or averages their signals out [24].

3.2.5. Multivoxel Pattern Analysis (MVPA)

Multivoxel or multivariate pattern analysis goes one step further and analyzes data from multiple voxels jointly by analyzing the underlying pattern, which allows us to *decode* information from fMRI data. MVPA refers to a set of tools for fMRI decoding, however, these tools were not invented specifically for analyzing fMRI data, but are rather a subset of commonly used ML methods that can be applied to fMRI data [25]. Generally speaking, MVPA is considered a supervised classification/regression problem where models are trained to predict information, such as the task or stimulus, from the underlying patterns in fMRI data.

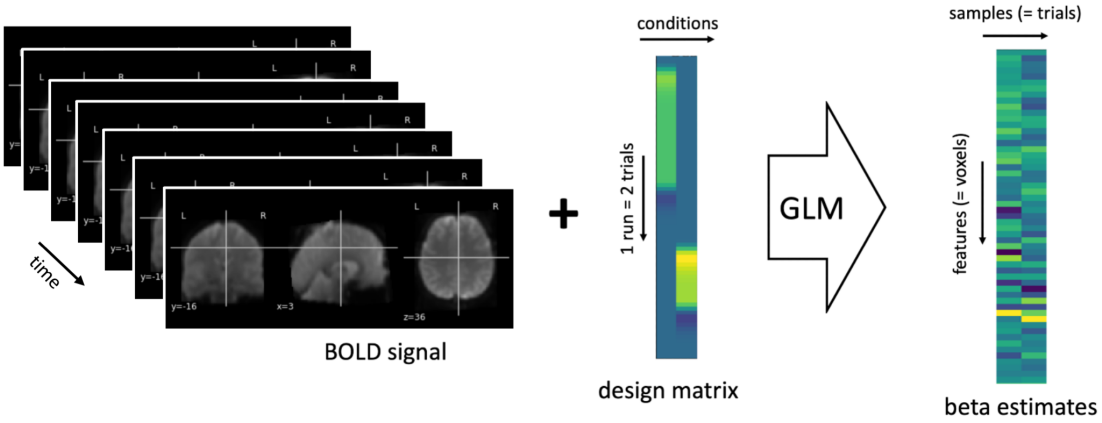
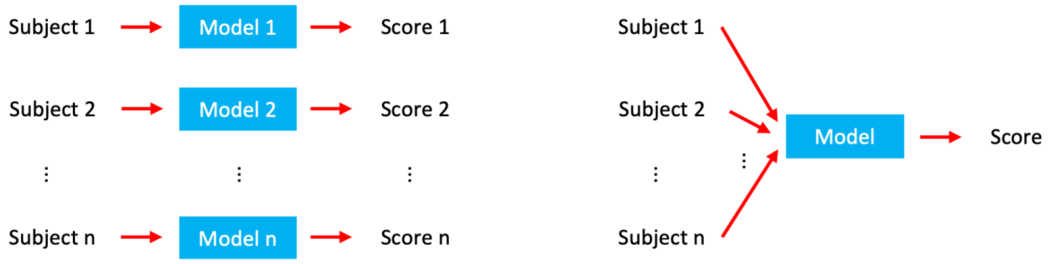


Figure 3.10.: GLM-based pattern estimation as preprocessing step for MVPA

3.2.6. Feature Extraction (GLM-Based Pattern Estimation)

GLM-based pattern estimation is a well-established preprocessing method for MVPA, that extracts features from fMRI data [14]. It transforms the data into samples, each

consisting of a set of features and corresponding labels. What is considered a sample depends on the underlying experiment design and target variable. It could be a single trial or multiple trials. For the purpose of this thesis, a sample is usually a single trial and the features are the voxels. At the end of the transformation, we should have extracted a single volume with three spatial dimensions for each trial. However, as explained in the above sections, in raw fMRI data we have an additional temporal dimension. Hence, we have not one but several volumes for each trial and in each volume we have different BOLD values for each voxel (see Figure 3.10). A naive method to consolidate these different BOLD values from the volumes into one single volume, would be to average them out. This, however, does not lead to a good model performance. Pattern estimation, therefore does not average the values of a trial, but fits a single GLM for each (see Figure 3.10). The estimated model parameters of the GLM are used to create so-called *beta images* for the run, one image per trial. These beta images can now be used as samples. A single beta image contains information about the BOLD response for all voxels during a single trial.



(a) Intra-subject MVPA (b) Inter-subject MVPA

Figure 3.11.: Comparison between intra- and inter-subject MVPA

3.2.7. Single-Subject vs. Group-Level MVPA

In this thesis, we use two different MVPA types: single-subject and group-level. With single-subject (also called intra-subject or within-subject) analysis we train a single model for each subject. This has the disadvantage that we have very few samples for each model and that the generalizability of the results is very limited. However, it has the advantage that within a single subject the neural response patterns are more similar compared to between subjects, where they can vary substantially [25].

Group-level MVPA on the other hand allows for generalization of the results to other subjects. This type of analysis can then again be split up into two more strategies. We can either first perform single-subject analysis and then consolidate the results into a

group-level (also called second-level) analysis or we can directly use the data from all participants to train a model. In this thesis, we use the second group-level approach, called inter-subject analysis as it facilitates interpretation [26].

3.2.8. Brodmann Areas (BAs)

After performing a GLM or MVP analysis, we usually map our results to anatomical or functional regions of the brain. This can for example allow us to draw conclusions about which parts of the brain are involved in processing a certain task or stimulus. The Brodmann areas (BAs) [27] are an anatomical classification system often used in this context. There are 52 BAs in the cerebral cortex of humans. With the help of previous studies, researchers have associated different functions with the different areas. By linking these results from the literature, we can identify which subtasks are involved in a complex task such as code comprehension.

4

Data

In this chapter, we provide a detailed overview of the data that was made available to us for evaluating our approach and introduce the underlying data format. The datasets stem from four different studies on code comprehension. An overview of the studies can be seen in Table 4.5. Altough we do not use all four of the datasets in this work, we do present all of them in this chapter, as it was part of the thesis to analyze them and to decide which ones are most suitable for the previously defined research objectives.

Table 4.1.: Sequence of a trial in study No.1

condition	duration
comprehension task	60s
rest	30s
syntax task	60s
rest	30s

4.1. Study No.1

Siegmund et al. conducted the first fMRI study on program comprehension [4]. As part of this exploratory study, they collected fMRI data from participants comprehend-

ing short source-code snippets in a bottom-up[1] manner. The experiment consisted of two kinds of tasks. During the so-called comprehension task, participants were presented with code snippets and had to identify the program's output. During the so-called syntax task, the participants had to identify syntax errors in code snippets. The code snippets were short algorithms that are taught in first-year undergraduate computer-science courses. The sequence of a single trial can be seen in Table 4.1. The authors used a random-effects GLM analysis on the collected fMRI data and found that there are five brain regions associated with understanding bottom-up program comprehension - BA 6, 21, 40, 44 and 47. Previous research has shown that these brain areas are related to the following cognitive processes: natural-language comprehension, problem solving, and working memory.

Table 4.2.: Sequence of a trial in study No.2

condition	duration
semantic-cues comprehension (beacons and layout pretty)	30s
rest	30s
semantic-cues comprehension (beacons and layout disrupted)	30s
rest	30s
semantic-cues comprehension (no beacons and layout pretty)	30s
rest	30s
semantic-cues comprehension (no beacons and layout disrupted)	30s
rest	30s
finding syntax errors or bottom-up comprehension	30s
rest	30s

4.2. Study No.2

In a second study [10] within this research field, Siegmund et al. explored the difference between brain activity associated with bottom-up code comprehension and brain activity associated with top-down[2] code comprehension. For this purpose, the design of the experiment was slightly more complex than the last and consisted of two sessions, a training session outside the fMRI scanner and a regular fMRI session. During the training session the participants familiarized themselves with some code

[1] Bottom-up code comprehension means that the code snippet does not contain familiar semantic cues. Therefore, the programmer needs to understanding every individual statement in order to understand the whole program. [10]

[2] Top-down code comprehension means that the code snippet contains familiar semantic cues that guide the understanding of the program. [10]

snippets containing semantic cues. This ensured that later, during the fMRI session, they would be able to employ top-down comprehension. During the fMRI session, the participants were also presented with code snippets, but this time the task was to determine whether the stimuli correctly implemented the same functionality as one of the code snippets they had already seen during training. Some of the stimuli presented during the second session were printed in a more readable layout and/or contained semantic cues, so-called beacons, while others did not. This was the experimental condition of the study. The sequence of one trial can be seen in Table 4.2. Similar to the last study, the control condition was locating syntax errors and the analysis method was a random effect GLM analysis. Using the described experimental set up, Siegmund et al. found that bottom-up and top-down code comprehension activate the same brain areas, but that the latter is associated with a lower neural activation suggesting that it is more efficient. However, the results showed that beacons or layout do not significantly influence the comprehension process.

Table 4.3.: Sequence of a trial in study No.3

condition	duration
top-down comprehension (trained, with beacons)	30s
distractor task	15s
rest	20s
bottom-up comprehension	30s
distractor task	15s
rest	20s
top-down comprehension (trained, no beacons)	30s
distractor task	15s
rest	20s
top-down comprehension (untrained, with beacons)	30s
distractor task	15s
rest	20s
finding syntax errors	30s
distractor task	15s
rest	20s

4.3. Study No.3

Peitek et al. [11] were the first ones to simultaneously measure eye tracking during an fMRI experiment on code comprehension. Since eye tracking has a much higher temporal resolution than fMRI (i.e., milliseconds instead of seconds), the combination

of these two measurements allows to more precisely identify the origin of neural activations in time. The study design of the authors is almost the same as the previously described one and therefore also focuses on the difference between bottom-up and top-down comprehension. Besides the addition of eye tracking, the authors also added a distractor task as a control condition prior to the rest periods in order to prevent the participants from thinking about the code snippets from the comprehension task during the rest period. The sequence of one trial can be seen in Table 4.3. The results of the experiment show that combining fMRI and eye tracking is challenging, but allows for new observations. Using this method the authors were able to confirm that there is a fixation on beacons before a particular neural activation (in BA 21) takes place. This confirms that programmers make use of programming plans in top-down code comprehension.

Table 4.4.: Example sequence of a trial in study No.4

condition	duration
code comprehension	max. 60s
distractor task	10s
rest	30s
code comprehension	max. 60s
distractor task	10s
rest	30s
code comprehension	max. 60s
distractor task	10s
rest	30s
finding syntax errors	max. 30s
distractor task	10s
rest	30s

4.4. Study No.4

In this work [12], Peitek et al. investigate whether and how complexity metrics reflect the difficulty of bottom-up program comprehension. For that they conducted an fMRI study observing program comprehension of short code snippets with a wide range of complexities. The design of this study was more complex that the previous ones as there was not a fixed trial sequence order and the time for which participants were presented a stimulus was not fixed either. An example sequence of the study can be seen in Table 4.4. In this study too, the authors performed a random-effects GLM analysis. The results of the study show that complexity metrics have a weak to

medium correlation with programmer's cognition during program comprehension and offer insights into why some code properties, such as code's textual size and vocabulary size, are difficult to process.

4.5. Data Format

The fMRI data from these studies comes is in the form of NIfTI[3] files. One file per session, in most cases also per participant. Since this is also one of the most popular file formats for this type of data, in this work we will focus only on this format. However, if needed, fMRI data from other formats can be converted in a NIfTI format using standard neuroimaging software packages. Additionally to the 3D scans of the participant's brains over time, the NIfTI files contain more information, called the affine. The affine is a 4x4 matrix which allows us to map voxel coordinates to real world coordinates by taking the dot product of the the affine and a set of voxel coordinates.

The meta data of the experiments are provided in the form of CSV[4] files. For each experiment we have one file containing the information about the stimuli - at what point in time was which stimuli shown to the participants. Additionally, for each experiment we also have a file containing information about the participants, such as age, gender, or experience level.

[3] Neuroimaging Informatics Technology Initiative file format
[4] Comma-Separated Values file format

Table 4.5.: Overview of studies on code comprehension which serve as data sources for this work

Study No.	1	2	3	4
topic	bottom-up code comprehension	difference between bottom-up and top-down comprehension	difference between bottom-up and top-down comprehension	code complexity metrics
modality	fMRI	fMRI	fMRI, eye tracking	fMRI, eye tracking, physio
number of participants	17	11	22 (10 eye tracking complete)	18
number of sessions	17	14 (3 participants twice)	22	18
session duration	32min 50s	62min (2×31min)	28min	45min
number of volumes per session	930	985	878	variable
number of trials	12	12	5	5
trial duration	150s	300s	335s	variable
TR	2s	2s	2s	1.2s

5

Method

To answer the research questions presented in Chapter 2, we proceed as follows:

1. We choose the most adequate dataset for the underlying scenario, depending on the target of interest.
2. We preprocess the data using GLM-based feature extraction.
3. We use feature selection to reduce the dimensionality of the data.
4. We perform an intra-subject as well as a inter-subject MVPA on the data and test the significance of our results using permutation testing.

For the implementation of the above we mainly use scikit-learn (v1.1.2) [28], a general machine learning library, as well as nilearn (v0.9.2) [29], an ML library for the neuroimaging domain built on top of scikit-learn. For the input and output of the brain images we use the package nibabel (v4.0) [30]. Finally, we use BrainNet viewer (v1.63) [31] for the visualizations of the models' weights.

5.1. Task Prediction (RQ1)

For our first research question, we investigate whether we can predict the task a participant was performing using fMRI data only.

5.1.1. Dataset Selection and Analysis

In the above described scenario, our target of interest is the underlying task. Therefore, we need a dataset with at least two different tasks (A and B), one of them being code comprehension. Ideally, this dataset should be balanced, that is, it should contain an equal or at least similar number of trials with task A and task B. This is important, as otherwise the trained model is likely to prioritize the class (i.e., task category) which occurs more often. Furthermore, we want to have a dataset with as many samples as possible, since machine learning models perform better with more data. Finally, we want to have a dataset where besides the task there is not much difference between the trials. This means, that the difficulty and the presentation of the stimuli should be similar.

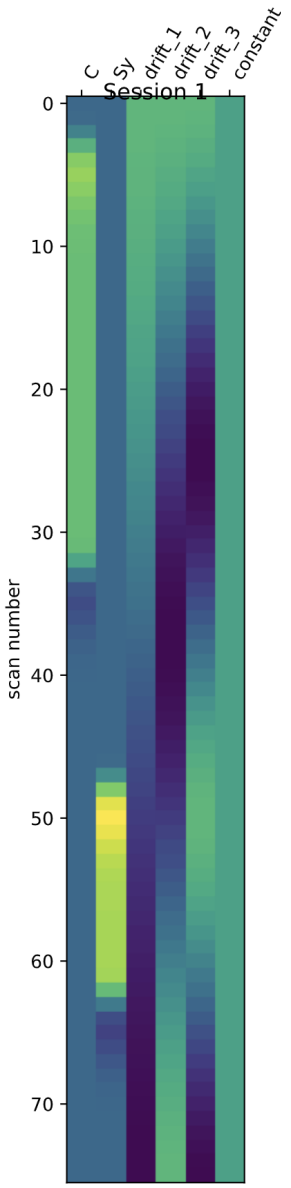
All datasets, apart from Study No.1, are not suitable for task prediction using MVPA/ This is because in these studies within one category of task, there are different categories of stimuli shown (e.g., with beacons vs. without beacons). Therefore there are very few samples for each of the resulting subclasses.

In Study No.1 we have two tasks: code comprehension and identifying syntax errors. The difficulty (measured using complexity metrics) and presentation of the code snippets are similar across trials. During a session, a participant performs each task 14 times, which means that the classes are perfectly balanced and that for each session and participant we have 28 trials. As we have 16 participants, our dataset consists of $28 * 16 = 448$ trials overall. Each trial consists of $91 * 109 * 91 = 902\,629$ voxels and a certain number of slices. The number of slices depends on the duration of the task as well as the repetition time. Trials where the code comprehension task was performed consist of $60\,\text{s} \;/\; 2\,\text{TR} = 30$ slices and therefore overall of $902\,629$ voxels $* 30$ slices $= 27\,078\,870$ floating point numbers. Trials where the syntax task was performed consist of $30\,\text{s} \;/\; 2\,\text{TR} = 15$ slices and therefore overall of $902\,629$ voxels $* 15$ slices $= 13\,539\,435$ floating point numbers.

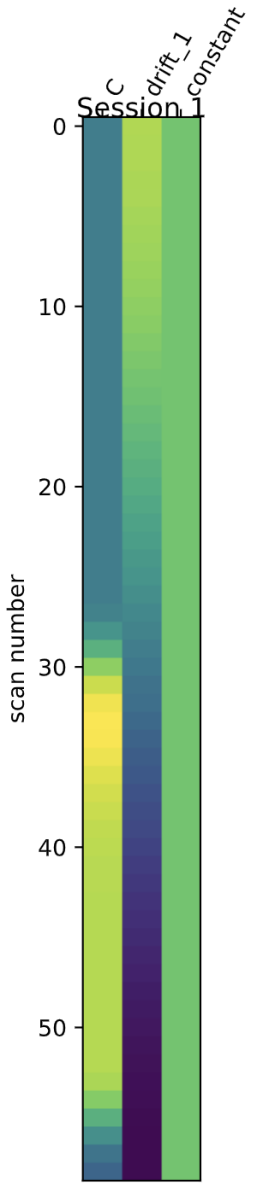
5.1.2. Data Preprocessing

This dataset has already undergone the typical fMRI preprocessing steps, including 3D-motion correction, linear trend removal, high-pass filtering, co-registration, and spatial smoothing. Therefore, in the following we only discuss preprocessing steps specifically related to the decoding analysis.

As discussed in Section 3.2.6 a conventional preprocessing step in MVPA is to transform the fMRI data, such that we end up with a matrix of samples with corresponding features. For this purpose, we first create a design matrix. Since in this dataset all runs look the same for all participants, we have the same design matrix for every run of each participant (see Figure 5.1(a)). Then we fit a single GLM per run using the design matrix, which results in two beta images each, as we have two trials within



(a) Design matrix for one run (i.e., two trials) for task prediction (C = code comprehension, Sy = locating syntax errors)



(b) Example of a design matrix for one trial for stimulus prediction (C = code comprehension)

Figure 5.1.: Design matrices used for feature extraction

a run. Overall, we end up with 28 beta images (i.e., samples) per participant. We standardize these beta images to zero-mean and unit variance.

5.1.3. Feature Selection

The dimensionality of our dataset is very high (i.e., $902\,629$ features), while we only have a small number of samples (i.e., 28 per participant or 448 overall). In order to improve this ratio, we perform feature selection for both intra- and inter-subject MVPA in the following way. For each CV iteration during the analysis, before training, we use a one-way ANOVA F-test and keep only the top 30% features/voxels with the highest scores for the current iteration.

5.1.4. Models and Metrics

We have selected five different machine learning models commonly used for MVPA and perform all the following steps of our approach using each of the models. In order to mitigate overfitting, all of the selected models use a regularization technique:

- Linear Support Vector classifier with L2 penalty
- Linear Support Vector classifier with L1 penalty
- Logistic Regression with L2 penalty
- Logistic Regression with L1 penalty
- Ridge classifier (= Linear Regression with L2 penalty)

Although this dataset is balanced, we use balanced accuracy as a metric in order to have a consistent metric throughout all classification tasks in this thesis. Furthermore, this metric is stricter than regular accuracy.

5.1.5. Intra-Subject MVPA

For the intra-subject MVPA we train each of the above models with the data of each participant separately, such that we end up with $5 * 16 = 80$ different models. Each model is trained using cross-validation with a leave-one-run-out (LORO) strategy. This means that during each CV fold, we do not randomly split the data, but leave out the data from one run as a test set. This is a common cross-validation strategy for MVPA, as it ensures that the train and test set are independent from each other while at the same time being more stable than leave-one-out (LOO) CV, where a single trial is left out as a test set [20]. With our dataset, LORO CV boils down to training a model

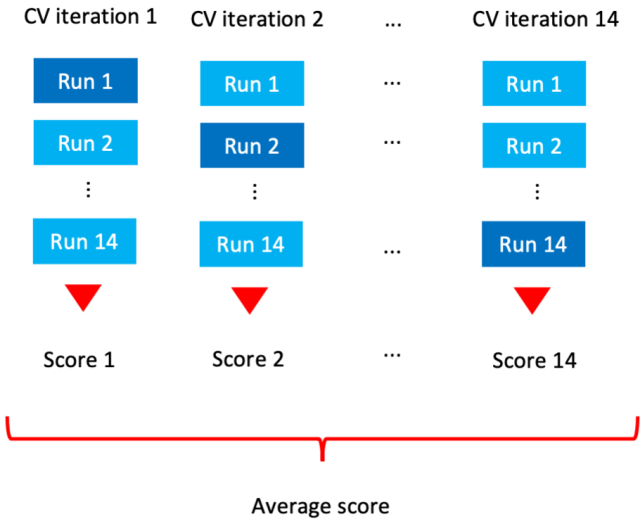


Figure 5.2.: Intra-subject training procedure using leave-one-run-out cross-validation. Runs which are part of the training set for the current CV iteration are shown in light blue, while the ones used as test set are shown in dark blue.

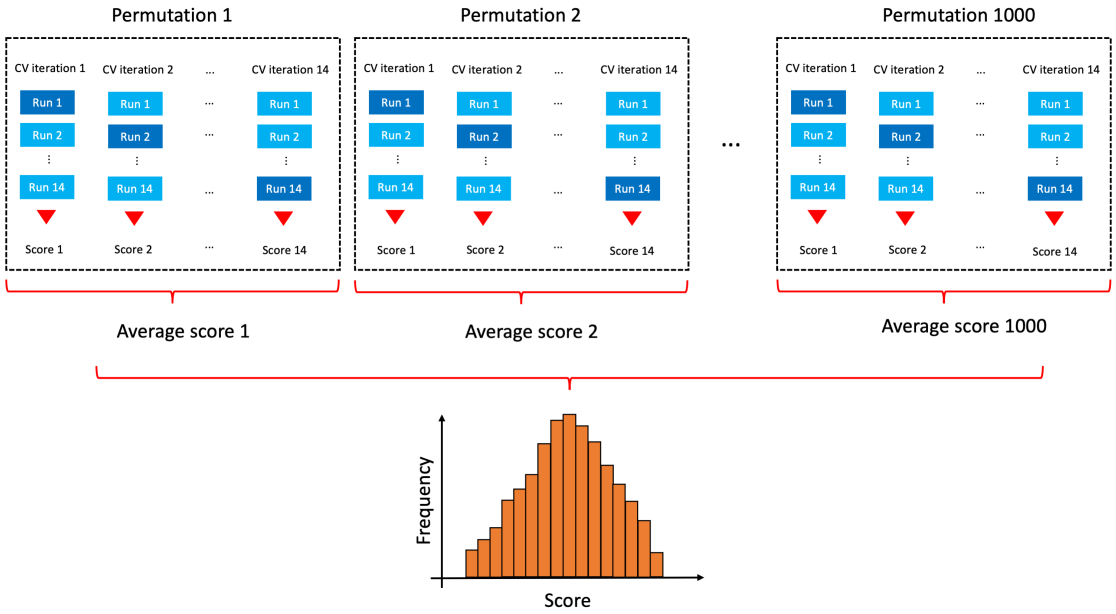


Figure 5.3.: Intra-subject significance testing using a permutation test with 1000 iterations and leave-one-run-out cross-validation. Runs which are part of the training set for the current CV iteration and permutation iteration are shown in light blue, while the ones used as test set are shown in dark blue.

using 26 samples as a training set and 2 samples as a test set during each CV iteration (see Figure 5.2).

For each subject and model we compute the significance of our results by performing a permutation test with 1000 permutations (see Figure 5.3). During each iteration of the test we use the exact same procedure as while training with the original labels. The only difference between the permutation iterations is the order of the labels of the data, which are shuffled each time within the CV groups.

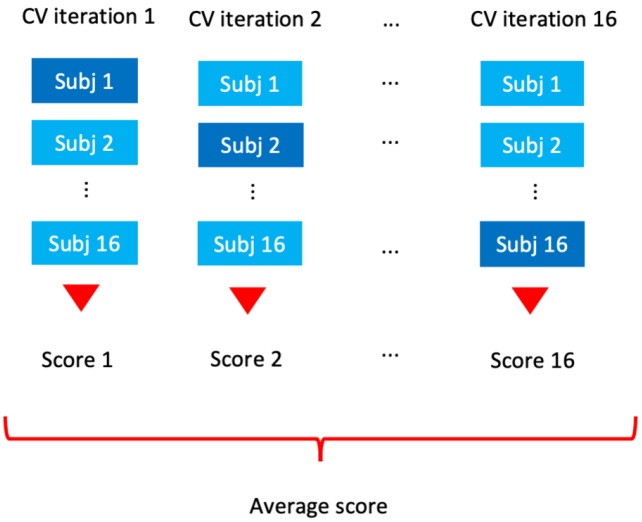


Figure 5.4.: Inter-subject training procedure using leave-one-subject-out cross-validation. Subjects which are part of the training set for the current CV fold are shown in light blue, while the ones used as test set are shown in dark blue.

5.1.6. Inter-Subject MVPA

For the inter-subject MVPA we train each of the above mentioned classifiers with the data of all participants instead of just one, such that we end up with 5 different models.

Each model is trained using cross-validation with a leave-one-subject-out (LOSO) strategy. This means that during each CV fold, we leave out the data from one participant (i.e., 28 samples) as a test set, such that we train a model using 420 samples as a training set and 28 samples as a test set (see Figure 5.4).

For each model we compute the significance of our results by performing a permutation test with 1000 permutations (see Figure 5.5).

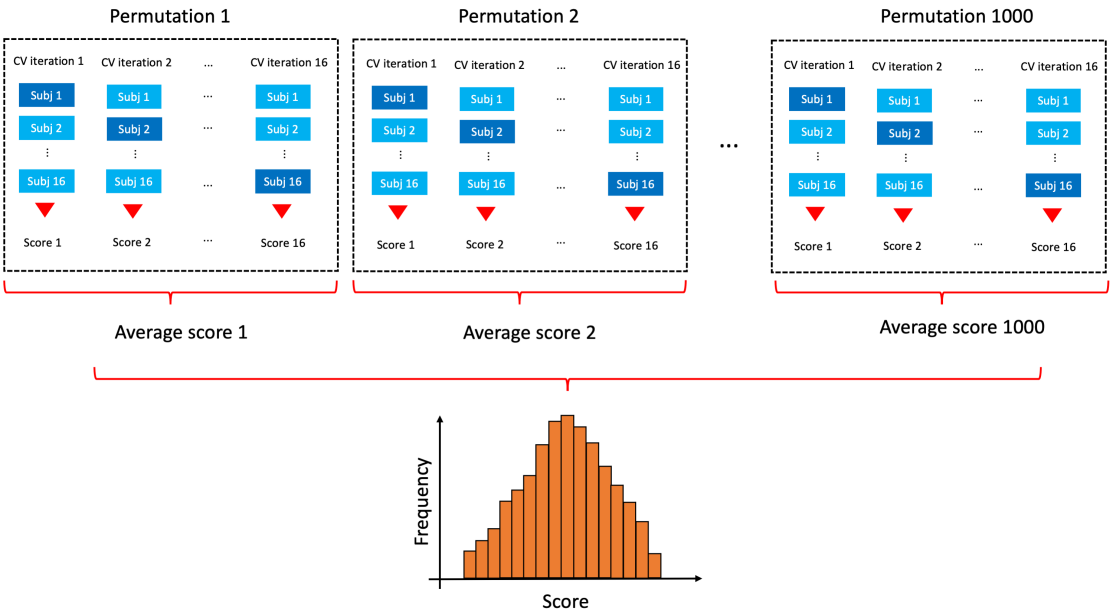


Figure 5.5.: Inter-subject significance testing using a permutation test with 1000 iterations and leave-one-subject-out cross-validation. Subjects which are part of the training set for the current CV fold and permutation iteration are shown in light blue, while the ones used as test set are shown in dark blue.

5.2. Stimulus Attribute Prediction (RQ2)

To answer our second research question, we investigate whether we can predict attributes of the stimulus a participant was presented using fMRI data only.

5.2.1. Dataset Selection and Analysis

As stated in Chapter 1, there is a wide variety of stimulus characteristics that we could focus on. In this work, we focus on attributes which are well differentiated in our datasets. It is important, that we select a dataset where the only difference between the samples is the stimulus attribute of interest, instead of, for example, the task. Therefore, we need a dataset where for most of the trials the task is code comprehension. This is the case for all datasets, apart from the one from Study No.1.

The stimulus attributes that could potentially be decoded in Study No.2 and 3 revolve mostly around code presentation, the ones in Study No.4 are more associated with code structure. Both code presentation and code structure are important parts of code comprehension, which should be analyzed further. We choose to use Study No.4 for this thesis, as it is the most recent study using a newer fMRI protocol, and

we therefore have a higher temporal resolution.

Study No.4 contains data from 18 participants, however, we exclude the data from one participant as it could not be processed due to format issues, and we are therefore, left with 17 participants. During a session, each participant performs three different tasks: code comprehension, identifying syntax errors, and a distractor task. As we are only interested in code comprehension, we only use these trials for the following analysis. The code comprehension stimuli in the dataset differ among others along the attributes algorithm type and code complexity. Since we need different methods for predicting these two stimuli attributes, we split the following procedure description in two parts.

Algorithm Type

Each participant is shown 16 stimuli from three different algorithm categories:

- 7 x Iteration
- 6 x Recursion
- 3 x Conditionals

As the conditionals class has only 3 members per participant, we exclude the samples from this class from the analysis since the dataset would otherwise be highly imbalanced. This means, that we are left with 13 samples per participant and $13 * 17 = 221$ samples overall.

Complexity Metrics

In addition to the algorithm type, we aim to predict the following four complexity metrics, all stemming from different complexity metric classes:

- LOC (code size)
- McCabe (control-flow complexity)
- Halstead (vocabulary size)
- DepDegree (data-flow complexity)

As Figure 5.6 shows none of the metrics are balanced in this dataset. In this case, however, the imbalance is not easy to remediate. There are common ML techniques for counteracting imbalance, such as over- and undersampling, but these are difficult to apply on such a small sized and high dimensional dataset. Therefore, we use the dataset as is. This means, that for each participant there are 16 samples and $16 * 17 = 272$ samples overall.

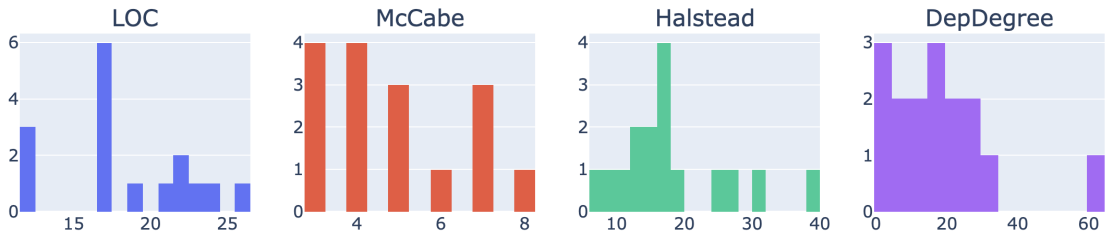


Figure 5.6.: Distribution of complexity metric classes within one session

5.2.2. Data Preprocessing

This dataset, too, has already undergone the typical fMRI preprocessing steps. Thus, we only need to perform feature extraction as described in Section 3.2.6. We fit one GLM for each code comprehension trial. However, as in this dataset each session has a different length, the design matrices for each trial are also different. An example of a design matrix for this dataset can be seen in Figure 5.1(b). We standardize the resulting beta images to zero-mean and unit variance.

5.2.3. Feature Selection

We perform the same feature selection steps as in Section 5.1.3.

5.2.4. Models and Metrics

Algorithm Type

As the algorithm type is a categorical variable, predicting it results to a classification task. We use the same models and metrics as described in Section 5.1.4.

Complexity Metrics

Predicting the complexity degree, however, is a regression task as the target is continous. Therefore, we do not use classifiers as models but the following two commonly used regressors:

- Support Vector Regressor with L2 penalty
- Ridge Regression (=Linear Regression with L2 penalty)

We use the R^2 score as a scoring metric for our models, as it is easier to interpret than other commonly used regression metrics.

5.2.5. Intra-Subject MVPA

Algorithm Type

First we perform an intra-subject MVPA by training the five classifiers for each participant separately. We end up with $5 * 17 = 85$ different models. The runs in this dataset are comparatively longer than in the last. If we were to use, leave-one-run-out CV, we would be left with too few samples for training. Therefore, we train the models using 6-fold CV instead, which means that during each CV iteration we randomly select $1/6$ (i.e., $2 - 3$ samples) of the dataset samples to be used as a test set for the current fold. As there is a distractor task as well as a rest period between each code comprehension task, the samples are independent from each other and it is not a problem that trials from the same run could be in the train and in the test set. Other than the cross-validation strategy, the training and testing procedure remains the same as in Section 5.1.5.

Complexity Metrics

For the intra-subject of the complexity metrics, we also use the same method as for the task prediction but with 8-fold CV. This is because, in contrast to the preprocessing of the dataset for the algorithm type, in this case we did not filter out any samples but use the dataset as is.

5.2.6. Inter-Subject MVPA

Algorithm Type

We perform the inter-subject MVPA in the exact same way as already described in Section 5.1.6 and end up with five different models.

Complexity Metrics

The inter-subject MVPA for the different complexity metrics is also very similar to the one for the task prediction (see Section 5.1.6) with minor differences. We perform the training, evaluation, and significance testing for each complexity metric separately. As with the regression task we compare only two different models (i.e. SVR and ridge regression), we end up with two instead of five different models per target (i.e., complexity metric).

5.3. Participant Attribute Prediction (RQ3)

Similarly to the stimulus attributes, there are many different participant attributes that we could try to predict, such as the experience, age, gender, education, etc. In this thesis we focus on the participant's programming experience as this is the most interesting participant attribute in the field of code comprehension. For all our datasets we have a so-called programming experience score for each participant, which has been determined using a validated questionnaire [32].

The prediction of participant attributes is a more difficult task than the last two tasks, as for each dataset we have only as many samples as we have participants. Therefore, a different approach is needed than the one described at the beginning of this chapter.

5.3.1. Approach No.1

One approach which has already been shown to be successful, is to first perform an intra-subject MVPA on a different target, for example the task, and then, in a second step, to examine the relationship between the performance of the trained models and some participant characteristic of choice (e.g. the expertise) [1]. If there is a significant relationship between these two variables, we could then use it to indirectly predict the participant characteristic.

Unfortunately, our previous results do not allow us to take this approach, as we do not have an intra-subject MVPA, where all the results are significant. Therefore, we cannot use any of our intra-subject MVPA results as a basis for the approach. However, the inter-subject MVPA of task prediction clearly shows significant results (Figure 6.2). Hence, in this approach we use the dataset from Study No.1 and leverage our previous result as follows. We use only the best performing model in this case (i.e. logistic regression (L1)) and examine the relationship between the model's performance for each subject's samples and the subject's experience. This means, that we investigate, whether the model can classify more accurately whether a subject is performing code comprehension or locating syntax errors, if the subject is more experienced (or less experienced). The model is trained only on the samples of the remaining participants. If such a relationship exists, then we can indirectly use it to predict the experience score of a participant.

5.3.2. Approach No.2

Finally, we examine one more approach which does not directly predict the participant's experience but is closely related to this topic. We investigate whether models which classify the algorithm type (recursion vs. iteration) and are trained only on a group of more experienced programmers, perform better than models only trained on

a group of less experienced programmers. If this is the case, it would show that the difference between recursion and iteration algorithms is more clearly represented in the brain images of experienced programmers.

To evaluate this hypothesis, we use the same dataset (i.e. Study No.4) and the same set up as in Section 5.2, but perform two separate inter-subject analyses - one where we train and test the models on a group of experienced subjects, and one where we train and test the models on a group of less experienced subjects.

Figure 5.7 shows the distribution of the subjects among the two groups. As the programming experience score of one participant is missing in this particular dataset, each group consists of 8 participants.

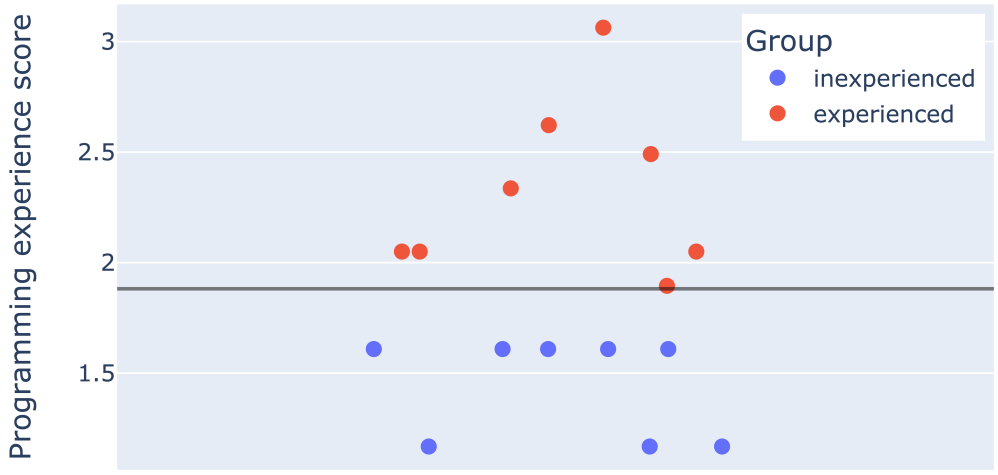


Figure 5.7.: Distribution of participants among groups based on programming experience score. The gray line shows the mean programming experience score.

6

Results and Discussion

In this chapter, we present the results of our multivoxel pattern analysis on the two selected datasets. This allows us to answer our three research questions and discuss their implications.

We assume a the commonly used statistical threshold of $p < 0.05$ for all all significance tests in this thesis.

6.1. Task Prediction (RQ1)

As we have done in the previous chapters, we first examine the underlying task as a target of interest. Our models try to predict the difference between the tasks code comprehension and locating syntax errors.

6.1.1. Intra-Subject MVPA

Figure 6.1 shows the results of the intra-subject MVPA for the first four participants. The results for all subjects can been seen in Appendix A. The permutation tests reveal that the results are only significant for some participants and model types. Overall, we have at least one significantly performing model for 6 out of 17 participants. However, there is not one model type which is significant for all these 6 participants. Furthermore, there is not one model type which performs best over all participants, the results are rather mixed.

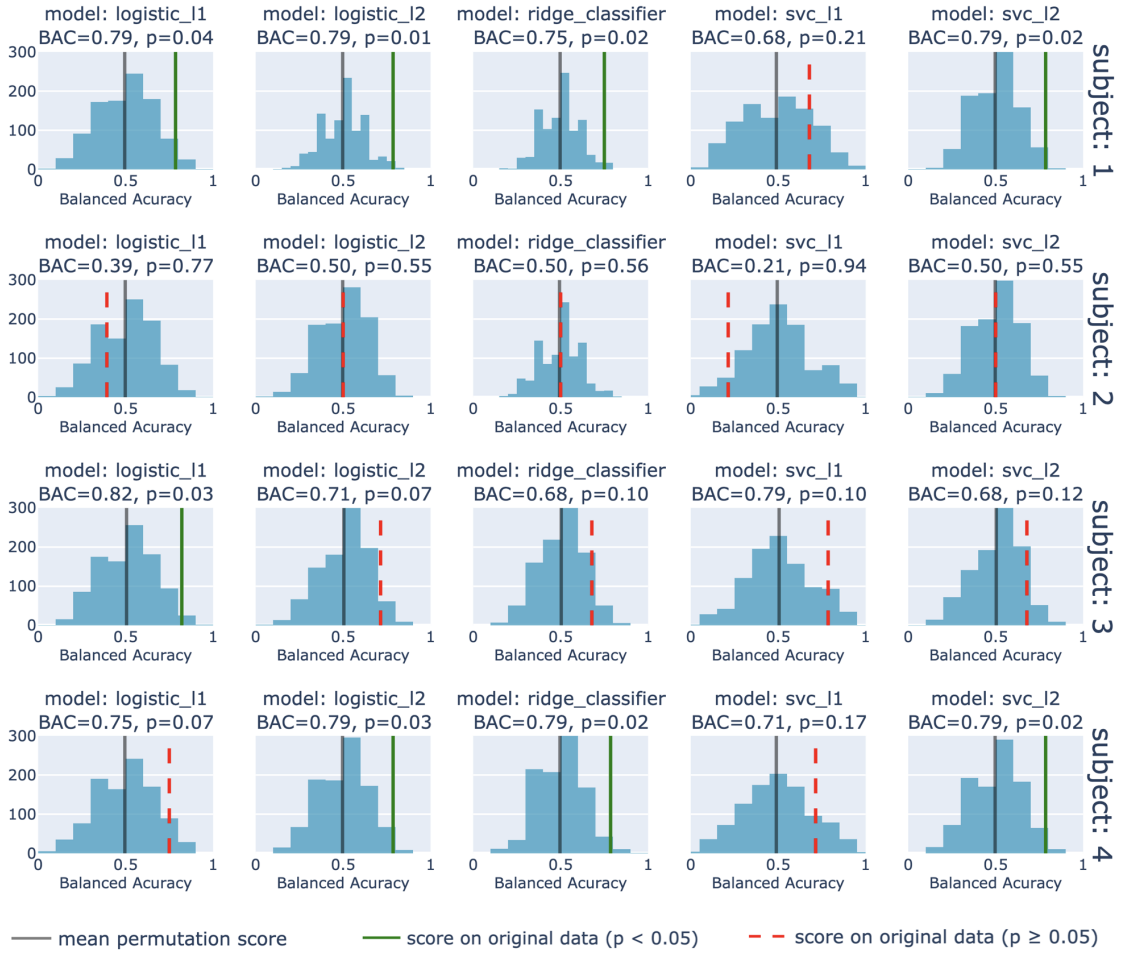


Figure 6.1.: Results of intra-subject MVPA for task prediction for subjects 1–4. Each histogram shows the results for a certain subject and model type.

6.1.2. Inter-Subject MVPA

Figure 6.2 shows the results of the inter-subject MVPA on the same data and target of interest. We can see that all five results are highly significant $(p < 0.001)$ and that the models perform well above chance level $68\% - 74\%$. This shows that the models can predict whether a participant is comprehending code or locating syntax errors solely based on brain imaging data. Furthermore, as this is an inter-subject MVPA, it shows that the differences between these two tasks are similar between participants.

The best performing classifier, achieving 74% balanced accuracy, seems to be the logistic regression with an L1 penalty. That is why, we choose to plot the weights of this particular model. We apply a cluster size threshold of 20 voxels to the weight map which can be seen in Figure 6.3. It shows that BA 11, 19, 37, 45, 46, 47 and 48 push

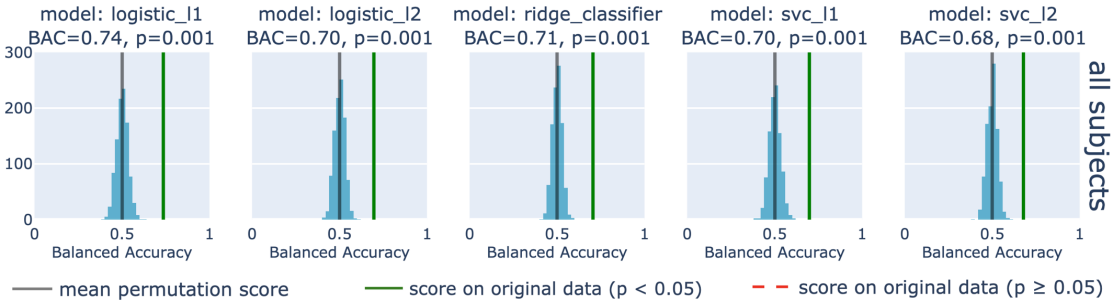


Figure 6.2.: Results of inter-subject MVPA for task prediction. Each histogram shows the results for a certain model type.

the classifier's decision towards code comprehension, while BA 6, 8, 18, 20, 21, 25 and 36 push the decision towards locating syntax errors.

> **Answer to RQ1:** We performed an intra- as as well as an inter-subject MVPA to classify the difference between the tasks "code comprehension" and "locating syntax errors" based solely on fMRI data. The inter-subject analysis shows that we can predict the task only for 6 out of 17 subjects in a significant manner. The inter-subject analysis shows that we can build five different models which predict the task in a significant manner (i.e., $68\% - 74\%$ balanced accuracy with $p < 0.001$) using the data from all subjects. Therefore, we conclude that we were able to predict different tasks from fMRI data on code comprehension with an inter-subject MVPA, but not with an intra-subject MVPA.

6.1.3. Discussion

Overall, the results of the inter-subject MVPA show a clear difference between the two tasks, while the results from the intra-subject MVPA vary depending on the subject and the model type. The difference between the two analysis types might be due to the different number of samples used. For training a single model during the intra-subject analysis we overall[1] use only 28 samples, while we use 16 times as much (i.e., 448) for training a single model during the inter-subject analysis. It is possible that if we had more samples per subject, we could also perform an intra-subject analysis successfully. Another fact that supports this hypothesis, is that in many instances within the intra-subject analysis we have models which perform way above chance level (e.g., subject 3), but are nevertheless not significant. It is, however, also possible that we have achieved these high scores by chance due to the small dataset size [19] and that there actually is no relationship between the input and the output variables.

[1] over all CV iterations

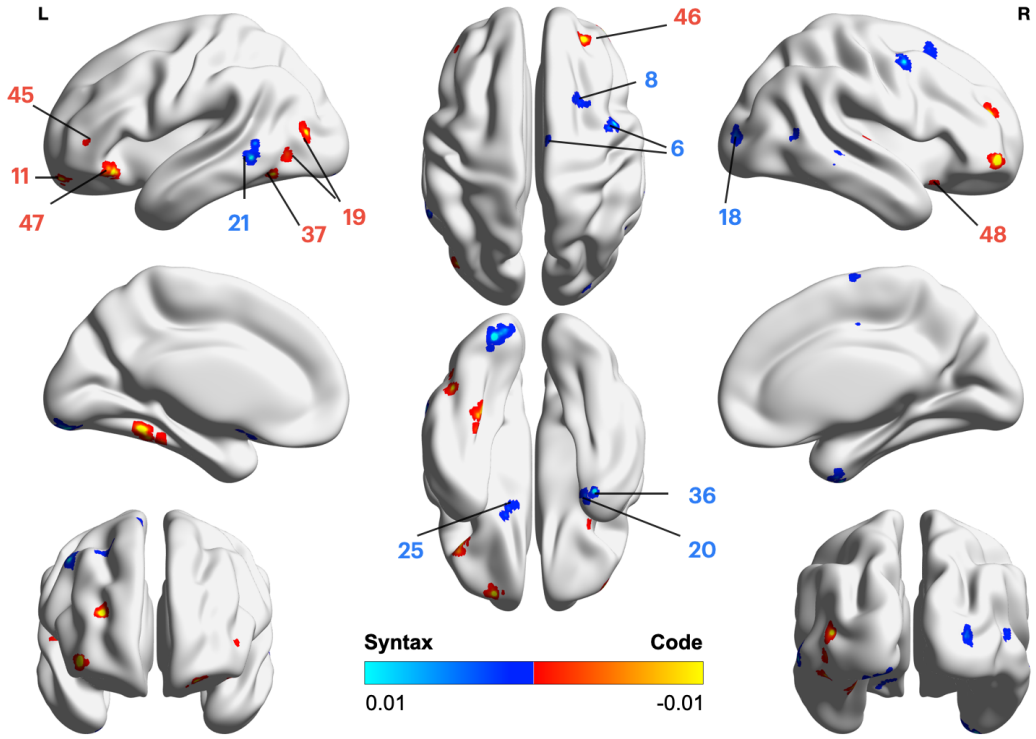


Figure 6.3.: Average weight map for logistic regression (L1 penalty) for task classification. Cluster size threshold $= 20$ voxels. Brain regions marked in "hot" colors push the decision of the classifier towards code comprehension, while the ones marked in "cold" colors push the decision towards locating syntax errors. Brain regions are labeled with the corresponding Brodmann areas that they belong to. Each BA is labeled only once, even if it can be seen multiple times on the figure from different perspectives.

For each analysis we have two sets of models where we have used the same model type (i.e., logistic regression and SVC) but with a different regularization technique (i.e., L1 and L2). The results of the intra-subject analysis do not suggest that one regularization technique is better than the other. The results of the inter-subject analysis, however, show that for both logistic regression and SVC the model with the L1 regularization performed slightly better than the one with the L2 regularization. This might be due to the fact that the models using L1 regularization set many of the predictors to zero, which minimizes the high dimensionality problem.

The underlying dataset stems from a study [4], where the same fMRI data was analyzed using a traditional GLM analysis. The results of this study show that the two tasks in question have distinct activation patterns. Furthermore, the authors also identified Brodmann areas which are associated with code comprehension. The results

of this thesis correspond only in part with the ones from Siegmund et al. - we have identified three of the five BAs identified by the authors (i.e., BAs 6, 21, 47). This result shows that the identified brain areas can differ depending on the method of analysis that is applied and is not surprising as GLM analysis either analyzes voxels individually or averages their signals out while MVPA analyzes voxels jointly without averaging them out [24].

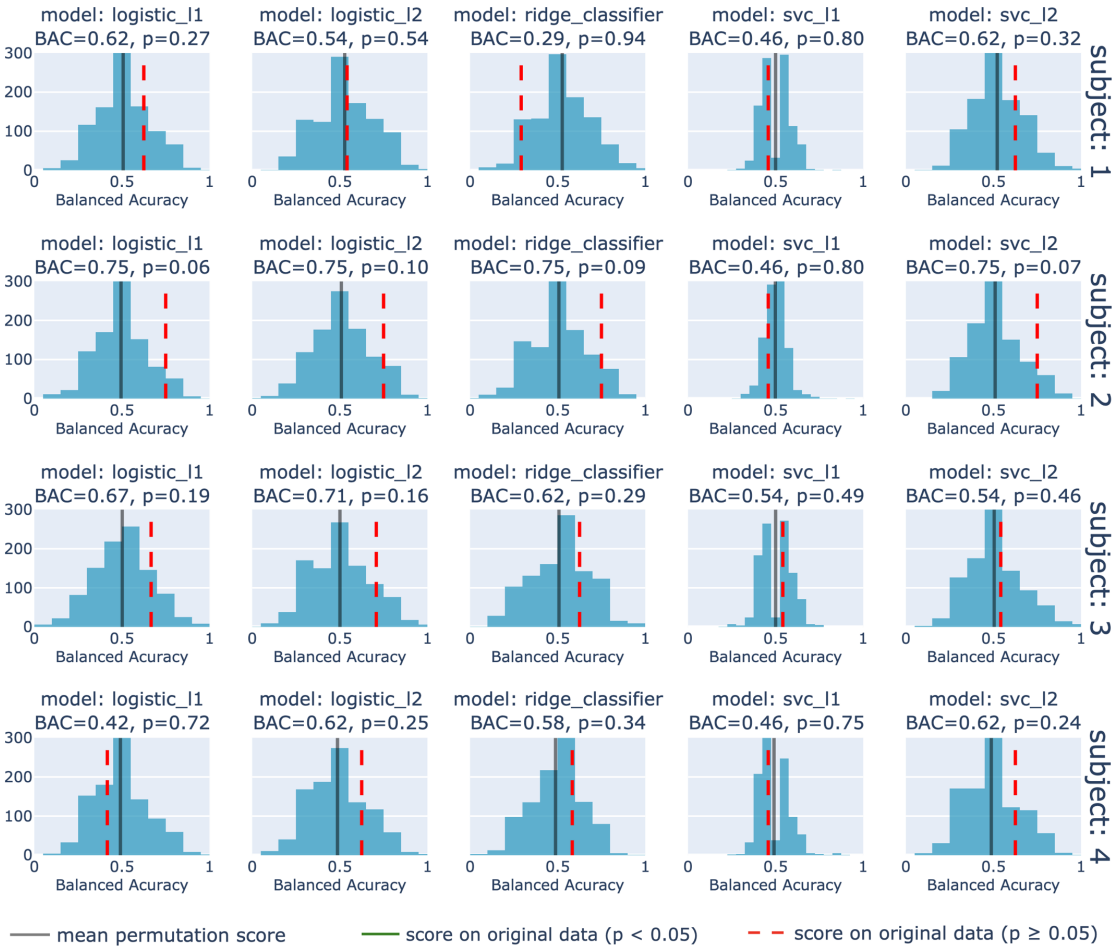


Figure 6.4.: Results of intra-subject MVPA for algorithm type prediction for subjects 1–4. Each histogram shows the results for a certain subject and model type.

6.2. Stimulus Attribute Prediction (RQ2)

Next, we take a look at the results for the target stimulus attribute, namely algorithm type and complexity metrics.

6.2.1. Intra-Subject MVPA

Algorithm Type

Figure 6.4 shows the results of the intra-subject MVPA for the algorithm type (i.e., recursion vs. iteration) for the first four participants. The results for all subjects can been seen in Appendix B. Overall, we have only one model which predicts the difference between recursion and iteration in a significant manner (subject 17, model logistic regression (L2)).

Complexity Metrics

The intra-subject MVPA for the four different complexity metrics did not produce any model with an R^2 score above 0, which means that none of the models explains any of the variance in the data. Therefore, we do not show all of the results for all four metrics but just an excerpt (see Figure 6.5). The remaining results for this part of our MVPA look similar.

6.2.2. Inter-Subject MVPA

Algorithm Type

The results for the inter-subject MVPA for the algorithm type can be seen in Figure 6.6. All models perform better than chance level, but only very slightly ($54-58\%$). The permutation tests show that only the ridge classifier performs significantly better than chance level.

Complexity Metrics

The results for the inter-subject MVPA for the four different complexity metrics can be seen in Figure 6.7. The R^2 score is very low for all models. For most of them it is even below 0, which means that the models explain the variance in the data even worse than a horizontal line. The permutation tests reveal that the only models with significant results are the ones for LOC ($p < 0.05$). However, here too, the model scores are very low ($R^2 = 0.03$), which means that they explain only very little of the variance in the data.

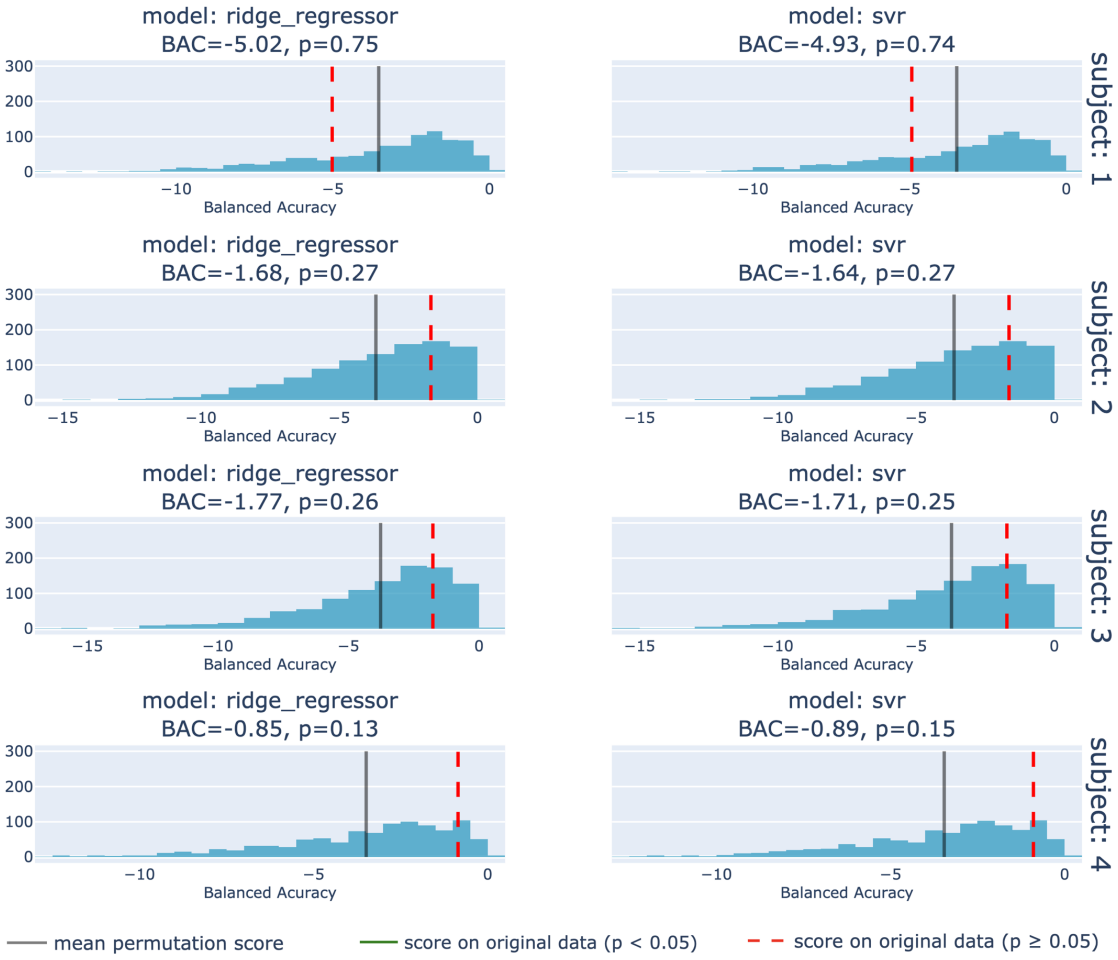


Figure 6.5.: Results of intra-subject MVPA for complexity metric (McCabe) prediction for subjects 1–4. Each histogram shows the results for a certain subject and model type.

Answer to RQ2: We performed and intra- as well as an inter-subject MVPA for the targets algorithm type and four different complexity metrics. The results of all intra-subject analyses for all targets were insignificant. This lets us conclude that we cannot use an intra-subject MVPA to predict the algorithm type or the complexity metrics using the datasets made available to us. The inter-subject MVPA for the algorithm type (iteration vs. recursion) attribute of the stimulus shows that we can use MVPA to train a model which better than chance level, however, not by much ($BAC = 58\%$). The results for the complexity metrics suggest that it is not possible to predict any of the four selected complexity metrics with an inter-subject MVPA using the fMRI data made available to us.

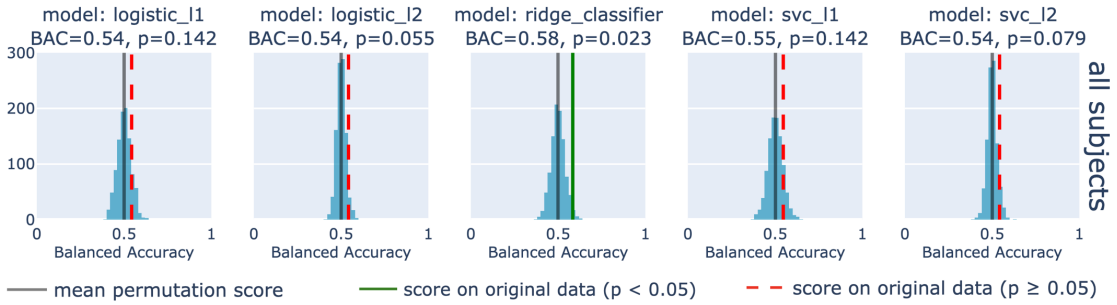


Figure 6.6.: Results of inter-subject MVPA for algorithm type prediction. Each histogram shows the results for a certain model type.

6.2.3. Discussion

In the case of the algorithm type as a target, we can again see the same phenomenon as with the intra-subject MVPA for RQ1. For many of the subjects we have rather high BAC scores, however, the results are still not significant (e.g., subject 2, all models except SVC with L1 penalty). Again, this could be due to the small dataset sizes used for the intra-subject MVPA. In this case, we have even less samples than for the intra-subject MVPA for task prediction - overall only 13 samples per participant. However, as the results of the inter-subject MVPA for the algorithm type, where we have 221 samples overall, were not as promising either, it is very well possible that there is not a large difference between the neural representations of recursion and iteration.

To the best of our knowledge, no other study has yet tried to classify the difference between these two algorithm types. However, there have been similar studies. Ikutani et al. have performed an MVPA, where they were able to decode the category of code (i.e., math, search, sort, string) [2]. Yet, one could argue that the subjective difference between these code categories is higher than between the two algorithm types we focus on. Further, Ikutani et al. used 216 instead of 13 samples per participant.

A study where the target was more similar to "algorithm type" was performed by Srikant et al. It shows that it is possible to decode the control flow of a code snippet (i.e., iteration, conditionals, sequential instructions) [3]. In contrast to our approach, the authors did not perform a whole-brain analysis but used preselected brain areas (e.g., multiple demand and language system) and had 48 samples per participant.

Our two analyses for the different complexity metrics suggest that it is not possible to decode complexity metrics at all using MVPA. This result coincides with the findings of Peitek et al. [11], who analyzed the same data with a traditional GLM approach and found that the four complexity metrics show only weak to medium correlations with programmer's correctness and response time. The hypothesis that the authors pose, that is, that complexity metrics do not reliably reflect the difficulty of comprehending code, could also explain our negative results for this target.

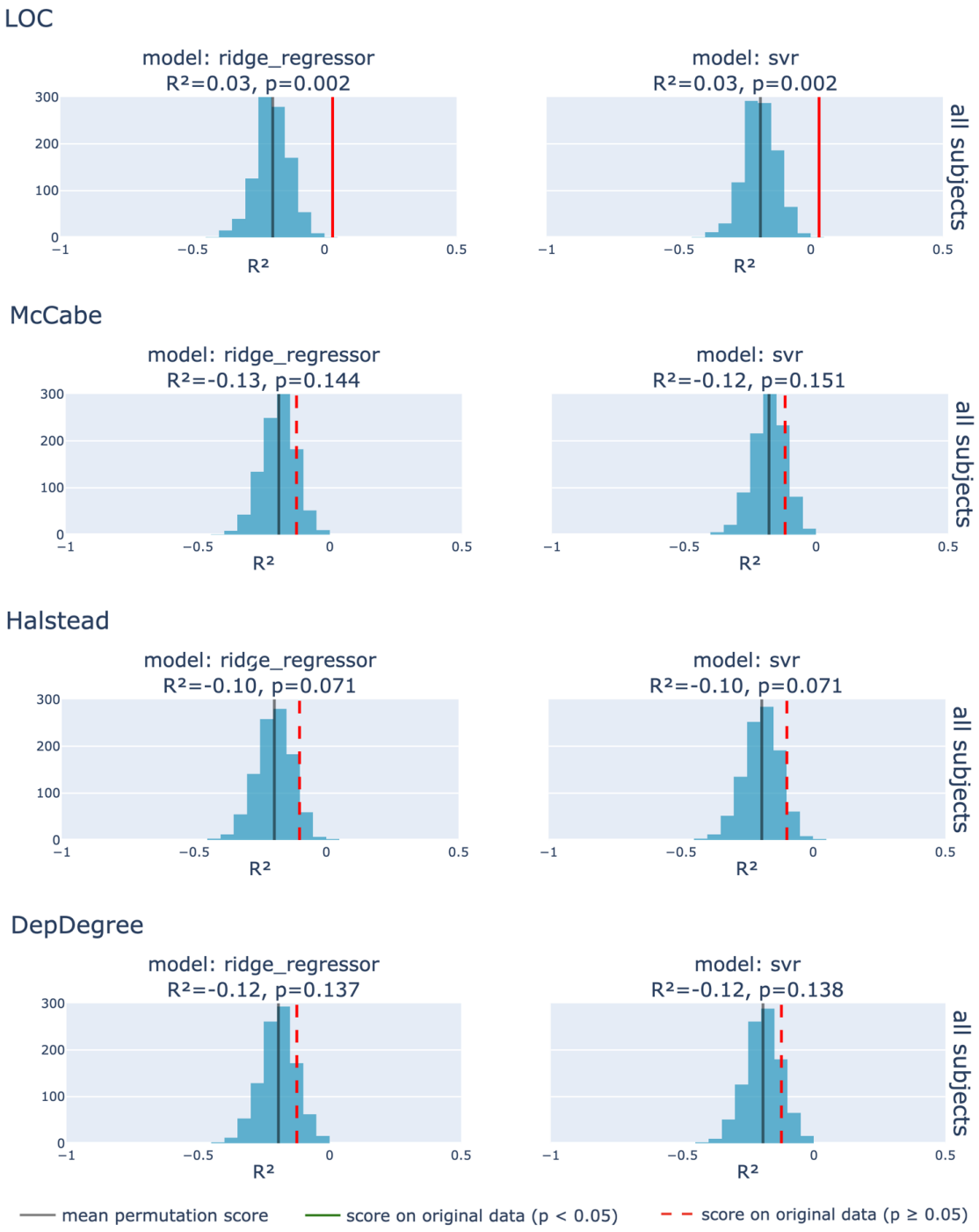


Figure 6.7.: Results of inter-subject MVPA for complexity metrics prediction. Each histogram shows the results for a certain model type.

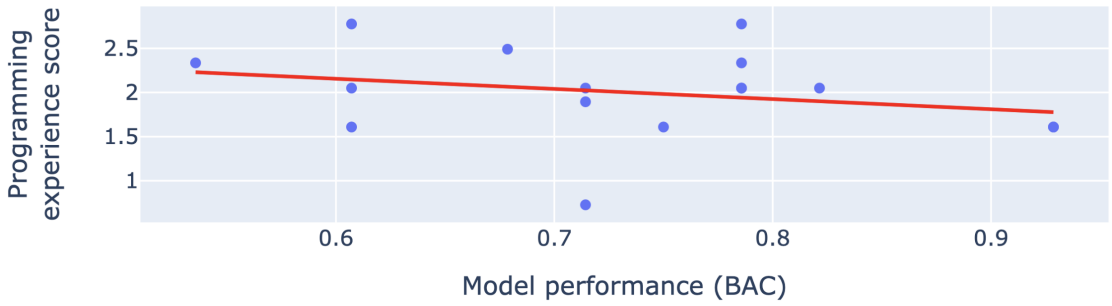


Figure 6.8.: Relationship between model performance and experience score (r=-0.25, p=0.35). The model is a logistic regression (L1 penalty) trained to classify the task a subject is performing (code comprehension vs. locating syntax errors).

6.3. Participant Attribute Prediction (RQ3)

Finally, we try to predict a participant attribute, namely the experience score.

6.3.1. Approach No.1

In this section, we investigate whether a model can classify the task a subject is performing more accurately, if this subject is more experienced (or less experienced). The results can be seen in Figure 6.8 and show that there is no significant relationship between the model performance and the experience score $(p = 0.35)$. This result suggest that the difference between the tasks code comprehension and locating syntax errors is similar for experienced and inexperienced programmers.

6.3.2. Approach No.2

Finally, we investigate whether models that classify the algorithm type and are trained on a group of more experienced programmers, perform better than the same models trained on a group of less experienced programmers.

The results show that the models trained on more experienced programmers perform better (see Figure 6.9). The experienced group achieves a higher BAC scores for all five different model types, but only three of the results are significant.

> **Answer to RQ3:** We were not able to predict the experience of a subject using MVPA on the datasets made available to us. However, we showed that the level of experience of the programmer does make a difference when trying to predict the algorithm type of a stimulus.

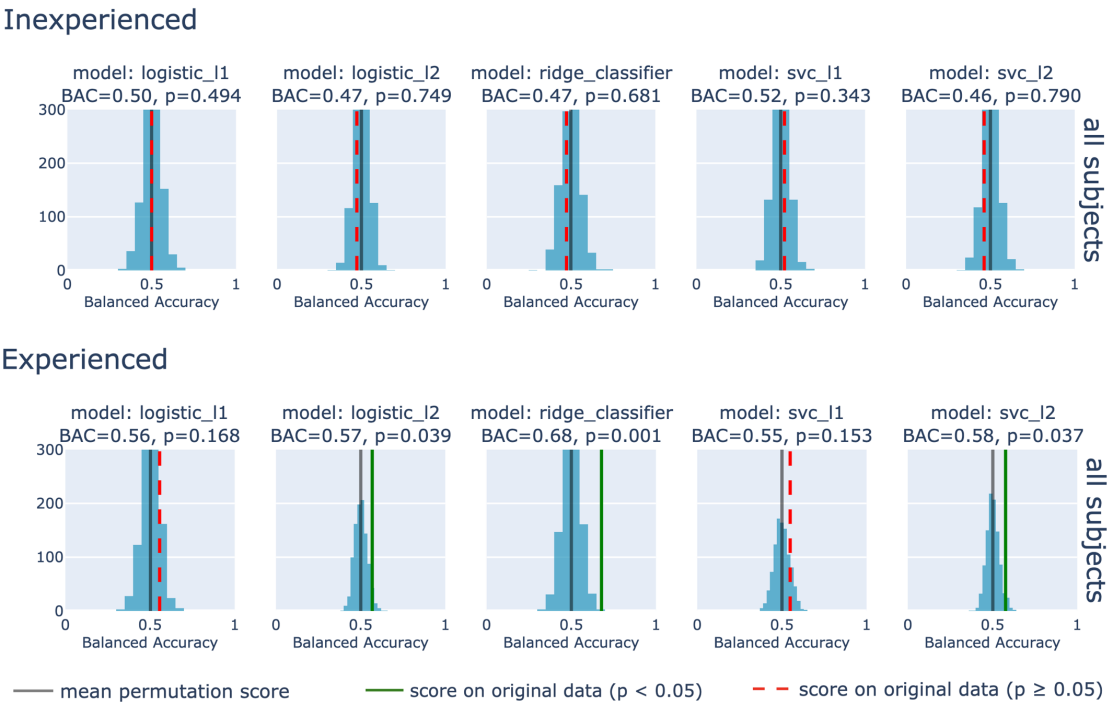


Figure 6.9.: Results of inter-subject MVPA on two separate groups of programmers (experienced vs. inexperienced) for target algorithm type (iteration vs. recursion).

6.3.3. Discussion

As already mentioned, predicting participant attributes is the most difficult task as we only have very few samples. Therefore, we believe that future studies which investigate such targets should be performed on much larger datasets. If participant attribute prediction is the goal, not only more trials per participant but also more participants are needed. Furthermore, none of the datasets available to us had a participant sample with very high differences between the participants' experience. If the target of interest is a participant attribute, future research should select a participant pool with a higher variability for the target of interest.

To the best of our knowledge, no study has yet focused on experience as the target of interest. However, two previous studies have already used MVPA to show that expertise matters and one could argue that expertise and experience are highly related. Floyd et al. have found that expert programmers treat code more similar to prose [1], while Ikutani et. al have shown that models decode the category of code with a higher accuracy for expert programmers [2]. Our results coincide with these findings, showing that a model can predict the algorithm type of code with a higher accuracy for expert programmers.

7

Limitations & Threats to Validity

There are some limitations of our work which we have to take into account and discuss.

7.1. Internal

A limitation that arises in all fMRI studies on code comprehension that have been performed so far is that the code snippets are comparatively small. Therefore, they can represent average computer programs only to a certain degree. This is because the screen on which the snippets are presented in an fMRI scanner is rather small. Similarly, the type of code snippets used in this thesis do not represent all possible, or common, code code snippets, as we only have a very limited number of different snippets per dataset.

Another threat to the internal validity is that, since the datasets were not collected specifically for this thesis, it was not possible to keep all other variables constant when predicting a certain variable of interest. Therefore, it is possible that other variables such as participant characteristic differences have influenced our models.

Further, it is possible that using different machine learning models could yield different results than the ones presented in this thesis. However, we have not used one but several ML models and we have chosen them because they are commonly used in this field and favorable for fMRI data. Therefore, we do not expect other models to perform better. Similarly, using different feature selection techniques could result in a different set of predictors/voxels being used and eventually lead to different results.

Finally, it is possible that the implementation of the approach described in this thesis contains bugs. We have tried to mitigate this problem by working with commonly used and tested libraries.

7.2. External

One limitation for our external validity is that we do not have a representative group of subjects as most of them are male students. This for example does not allow us to draw conclusions about female programmers.

Furthermore, the studies that we have used in this thesis were performed in special environments (i.e., fMRI scanner) and not in a normal setting for code comprehension. Programmers could potentially behave differently when comprehending code in a more natural environment.

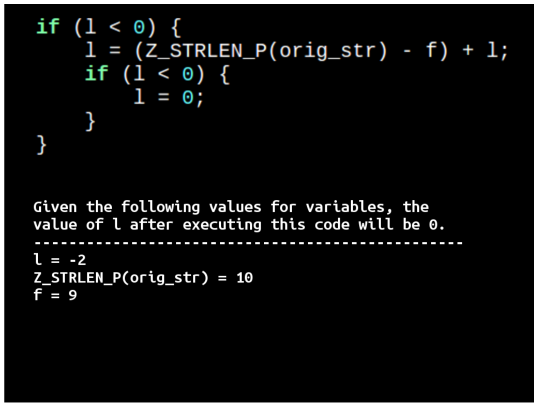
8

Related Work

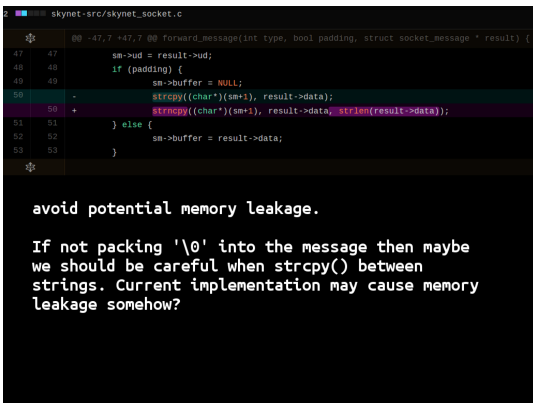
In this chapter we will summarize the work related to this thesis. Publications which use traditional GLM analysis to study fMRI data on code comprehension were already introduced in Chapter 4. Therefore ,in the following, we will only focus on publications where fMRI data on code comprehension was analyzed using MVPA.

8.1. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise

Floyd et al. were the first to decode information from fMRI data on code comprehension [1]. They explored the relationship between code review, natural language, and programmer expertise. The experiment involved three tasks: code comprehension, code review, and prose review (see Figure 8). To identify the underlying patterns from the collected fMRI data, one GLM was estimated for each trial. The resulting beta images were then used to train a Gaussian Process Classification model. The goal of the model was to predict the underlying task type. The model performance was assessed using a LORO-CV. The above described steps were completed for each of the three scenarios of interest, i.e., Code Review vs. Prose Review, Code Comprehension vs. Prose Review, and Code Review vs. Code Comprehension. The best performing classifier (code comprehension vs. prose review at 79% balanced accuracy) shows that our brain processes programming languages in a different way than natural languages. The authors also computed the correlation between expertise and model



(a) Code Comprehension



(b) Code Review

The study revealed that the conditions of a cat's teeth, eyes, and fur are good ~~indices~~ indexes of the cat's health. ~~Importantly,~~Note that the study only considered male cats, so these results are not necessarily generalizable. However, another independent study shows that females with these characteristics live longer ~~like~~ than the males do.

(c) Prose Review

Figure 8.1.: Different types of task stimuli [1]

performance. They found that the lower the performance of a classifier for a certain programmer is, the higher their expertise is. Thus, Floyd et al. conclude that experts treat programming languages more like natural languages.

8.2. Expert Programmers Have Fine-Tuned Cortical Representations of Source Code

In a more recent work [2], Ikutani et al. focus on decoding programming expertise from fMRI data. To this end, they conducted a study with programmers with different levels of expertise, measured using programmer's ratings in competitive programming contests. During the experiment the subjects were presented with stimuli in the form of code snippets implementing well-known algorithms from four different functional

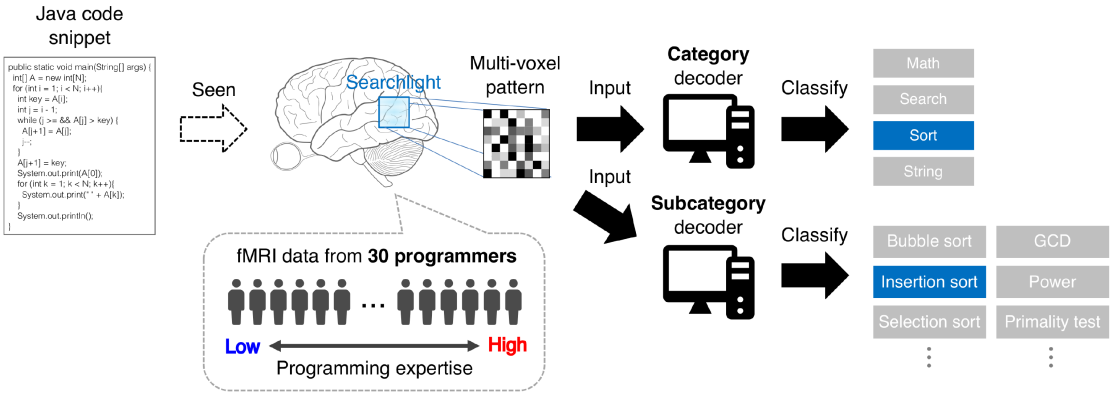


Figure 8.2.: Overview of decoding framework for category and subcategory of code [2]

categories. An overview of the work can be seen in Figure 8.2. Similarly to the work of Floyd et al. [1], the collected data was first preprocessed using GLM-based feature extraction. The resulting beta images were then used to train and evaluate a linear-kernel SVM for each subject separately using LORO-CV. The resulting models were able to decode the different functional categories of code snippets from brain activity. A whole-brain searchlight analysis [33] was employed to identify where significant decoding accuracies exist. Furthermore, the authors show that there is a high association between programming expertise and decoding accuracies. Ikutani et al. found that programmer's outstanding performances depend on fine-tuned cortical representations of source code.

8.3. Convergent Representations of Computer Programs in Human and Artificial Neural Networks

Finally, the most recent publication related to this thesis is from Srikant et al. [3]. The work of the authors is divided in two main parts (see Figure 8.3). First they use MVPA to predict several code properties (e.g., control flow, data type, static, and dynamic analysis). The authors use linear models with L2 penalty and a LORO-CV training procedure. However, they do not perform their analysis on the whole brain, but instead on preselected brain areas, such as the multiple demand system and the language system, training separate models for each brain region.

Second, Srikankt et al. examine how similar ML representations and brain imaging representation of programs are. This is done by first producing ML representations of the code snippets by a suite of ML models trained on code. Afterwards, another set of models is used to predict these ML representations solely using the correspond-

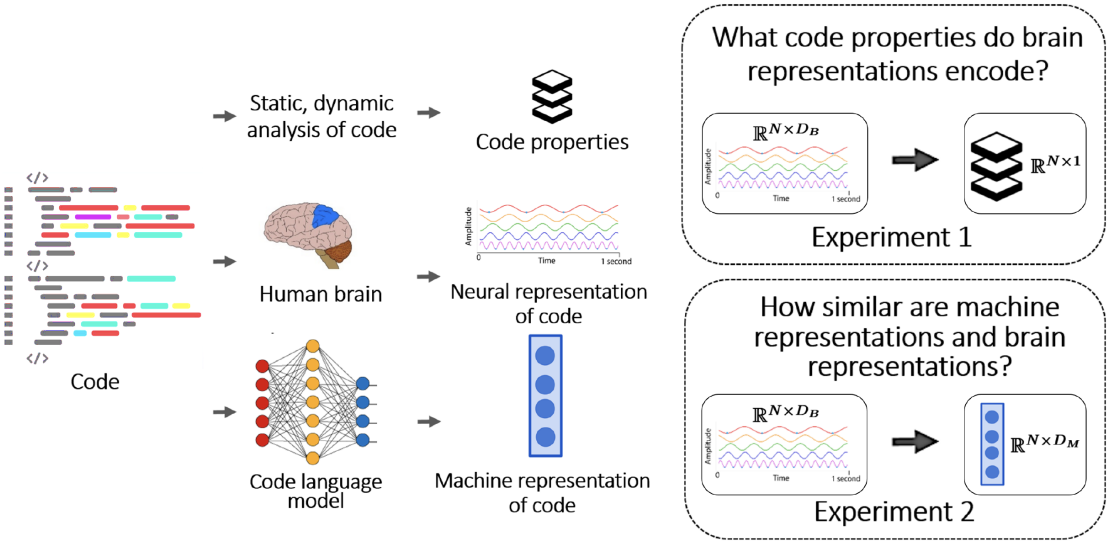


Figure 8.3.: Overview of approach to compare representations of computer programs in human and artificial neural networks [3]

ing brain imaging representations of the code snippets. The results from part one of the experiment show that the multiple demand system encodes dynamic code properties, while the language system encodes syntactic code properties. Further, part two of the experiment shows that the neural representations of code align with the ML representations of code.

9

Future Work

As already mentioned, so far there are only very few studies on code comprehension which use MVPA methods. Therefore, there is much more research that can be done in this area. In this chapter, we describe some possible future work.

9.1. Dataset Size

We have seen in our results that models often performed well and were nevertheless often not significant. This could be because of our small dataset sizes, which do not allow us to train the models with a sufficient number of samples. Similar studies used much larger datasets (e.g., 30 subjects with 216 samples each [2]). However, as fMRI studies are very costly, it would be beneficial to investigate what a sufficient amount of data for this areas is. How many participants are needed? How many trials/runs per participant are needed? How do these values depend on the target of interest?

9.2. Targets of Interest

As already mentioned in previous chapters, there are many more targets of interest that could be addressed in future analyses. Targets of interest related to the stimulus that have not been explored yet using MVPA could, for example, be different presentation types of code. How can we present code in a way that is most easiest to comprehend? Further, there are also many targets of interest related to the participant which have not been explored yet, such as the gender, the age or the education type.

9.3. Searchlight Analysis

One MVPA method for identifying informative brain areas which we could not use in thesis because of its computational expensiveness, is searchlight analysis [33]. This method iteratively selects a cluster of neighboring voxels (so-called "searchlights") and uses only those to predict a certain target of interest. The algorithm does this as many times as necessary until either the whole brain or the whole region of interest is covered. The result is a brain map showing the prediction accuracies for each searchlight center. This method has the advantage that it addresses the high dimensionality problem of the data during the training procedure [34] instead of beforehand like the feature selection method used in this thesis (i.e., ANOVA). Ikutani et. al use searchlight analysis to predict different code categories. In future research, this method could be used to predict other relevant targets of interest in the code comprehension scenario.

9.4. Representational Similarity Analysis

Next to decoding analysis, representational similarity analysis (RSA) [35] is another popular MVPA method for the analysis of fMRI data. With the help of RSA one can examine the similarity between different neural responses. This is done by computing a dissimilarity matrix of the properties of the target of interest (e.g., properties of a stimulus) as well as a dissimilarity matrix of the neural images corresponding to them. Finally, the two matrices are compared to each other using a correlation metric in order to examine whether the brain encodes the same properties of a target of interest as we believe it to. RSA has not been used in the domain of code comprehension yet, although it could potentially reveal more information about how regions of the brain represent information. In contrast to MVPA, RSA allows to compare targets of interest across multiple different dimensions simultaneously instead of only on a single category level [36].

10

Conclusion

Scientists agree that a deeper understanding of how programmers comprehend code, could help us improve the education process of future programmers, programming tools, and languages, and based on that the quality of newly generated code overall.

One way to improve our understanding of code comprehension is to "look" into the brain of programmers, that is to analyze brain imaging (i.e., fMRI) data. Due to the high cost of fMRI studies, so far, there have been only very few studies on code comprehension and even fewer studies which use the machine learning based multivoxel pattern analysis (MVPA) as an analysis method for the task. This rather new analysis method, however, is more powerful than traditional approaches, as it allows us to decode information from the brain by analyzing the underlying activation patterns.

We perform MVPA on data from existing fMRI studies, which have only been analyzed using traditional GLM-based methods in order to gain additional knowledge. We implement and evaluate several MVPA decoding pipelines which predict different targets of interest based solely on fMRI data.

Our results show that we are able to build classifiers which can predict whether a programmer is comprehending code or locating syntax errors. Furthermore, we were also able to build a classifier which predicts whether a programmer is looking at a code snippet with a recursion based or iteration based algorithm. Finally, we find that experience matters - a classifier which predicts the algorithm type performs better if trained and tested on more experienced programmers. We were, however, not able to decode the complexity of a code snippet or the experience level of a programmer.

Although, we were able to build well performing models for some of our targets of interest, our study showed that the datasets we use within this thesis are not optimal

for MVPA, mostly due to their small size. Future research in this area should be performed on datasets with more samples in order to be able to rule out the dataset size as a potential reason for the lack of significant results. Finally, we find that if a small dataset is used, an inter-subject analysis is more likely to deliver positive results than an intra-subject analysis.

A

Complete Results of Task Prediction

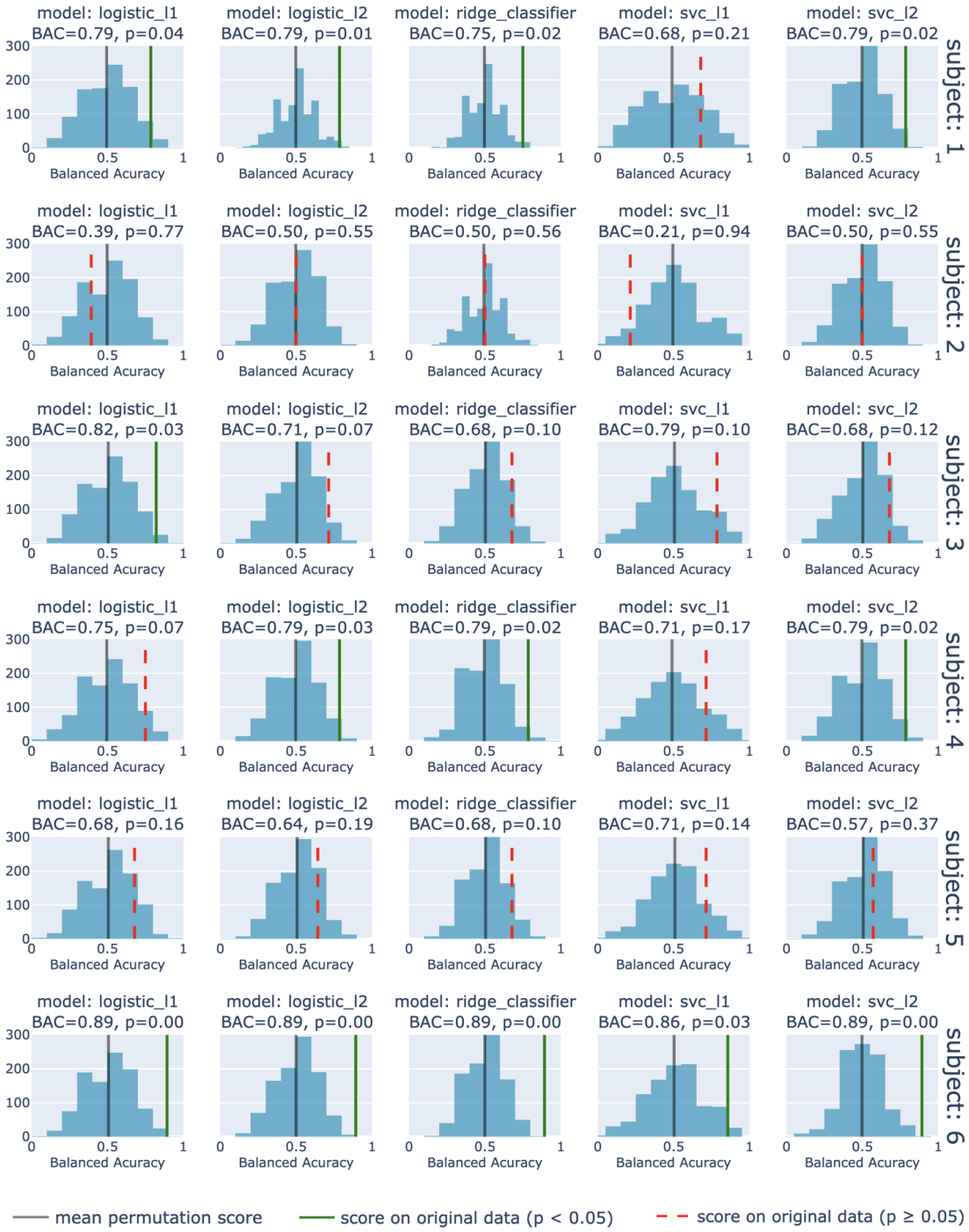


Figure A.1.: Results of intra-subject analysis for task prediction for subjects 1–6.

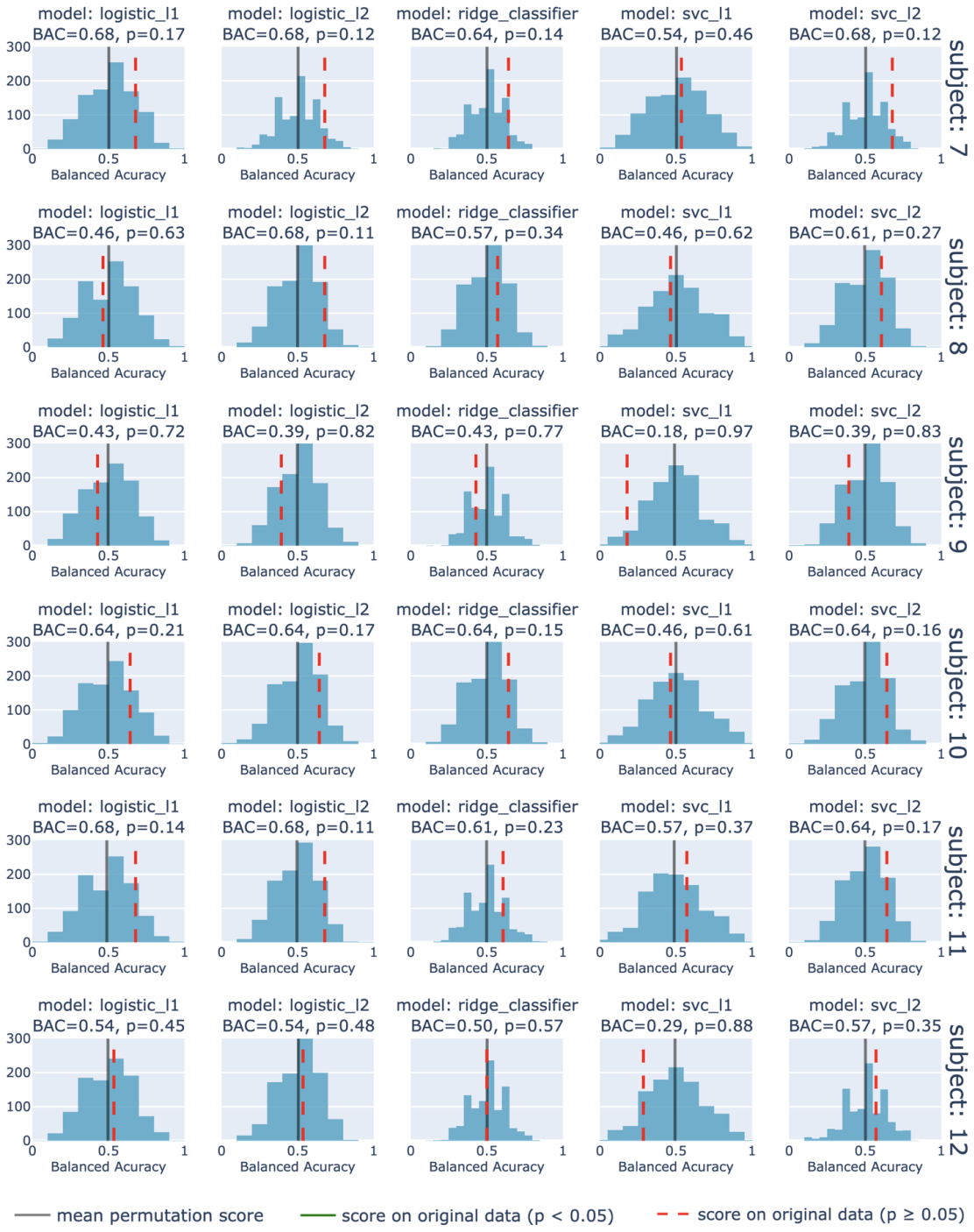


Figure A.2.: Results of intra-subject analysis for task prediction for subjects 7–12.

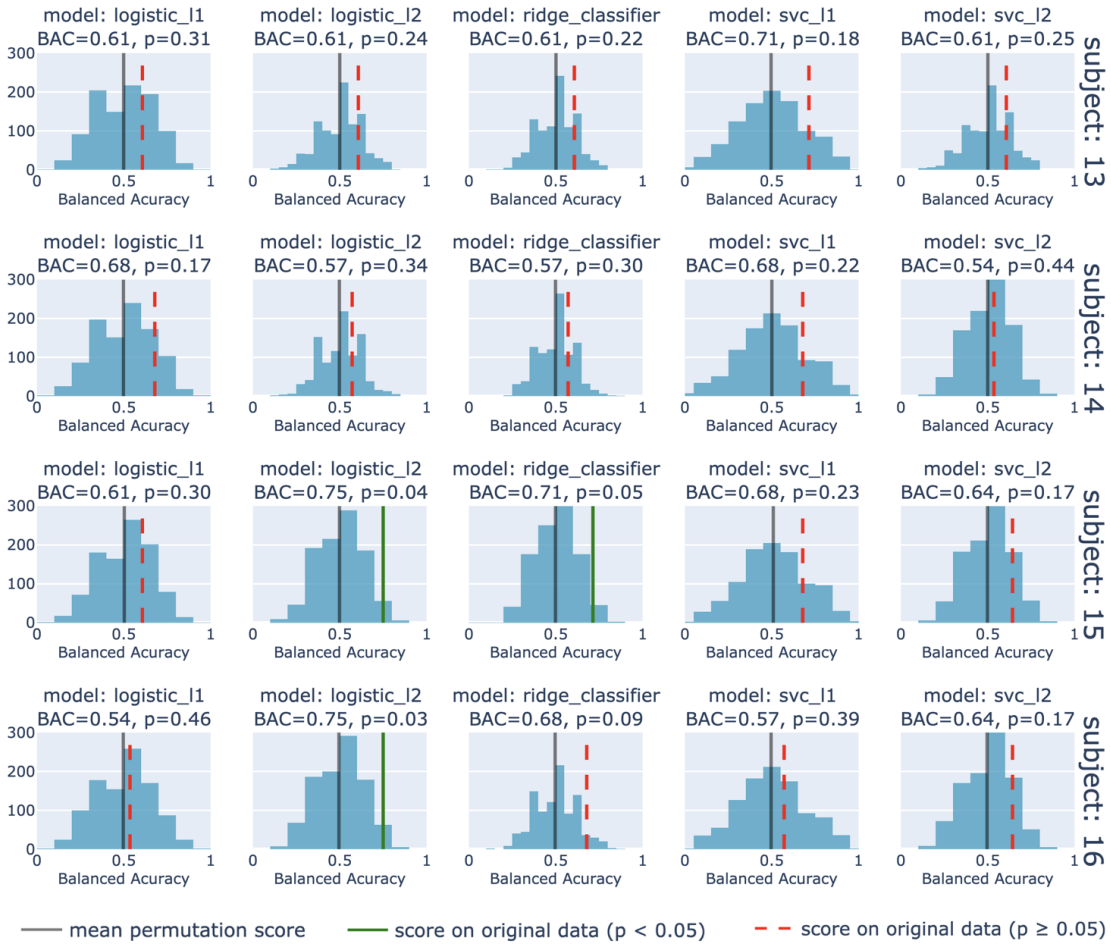


Figure A.3.: Results of intra-subject analysis for task prediction for subjects 12–16.

B

Complete Results of Algorithm Type Prediction

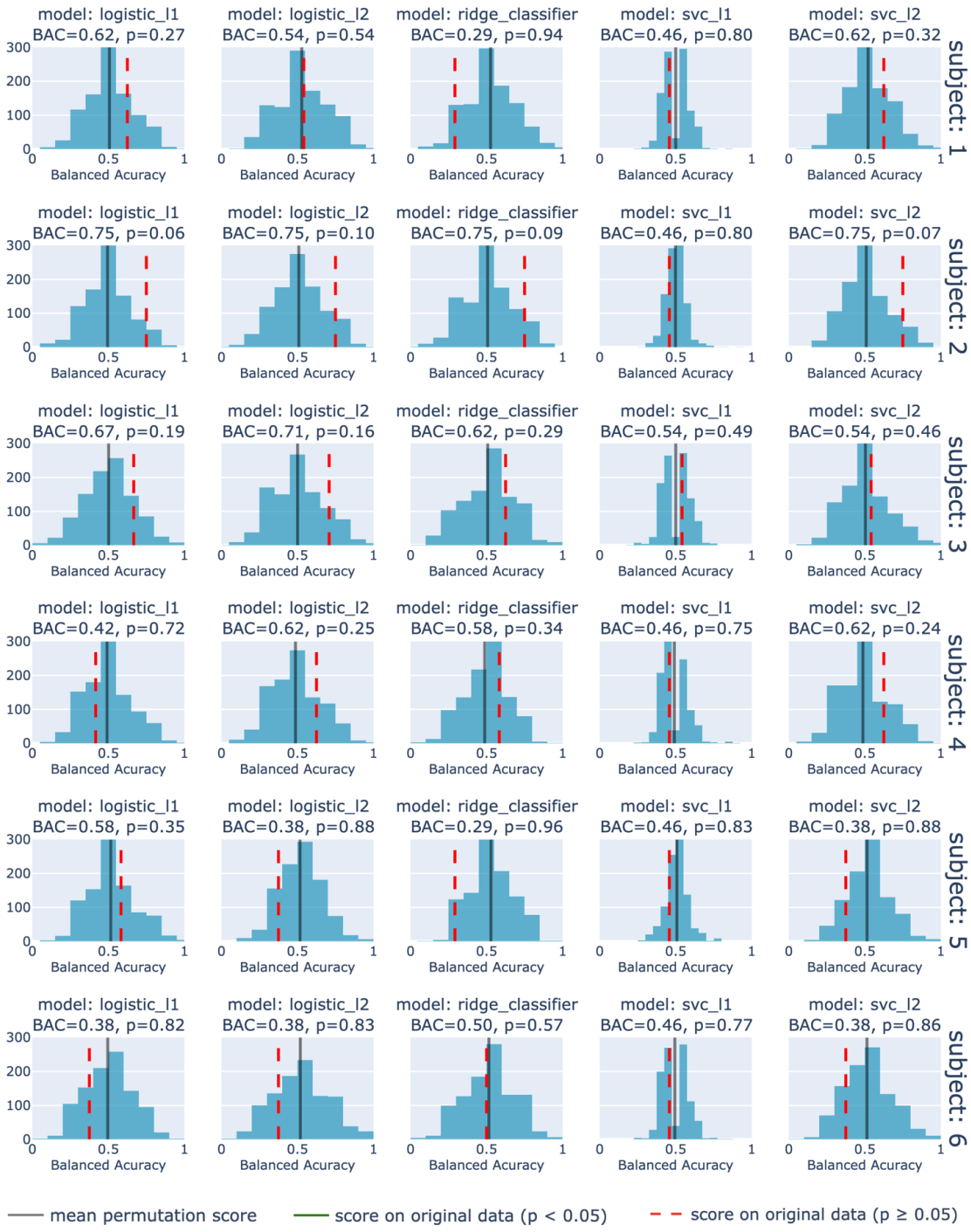


Figure B.1.: Results of intra-subject analysis for algorithm prediction for subjects 1–6.

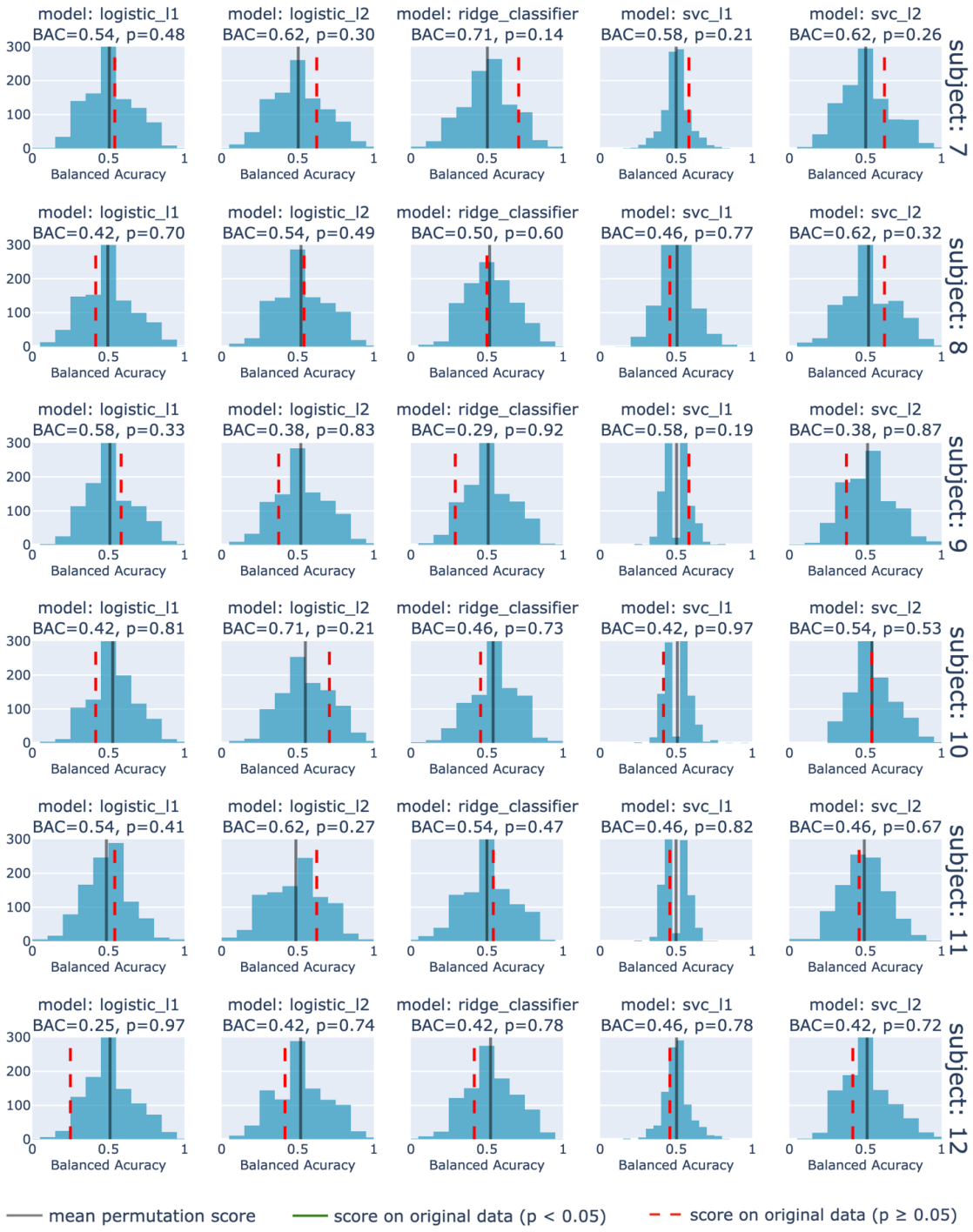


Figure B.2.: Results of intra-subject analysis for algorithm prediction for subjects 7–12.

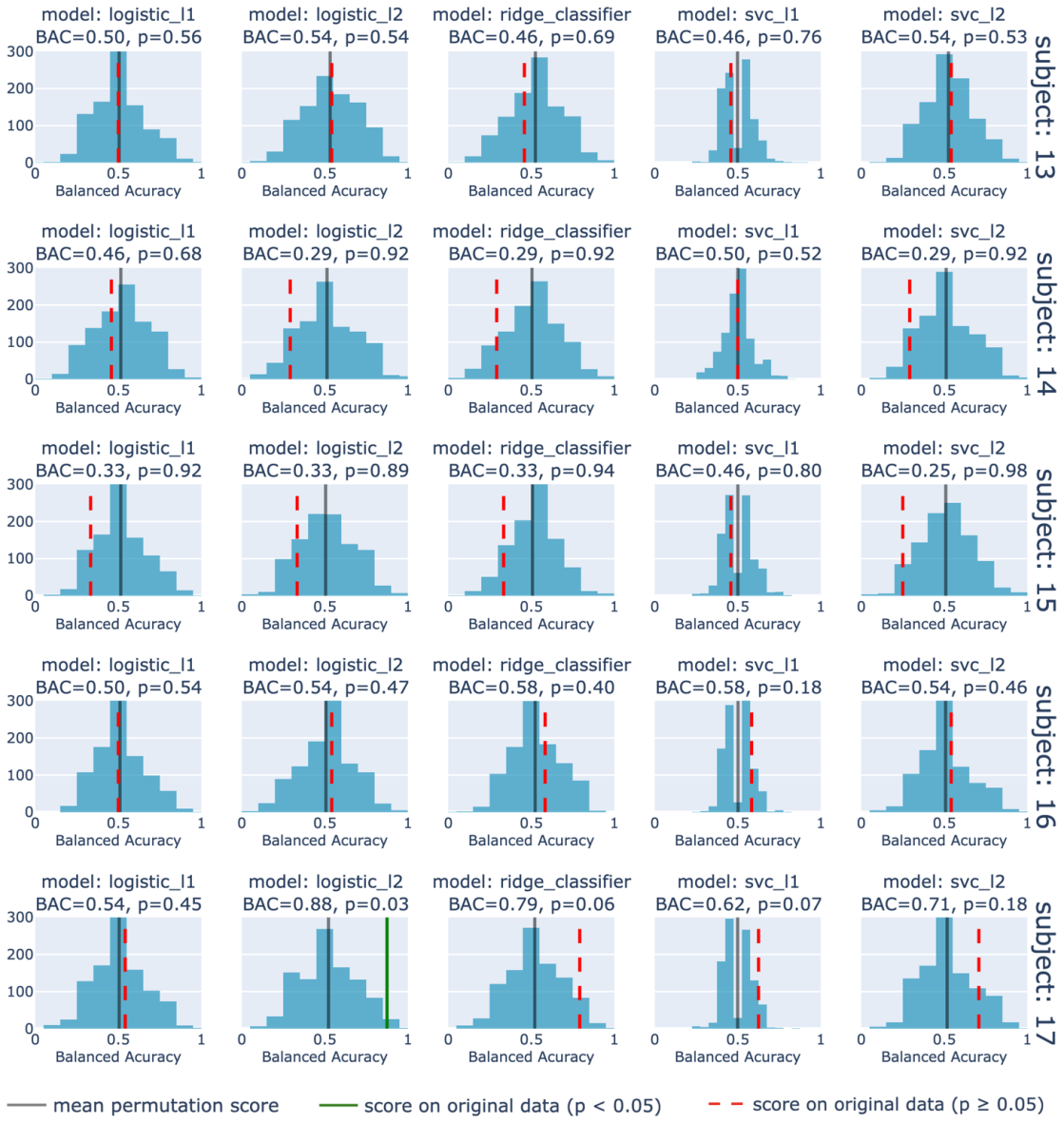


Figure B.3.: Results of intra-subject analysis for algorithm prediction for subjects 12–17.

Bibliography

[1] Benjamin Floyd, Tyler Santander, and Westley Weimer. Decoding the representation of code in the brain: An fmri study of code review and expertise. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 175–186, 2017.

[2] Yoshiharu Ikutani, Takatomi Kubo, Satoshi Nishida, Hideaki Hata, Kenichi Matsumoto, Kazushi Ikeda, and Shinji Nishimoto. Expert programmers have fine-tuned cortical representations of source code. *eNeuro*, 8(1), 2021.

[3] Shashank Srikant, Ben Lipkin, Anna A. Ivanova, Evelina Fedorenko, and Una-May O'Reilly. Convergent representations of computer programs in human and artificial neural networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[4] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 378–389, New York, NY, USA, 2014. Association for Computing Machinery.

[5] Kenneth A. Norman, Sean M. Polyn, Greg J. Detre, and James V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fmri data. *Trends in Cognitive Sciences*, 10(9):424–430, 2006.

[6] Abdelhak Mahmoudi, Sylvain Takerkart, Fakhita Regragui, Driss Boussaoud, and Andrea Brovelli. Multivoxel pattern analysis for fmri data: A review. *Computational and Mathematical Methods in Medicine*, 2021, 2021.

[7] James V. Haxby, M. Ida Gobbini, Maura L. Furey, Alumit Ishai, Jennifer L. Schouten, and Pietro Pietrini. Distributed and overlapping representations of faces and objects in ventral temporal cortex. *Science*, 293(5539):2425–2430, 2001.

[8] Tom M. Mitchell, Rebecca Hutchinson, Radu S. Niculescu, Francisco Pereira, Xuerui Wang, Marcel Just, and Sharlene Newman. Learning to decode cognitive states from brain images. *Mach. Learn.*, 57(1-2):145–175, oct 2004.

[9] Yukiyasu Kamitani and Frank Tong. Decoding seen and attended motion directions from activity in the human visual cortex. *Current Biology*, 16(11):1096–1102, 2006.

[10] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. Measuring neural efficiency of program comprehension. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 140–150, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Norman Peitek, Janet Siegmund, Chris Parnin, Sven Apel, Johannes C. Hofmeister, and André Brechmann. Simultaneous measurement of program comprehension with fmri and eye tracking: A case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '18, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, and Janet Siegmund. Program comprehension and code complexity metrics: An fmri study. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 524–536, 2021.

[13] Scott A. Huettel, Allen W. Song, Gregory McCarthy, et al. *Functional magnetic resonance imaging*, volume 1. Sinauer Associates Sunderland, 2004.

[14] Jeanette A. Mumford, Benjamin O. Turner, F. Gregory Ashby, and Russell A. Poldrack. Deconvolving bold activation in event-related designs for multivoxel pattern classification analyses. *NeuroImage*, 59:2636–2643, 2012.

[15] Wenyan Xu, Qing Li, Xingyu Liu, Zonglei Zhen, and Xia Wu. Comparison of feature selection methods based on discrimination and reliability for fmri decoding analysis. *Journal of Neuroscience Methods*, 335:108567, 2020.

[16] R.A. Fisher. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

[17] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013.

[18] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition*, pages 3121–3124, 2010.

[19] Etienne Combrisson and Karim Jerbi. Exceeding chance level by chance: The caveat of theoretical chance levels in brain signal classification and statistical assessment of decoding accuracy. *Journal of neuroscience methods*, 250, 01 2015.

[20] Giancarlo Valente, Agustin Lage Castellanos, Lars Hausfeld, Federico De Martino, and Elia Formisano. Cross-validation and permutations in mvpa: Validity of permutation strategies and power of cross-validation schemes. *NeuroImage*, 238:118145, 2021.

[21] Kâmil Uğurbil. Development of functional imaging in the human brain (fmri) the university of minnesota experience. *Neuroimage*, 62(2):613–619, 02 2012.

[22] S. Ogawa, T. M. Lee, A. R. Kay, and D. W. Tank. Brain magnetic resonance imaging with contrast dependent on blood oxygenation. *Proceedings of the National Academy of Sciences of the United States of America*, 87(24):9868–9872, 1990.

[23] Martin Monti. Statistical analysis of fmri time-series: A critical review of the glm approach. *Frontiers in human neuroscience*, 5:28, 03 2011.

[24] Jonathan D. Cohen, Nathaniel D. Daw, Barbara Engelhardt, Uri Hasson, Kai Li, Yael Niv, Kenneth A. Norman, Jonathan W. Pillow, Peter J. Ramadge, Nicholas B. Turk-Browne, and Theodore L. Willke. Computational approaches to fmri analysis. *Nature Neuroscience*, 20(3):304–313, 2017.

[25] Miriam E. Weaverdyck, Matthew D. Lieberman, and Carolyn Parkinson. Tools of the Trade Multivoxel pattern analysis in fMRI: a practical introduction for social and affective neuroscientists. *Social Cognitive and Affective Neuroscience*, 15(4):487–509, 04 2020.

[26] Qi Wang, Bastien Cagna, Thierry Chaminade, and Sylvain Takerkart. Inter-subject pattern analysis: A straightforward and powerful scheme for group-level mvpa. *NeuroImage*, 204:116205, 2020.

[27] Korbinian Brodmann and Laurence J. Gary. *Brodmann's localisation in the cerebral cortex : the principles of comparative localisation in the cerebral cortex based on cytoarchitectonics*. Springer, 2006.

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

[29] Alexandre Abraham, Fabian Pedregosa, Michael Eickenberg, Philippe Gervais, Andreas Mueller, Jean Kossaifi, Alexandre Gramfort, Bertrand Thirion, and Gael Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in Neuroinformatics*, 8, 2014.

[30] Matthew Brett, Christopher J. Markiewicz, Michael Hanke, Marc-Alexandre Côté, Ben Cipollini, Paul McCarthy, Dorota Jarecka, Christopher P. Cheng, Yaroslav O. Halchenko, Michiel Cottaar, and et al. nipy/nibabel:. Jun 2022.

[31] Mingrui Xia, Jinhui Wang, and Yong He. Brainnet viewer: A network visualization tool for human brain connectomics. *PLOS ONE*, 8(7):1–15, 07 2013.

[32] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empirical Softw. Engg.*, 19(5):1299–1334, oct 2014.

[33] Nikolaus Kriegeskorte, Rainer Goebel, and Peter Bandettini. Information-based functional brain mapping. *Proceedings of the National Academy of Sciences*, 103(10):3863–3868, 2006.

[34] Joset Etzel, Jeffrey Zacks, and Todd Braver. Searchlight analysis: Promise, pitfalls, and potential. *NeuroImage*, 78, 04 2013.

[35] Nikolaus Kriegeskorte, Marieke Mur, and Peter Bandettini. Representational similarity analysis – connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4, 02 2008.

[36] Haroon Popal, Yin Wang, and Ingrid R Olson. A Guide to Representational Similarity Analysis for Social Neuroscience. *Social Cognitive and Affective Neuroscience*, 14(11):1243–1253, 01 2020.