# PyPortal. Руководство пользователя

# Общие положения

Это руководство сделано на коленке, и ответственности за его содержание никто не несёт. При изменении функционала ПО руководство может обновляться с задержкой, потому что мне будет лень. Тем не менее, при возникновении вопросов рекомендуется сначала проверить актуальную версию руководства на репозитории github: <https://github.com/asbobe/PyPortal>, а потом уже давать автору сиречь мне, Анатолию Бобе, запрос на устранение ошибок/дополнение текста.

# Алгоритм Portal

Алгоритмы, составленные по структуре Portal, имеют следующие базовые модули:

1. Считывание данных ЭЭГ
   1. Загрузка данных edf, файлов меток и событий с заданной структуры
   2. Разбиение данных на эпохи и окна
2. Адаптивная предобработка (ICA преобразование, подбор информативных каналов/компонент итд.)
3. Расчет признаков по окнам
4. Обучение линейных преобразований (PCA, CSP, LDA, итд) и/или моделей классификации
5. Расчет и вывод результатов в виде Confusion Matrix, экспорт в Excel

# Как запустить

Задаём параметры чтения данных:

*reader = edfFileReader(…)*

Задаём классы для распознавания:

*reader.setClasses(…)*

Задаём параметры алгоритма:

*PortalParams = Params(…)*

Инициализируем выбранную модификацию алгоритма с заданными параметрами:

*PortalWorker = PortalSomething(PortalParams)*

Делаем адаптивную предобработку (при необходимости):

*someOutput = adaptivePreprocessing.someFun(…)*

Обучаем модель:

*PortalWorker.train(…)*

Тестируем модель на выборке для валидации:

*PortalWorker.validate(…)*

Используем его:

*PortalWorker.processChunk (…)*

Примеры работы есть в mainTest- функциях.

# Чтение данных из папки эксперимента (FileReader)

Функции для чтения данных из папки эксперимента находятся в файле **dataReader.py**

На данный момент реализован класс чтения данных ЭЭГ формата edf: **edfFileReader**

Для работы необходимо создать структуру каталогов, соответствующую протоколу хранения данных экспериментов. Общая схема хранения данных представлена на Рис. 1.

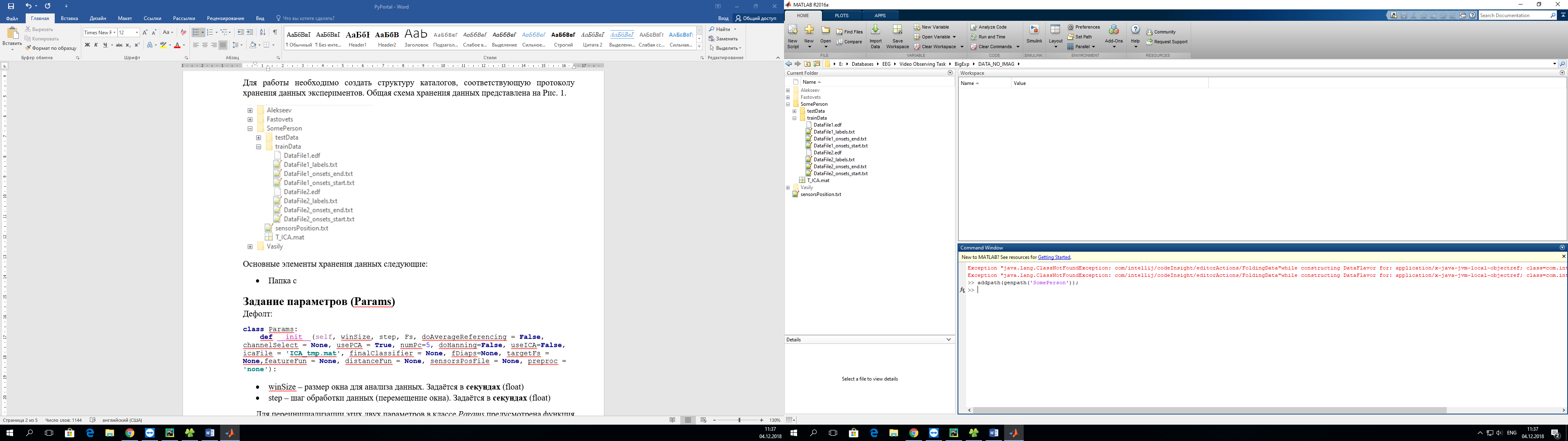
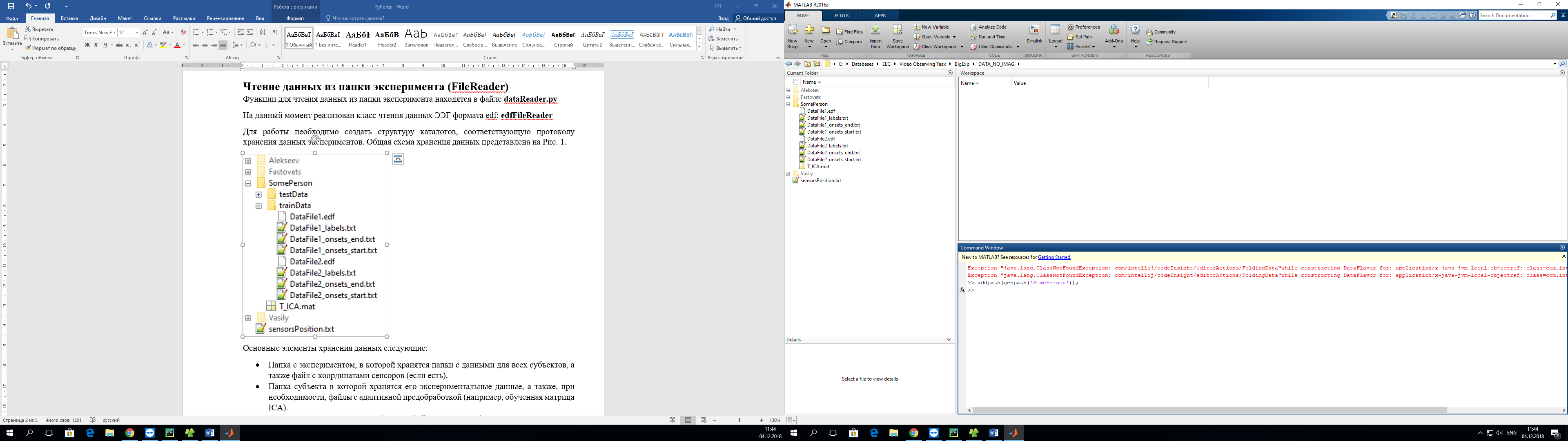
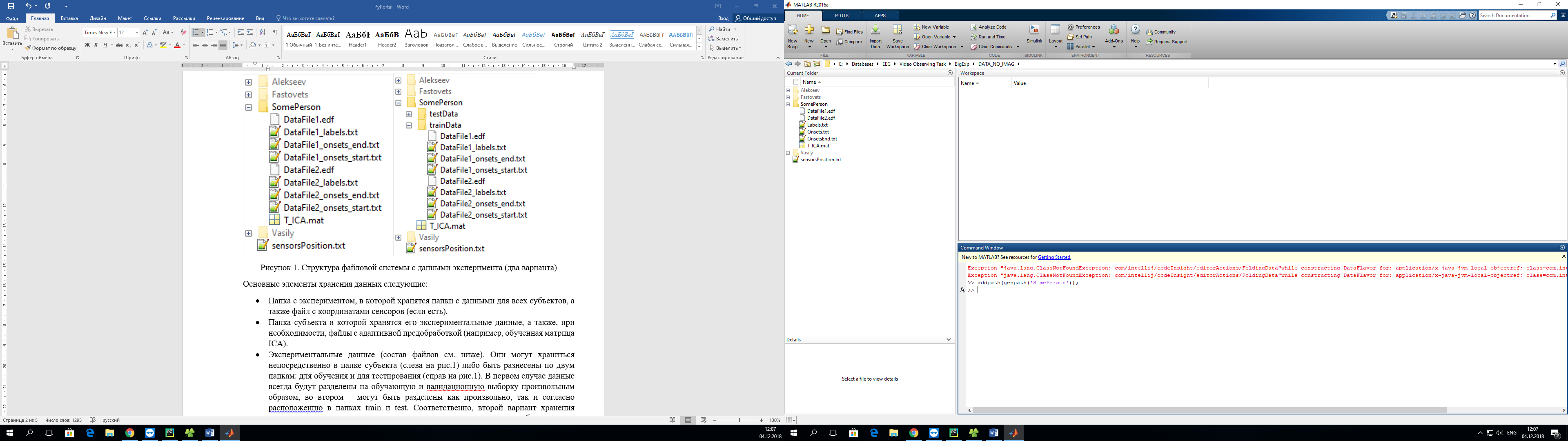
 

Рисунок 1. Структура файловой системы с данными эксперимента (три варианта)

Основные элементы хранения данных следующие:

* Папка с экспериментом, в которой хранятся папки с данными для всех субъектов, а также файл с координатами сенсоров (если есть).
* Папка субъекта в которой хранятся его экспериментальные данные, а также, при необходимости, файлы с адаптивной предобработкой (например, обученная матрица ICA). ВАЖНО: любые файлы с предобработкой создаются записываются автоматически в процессе обучения алгоритмов, во избежание путаницы с форматами внешних файлов.
* Экспериментальные данные (состав файлов см. ниже). Они могут храниться непосредственно в папке субъекта (слева и справа на рис.1) либо быть разнесены по двум папкам: для обучения и для тестирования (посередине на рис.1). В первом случае данные всегда будут разделены на обучающую и валидационную выборку произвольным образом, во втором – могут быть разделены как произвольно, так и согласно расположению в папках train и test. Соответственно, второй вариант хранения данных предпочтительнее, т.к. расширяет возможности разделения выборки.

Структура экспериментальных данных:

* Один или несколько файлов данных <ИмяФайла.edf>. При чтении данных все edf-файлы (как обучающие, так и тестовые) будут объединены в один массив.
* Файлы меток для каждого файла данных в формате <ИмяФайла\_labels.txt> Файл меток представляет собой текстовый столбец с порядковыми номерами-идентификаторами типа событий, начиная с 0. ВАЖНО: не допускается пропуск каких-либо номеров либо наличие отрицательных/дробных чисел в метках.
* Файлы времени начала событий для каждого файла данных в формате <ИмяФайла\_onsets\_start.txt>. Это текстовый столбец с временами начала событий относительно времени начала записи файла, **в секундах**. Данные заносятся в формате чисел с плавающей точкой.
* Файлы времени окончания событий для каждого файла данных в формате <ИмяФайла\_onsets\_end.txt>. Это текстовый столбец с временами окончания событий относительно времени начала записи файла, **в секундах**. Данные заносятся в формате чисел с плавающей точкой.

Если файлы меток либо времени начала/окончания событий совпадают для всех файлов данного субъекта, можно выбрать общие файлы для всех данных (рис.1 справа).

Задание параметров для чтения данных из описанной системы каталогов осуществляется классом **edfFileReader.** Экземпляр класса создается отдельно для каждого субъекта.Инициализация:

**class** edfFileReader:  
 **def** \_\_init\_\_(self, basedir, subj, chansToDrop=**None**, trainDir=**None**, testDir=**None**, icaFilename = **None**, commonLabelsPath=**None**, commonOnsetsPath = **None**, commonFinOnsetsPath=**None**)

* basedir – Абсолютный путь к общей (корневой) папке с данными (string)
* subj – имя папки субъекта (string)
* chansToDrop – «технические» каналы (аннотации, внешние каналы итд.). Подлежат удалению в самом начале эксперимента. Например: ['Diff 2', 'Diff 3', 'Diff 4', 'EDF Annotations'] (List<string>)
* trainDir – имя папки с обучающими данными (если есть) (string)
* testDir – имя папки с тестовыми данными (если есть) (string)
* icaFilename – имя файла с предобученной матрицей ICA (string)
* commonLabelsPath – имя файла с общими метками (если есть) (string)
* commonOnsetsPath – имя файла с общими временами старта событий (если есть) (string)
* commonFinOnsetsPath – имя файла с общими временами окончания событий (если есть) (string)

Чтение данных для конкретного субъекта:

edfFileReader.readData(startOffset=0.5, validationPart = 0.5, mixData=**True**)

* startOffset – время (в секундах) отступа от момента начала события (время на реакцию субъекта). Это время будет прибавлено ко всем меткам onsets\_start при фактическом разбиении данных (float)
* validationPart – доля данных, которые пойдут на тестовую выборку (активно только в случае случайного разделения данных) (float)
* mixData – флаг перемешивания данных. При его активации, данные для обучения и теста будут объединены в единый массив, перемешаны и случайно разделены заново. При неактивном флаге такая операция будет проделываться только в случае, когда данные исходно не разделены на папки train, test.

# Задание параметров (Params)

Дефолт:

**class** Params:  
 **def** \_\_init\_\_(self, winSize, step, Fs, doAverageReferencing = **False**, channelSelect = **None**, usePCA = **True**, numPc=5, doHanning=**False**, useICA=**False**, icaFile = **'ICA\_tmp.mat'**, finalClassifier = **None**, fDiaps=**None**, targetFs = **None**,featureFun = **None**, distanceFun = **None**, sensorsPosFile = **None**, preproc = **'none'**):

* winSize – размер окна для анализа данных. Задаётся в **секундах** (float)
* step – шаг обработки данных (перемещение окна). Задаётся в **секундах** (float)

Для переинициализации этих двух параметров в классе *Params* предусмотрена функция *setWinSizeAndStep(winSize, step)*

* Fs – частота дискретизации в **Гц** (float). ВАЖНО: не допускается в пределах одной сессии наличие данных с разной частотой дискретизации!
* channelSelect – порядковые номера каналов, которые будут использованы при обработке, либо номера компонент ICA в случае использования этого преобразования при предобработке. Анализ будет производится только по каналам, указанным в channelSelect. По умолчанию используются все каналы. (List<int>).
* fDiaps – частотные диапазоны, которые могут использоваться при расчете спектральных признаков для каждого из каналов, указанных в *channelSelect* (по умолчанию для каждого канала будут использоваться params.lowFreq=1, params.highFreq=35). Используется не во всех модификациях, т.к. приводит к различной размерности векторов по каналам и требует особой обработки данных. На данный момент опция доступна в PortalClassic и PortalCustomFreqs. (List<List<int>>, размерность [len(channelSelect), 2])
* doAverageReferencing – флаг, при выставлении которого производится вычитание из всех каналов усредненного сигнала по всем каналам (монтаж со средним референсом), (bool).
* usePCA – флаг использования метода главных компонент для сжатия размерности данных после расчета признаков (bool).
* numPC – число главных компонент PCA (в случае его использования). ВАЖНО: В реализации PortalClassic PCA применяется к склеенному из всех каналов единому вектору признаков, а в PortalIndepChan – отдельно по каждому из каналов обучается свой PCA. Т.е. число компонент нужно подбирать с учётом используемой модификации алгоритма. (int)
* doHanning – при расчете спектра – использование окна Хэннинга для предварительной свертки с сигналом. (bool)
* useICA – флаг, использование ICA-преобразования (bool)
* icaFile – адрес mat-файла с предобученной матрицей ICA для сигнала. Применяется **до** выбора channelSelect (т.е. при заданной матрице ICA индексы в channelSelect интерпретируются как номера компонент). (string)
* targetFs – частота, до которой будут передискретизированы данные. По умолчанию передискретизация не выполняется (int).
* sensorPosFile – адрес текстового файла с координатами сенсоров ЭЭГ-шлема. Файл должен быть в формате txt и содержать координаты в массиве размером (nChannels x 3). ВАЖНО: в текущей версии ПО параметр channelSelect не синхронизирован с sensorPosFile, т.е. нельзя выбирать каналы при расчетах, связанных с использованием координат сенсоров. (string)
* featureFun – функция расчета признаков сигнала (из файла featureFuncs). Подробнее функции будут рассмотрены ниже (function ID)
* distanceFun – функция расчета расстояний между векторами признаков. Применяется в случае, если отсутствует finalClassifier, либо если finalClassifier дает не номер класса, а характерный вектор для сэмпла. Подробнее функции будут рассмотрены ниже (function ID).
* finalClassifier – использование классификатора для анализа векторов признаков после их расчета (LDA, Random Forest, SVM, etc). Добавление классификаторов происходит в utilities, там прописываются соответствующие методы в trainClassifier и applyClassifier. Классификатор может давать окончательное решение, а может давать вектор (например, LDA). Во втором случае для расчета результатов понадобится использовать distanceFun (см. ниже). Подробнее функции будут рассмотрены ниже (function ID).
* Ещё туда можно вписывать всякие внутренние штуки типа параметров для функций расчета признаков (например, у меня там вбиты lowFreq=1, highFreq=35).

# Адаптивная предобработка

Адаптивная предобработка в среде Portal – это обучаемые методы преобразования данных, которые непосредственно влияют на базовые параметры алгоритма (т.е. на то, что входит в PortalParams), и поэтому вынесены в отдельные этапы, предшествующие инициализации объекта Portal. В текущей версии ПО реализованным вариантом такого метода является **autoSelectComps**. autoSelectComps анализирует мультиклассовые данные следующим образом:

* По каждому из *c* каналов (либо по каждой из компонент ICA) рассчитывается спектр
* Спектры усредняются для данных каждого из *k* классов
* Скользящее окно проходит по усредненным спектрам. После удаления артефактов рассчитывается среднее значение мощности *W̅c,f* соответствующей полосы частот (опять же, для каждого из классов)
* Для двух классов: с наименьшим значением *W̅c,f* (*Сmin*, *Сmax)* строится простейший пороговый классификатор:

*, ,,*

* Выбранный частотный диапазон сохраняется как информативный для данной компоненты, если *Acc1>compThresh, Acc2>compThresh*, где *compThresh* – заданный порог информативности компонент.
* По итогам анализа по каждой из компонент частотные диапазоны, выбранные по пересекающимся друг с другом окнам, объединяются в один большой диапазон.
* Если по результатам выполнения процедуры не получилось выбрать достаточное число компонент, содержащих информативные частотные диапазоны (*minCompsNum*), то алгоритм повторяется сначала, с уменьшенным на 0.05 значением *compThresh*.

Описанные параметры задаются в конструкторе метода:

**def** autoSelectComps(ClassifierParams, reader, compThresh, minCompsNum = **None**):

* ClassifierParams – объект класса PortalParams
* reader – объект класса dataReader
* compThresh – порог информативности компонент (0.5-0.95)
* minCompsNum – минимальное требуемое число компонент, которое должен выдать алгоритм (по умолчанию выбирается равным 2)

Метод возвращает значения comps, diaps, которые можно добавить в PortalParams для активации предобработки в основном пайплайне:

PortalWorker.params.channelSelect = comps  
PortalWorker.params.fDiaps = diaps

# Объекты класса Portal

Весь дальнейший пайплайн обработки данных производится объектами алгоритма Portal.

**class** Portal\_Classic():  
 **def** \_\_init\_\_(self, params):

params – описанные выше параметры (PortalParams)

Объекты типа Portal производят с данными последовательно следующие операции:

* Нарезка на окна по заданным параметрам
* Балансировка классов (по флагу *doBalanceLabels*)
* Расчет признаков по окнам
* Перемешивание признаков случайным образом
* Нормировка признаков на среднее, СКО
* Применение метода сжатия размерности (по заданным параметрам)
* Обучение (или запуск) классификатора
* Расчет принадлежности сэмпла к тому или иному классу по классификатору либо по функции расстояния (при валидации/работе)

В объектах Portal существует три основных функции:

* train: обучение модели
* validate: тестирование модели
* processChunk: работа модели (преобразование одного пришедшего куска данных)

train и validate принимают на вход объект класса **reader**, описанный выше. Функция train также принимает на вход флаг doBalanceLabels. При активном флаге происходит подсчет итогового числа сэмплов (уже после нарезки на окна) в каждом из классов, далее из массива данных случайным образом выкидываются сэмплы из тех классов, число сэмплов в которых больше, чем у самого малопредставленного класса.

Функция validate возвращает результаты работы модели на тестовых данных из reader-a, в форматах ClassAccuracy (array 1 x nClasses) и Confusion Matrix (array nClasses x nClasses).

Функция processChunk не возвращает ничего, но записывает внутрь объекта Portal результаты обучения в структуру trainResult. Если сохранить предобученный объект Portal в pickle-файл, его можно загрузить и использовать далее без каких-либо дополнительных настроек.

# Экспорт результатов классификации

В настоящее время реализован экспорт результатов в Microsoft Excel. Вспомогательный класс для этого – **ExcelExporter** – находится в файле PortalResExport.py. При инициализации объекта ExcelExporter создается файл Results.xlsx, в который записываются данные.

**def** addHeader(self, header):

Добавление заголовка (текст в ячейку (0,0))

**def** addNewSubject(self, subjName):

Добавить субъекта в таблицу

**def** addPairDataField(self, fieldName, dataVal1, dataVal2):

Добавить результаты бинарной классификации для текущего субъекта

**def** addFinisher(self, text, val):

Добавить поле под таблицей, снизу (можно добавлять несколько раз)

**def** Finish(self):

Закрыть файл