Разработка методов адаптации пользовательских интерфейсов для USSD-сервисов

**Человек:** Рост числа электронных устройств и их возможностей провоцирует развитие рекомендательных систем в целом и адаптивных интерфейсов в частности. Такие системы собирают информацию о пользовательском опыте и при помощи методов машинного обучения анализируют полученные данные. Растет не только количество устройств, но и пользователей, которые часто имеют различные нужды и требования. Особое внимание в данной статье уделяется интерфейсу USSD-сервисов. Целью работы является разработка подхода, позволяющего снизить среднее количество запросов, совершаемых пользователями, в процессе работы с текстовым меню. Для поиска подходящего решения были применены методы системного анализа, в результате была установлена, сильная зависимость между потребностями каждого пользователя и контексте, в которой он находится. Использование информации о контексте выполнения, позволило улучшить существующие методы и получить высокие результаты. В основе подхода лежит использование двух этапной кластеризации: кластеризация по интересам на первом этапе и по контекстам – на втором. Для тестирования были использованы данные, собранные в процессе взаимодействия пользователей с USSD-сервисом.

**Key words:** адаптивные интерфейсы, USSD, адаптация иерархичных интерфейсов, адаптация древовидных меню, кластеризация пользователей, построение модели пользователя, свойства пользователей, определение контекста, адаптация графа меню, стереотипная адаптация

=================================

**FastText\_KMeans\_Clean:** Группировка пользователей в кластеры позволяет оптимизировать интерфейс для каждой конкретной группы, при этом избежав больших накладных расходов для хранения индивидуального дерева меню. Возможные услуги нумеруются от 1 до N, и для каждой функции на основе статистических данных рассчитывается ее вес. Иными словами, вес внутренних вершин `v\_(i)` равен сумме всех листов исходного дерева, входящих в поддерево с корнем в `v\_(i)` , умноженное на длину пути до вершины `v\_(i)`. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь.

**Key words part:** 0.5

=================================

**FastText\_KMeans\_Raw/:** Группировка пользователей в кластеры позволяет оптимизировать интерфейс для каждой конкретной группы, при этом избежав больших накладных расходов для хранения индивидуального дерева меню. Опишем задачу оптимизации. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. `w\_(i) = sum\_(j=1)^K n\_(ij)`. Иными словами, вес внутренних вершин `v\_(i)` равен сумме всех листов исходного дерева, входящих в поддерево с корнем в `v\_(i)` , умноженное на длину пути до вершины `v\_(i)`. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь. Рис.3 Диаграмма "Оптиизация количества запросов (C учетом контекстов)".

**Key words part:** 0.53125

=================================

**FastText\_PageRank\_Clean/:** Опишем задачу оптимизации. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. Соответственно `v\_(i)` обозначим вершину под номером i. Если `v\_(i)` - лист. `w\_(i) = sum\_(j=1)^K n\_(ij)`. `w\_(i) = sum w\_(j)d\_(i)`. Отсортируем список вершин по весу, присвоим i-ому слоту поддерево i. `D=1- ( sum\_(j=1)^K n\_(j) )/( sum\_(j=1)^K m\_(j) ) `.

**Key words part:** 0.28125

=================================

**FastText\_PageRank\_Raw/:** Опишем задачу оптимизации. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. Соответственно `v\_(i)` обозначим вершину под номером i. Если `v\_(i)` - лист. `w\_(i) = sum\_(j=1)^K n\_(ij)`. `w\_(i) = sum w\_(j)d\_(i)`. Отсортируем список вершин по весу, присвоим i-ому слоту поддерево i. `D=1- ( sum\_(j=1)^K n\_(j) )/( sum\_(j=1)^K m\_(j) ) `.

**Key words part:** 0.28125

=================================

**Mixed\_ML\_TR/:** В данной статье предлагается использовать информацию о контексте (или окружении) пользователя. Наиболее важными из них являются: точность предлагаемых изменений относительно реакции пользователя, степень изменения и предсказуемости интерфейса, частота выполняемых адаптаций. Каждый алгоритм адаптации так или иначе использует математическую модель пользователя. Правильный выбор модели является крайне важным этапом разработки алгоритма, так как хранение лишней информации ресурсозатратно, а отсутствие какой-либо информации чревато потерей времени. Поскольку данная работа посвящена разработке метода автоматической адаптации USSD-меню, стоит немного рассказать о данной технологии. На вход нейронной сети подаются данные о кластеризуемом объекте. Этого должно хватить, поскольку пользователи открывшие данные контексты имеют схожие группы интересов, следовательно, имеют схожие контексты выполнения. Этот процесс выполняется для каждого кластера полученного на первом этапе (см. рис. 2). Каждой вершине присвоим вес `w\_(i)` , определим это число следующим образом. Также был разработан метод, позволяющий адаптировать меню на основе данных, собранных в процессе работы пользователей с сервисом.

**Key words part:** 0.6875

=================================

**MultiLingual\_KMeans/:** Наиболее важными из них являются: точность предлагаемых изменений относительно реакции пользователя, степень изменения и предсказуемости интерфейса, частота выполняемых адаптаций. Правильный выбор модели является крайне важным этапом разработки алгоритма, так как хранение лишней информации ресурсозатратно, а отсутствие какой-либо информации чревато потерей времени. Поскольку данная работа посвящена разработке метода автоматической адаптации USSD-меню, стоит немного рассказать о данной технологии. На вход нейронной сети подаются данные о кластеризуемом объекте. Этого должно хватить, поскольку пользователи открывшие данные контексты имеют схожие группы интересов, следовательно, имеют схожие контексты выполнения. Этот процесс выполняется для каждого кластера полученного на первом этапе (см. рис. 2). Каждой вершине присвоим вес `w\_(i)` , определим это число следующим образом.

**Key words part:** 0.625

=================================

**Multilingual\_PageRank/:** Кроме того, использование данного подхода не представляется возможным в рамках протокола USSD. Максимальная длина контента может составлять 182 символа. К сожалению, если пользователь слишком консервативен, данный вид адаптации не принесет никаких результатов. Во-первых, он гарантированно не увеличивает длину путей ни до одной из функций. Данный класс сетей реализует метод обучения без учителя. Для всех j, таких что вершина j входит в поддерево с корнем в i. В большинстве предыдущих работ этому процессу уделялось мало времени и при получении результатов не учитывалась вероятность выбора пользователем более длинного пути. Конечно, результаты - несколько ниже чем у предшественников.

**Key words part:** 0.53125

=================================

**RuBERT\_KMeans\_Without\_ST/:** Сначала происходит выборка тех сессий, которые являлись успешными, то есть в рамках которых пользователи подключили те или иные услуги. `w\_(i) = sum\_(j=1)^K n\_(ij)`. Но для этого необходимо разработать метод получения вероятности, с которой пользователь выберет более длинный путь. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь.

**Key words part:** 0.375

=================================

**RuBERT\_KMeans\_With\_ST/:** Во-вторых, при необходимости технической поддержки пользователя, можно указать расположение требуемых им функций в оригинальном меню. Опишем задачу оптимизации. `w\_(i) = sum w\_(j)d\_(i)`. Реализовать алгоритм случайного выбора того или иного пути пользователей не так трудно.

**Key words part:** 0.4375

=================================

**RUBERT\_page\_rank\_Without\_ST/:** Максимальная длина контента может составлять 182 символа. Число выходов равно числу кластеров (как это представлено на рис. 1). Затем полученные сессии кластеризуются на 7-9 групп. Для заказа любой услуги можно подсчитать количество нажатий клавиш. Конечно, результаты - несколько ниже чем у предшественников.

**Key words part:** 0.28125

=================================

**RUBERT\_page\_rank\_With\_ST/:** Это является причиной слишком глубоких меню. Опишем задачу оптимизации. `w\_(i) = sum\_(j=1)^K n\_(ij)`. `w\_(i) = sum w\_(j)d\_(i)`. `D=1- ( sum\_(j=1)^K n\_(j) )/( sum\_(j=1)^K m\_(j) ) `.

**Key words part:** 0.34375

=================================

**RUSBERT\_KMeans\_Without\_ST/:** Безусловно, это гораздо дешевле для производителей ПО, но пользователи, не являющиеся продвинутыми, могут не разобраться с возможностями ручной доработки интерфейса [2]. Чтобы адаптация была возможной, система должна собирать информацию о пользователе в процессе взаимодействия с ним. Этого должно хватить, поскольку пользователи открывшие данные контексты имеют схожие группы интересов, следовательно, имеют схожие контексты выполнения. Пусть N – количество услуг, K–количество пользователей, `n\_(ij)` ` ` — количество вызовов i-й услуги пользователем j, а `d\_i` —количество переходов необходимое для активации i-й услуги.

**Key words part:** 0.59375

=================================

**RUSBERT\_KMeans\_With\_ST/:** Поскольку информацию об устройстве нельзя получить в рамках протокола USSD, было решено использовать информацию о пользовательском опыте и контексте. `w\_(i) = sum\_(j=1)^K n\_(ij)`. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь. Первый пункт позволяет получить информацию о качестве исходного пользовательского меню, второй позволит нам сравнить эффективность использования двойной кластеризации.

**Key words part:** 0.53125

=================================

**RUSBERT\_page\_rank\_Without\_ST/:** Кроме того, использование данного подхода не представляется возможным в рамках протокола USSD. Этот подход обладает рядом преимуществ. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. Если `v\_(i)` - лист. `w\_(i) = sum\_(j=1)^K n\_(ij)`.

**Key words part:** 0.3125

=================================

**RUSBERT\_page\_rank\_With\_ST/:** Этот протокол имеет ряд ограничений, одним из которых является скудность получаемой и предоставляемой информации. Этот подход обладает рядом преимуществ. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. `w\_(i) = sum\_(j=1)^K n\_(ij)`. `D=1- ( sum\_(j=1)^K n\_(j) )/( sum\_(j=1)^K m\_(j) ) `.

**Key words part:** 0.28125

=================================

**Simple\_PageRank/:** Данная технология пользуется большим спросом в странах, где мобильный интернет слабо развит. Дизайн пользовательского интерфейса влияет на количество усилий, прилагаемых пользователем для достижения своих целей, а также на скорость освоения программной системы. Простейшим примером таких систем может служить браузер, который выносит наиболее часто посещаемые сайты на главную страницу или дополняет адрес при вводе первых символов. Для открытия сессии пользователь набирает команду на телефоне вида \*.#. В процессе изучения литературы посвященной вопросу создания адаптивных интерфейсов, бала изучена статья [1], в которой предлагалось разделить модель пользователя на 3 части: информация о пользовательском опыте, информация об устройстве, информация о контексте. Тестирование показало, что разработанный метод, использующий двухэтапную кластеризацию, позволяет сократить среднее количество действий пользователей на 34%.

**Key words part:** 0.5625

=================================

**TextRank/:** В то же время число абонентов огромно, поэтому не представляется возможным хранение индивидуального варианта меню для каждого пользователя. Группировка пользователей в кластеры позволяет оптимизировать интерфейс для каждой конкретной группы, при этом избежав больших накладных расходов для хранения индивидуального дерева меню. В данной статье предлагается использовать информацию о контексте (или окружении) пользователя. Для каждой группы происходит выборка всех сессий, которые были открыты пользователями данного кластера. Где K – колличество пользователей кластера, `n\_(j)` — количество запросов совершаемых пользователем в оптимизированном меню, `m\_(j)` — количество запросов совершаемых пользователем в первоначальном меню. Также был разработан метод, позволяющий адаптировать меню на основе данных, собранных в процессе работы пользователей с сервисом.

**Key words part:** 0.53125

=================================

**TF-IDF\_KMeans/:** Поскольку данная работа посвящена разработке метода автоматической адаптации USSD-меню, стоит немного рассказать о данной технологии. На втором этапе, когда пользователь начинает все чаще пользоваться функцией, расположенной в новом месте, старые пункты удаляются. Поскольку информацию об устройстве нельзя получить в рамках протокола USSD, было решено использовать информацию о пользовательском опыте и контексте. Рис. 1, Иллюстрация работы нейронной сети. Решать данную задачу перебором всех возможных вариантов меню не представляется возможным. Каждой вершине присвоим вес `w\_(i)` , определим это число следующим образом. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь. Для каждой модификации было подсчитано сокращение количества запросов.

**Key words part:** 0.625

=================================

**Текст:** Большой разброс интересов пользователей усложняет разработку интерфейсов и заставляет разработчиков отказаться от статичных страниц в пользу адаптируемых или же настраиваемых интерфейсов.. В данной статье рассматривается проблема адаптации меню на примере USSD протокола. Данная технология пользуется большим спросом в странах, где мобильный интернет слабо развит. Этот протокол имеет ряд ограничений, одним из которых является скудность получаемой и предоставляемой информации.. Интерфейсы USSD-сервисов на ряду с другими пользовательскими интерфейсами нуждаются в адаптации. В связи со спецификой данного протокола, структура меню хранится на стороне оператора. В то же время число абонентов огромно, поэтому не представляется возможным хранение индивидуального варианта меню для каждого пользователя... USSD протокол имеет ряд преимуществ: простота, защищенность, бесплатность. Но в то же время он имеет ограничение на размер одного сообщения. Это является причиной слишком глубоких меню. Большое число подменю увеличивает ресурсозатраты операторов и снижает удовлетворенность пользователей.. Группировка пользователей в кластеры позволяет оптимизировать интерфейс для каждой конкретной группы, при этом избежав больших накладных расходов для хранения индивидуального дерева меню. Стоит так же заметить, что процесс оптимизации - процесс вычислительно сложный и производить его в режиме реального времени в большинстве случаев не представляется возможным.. В данной статье предлагается использовать информацию о контексте (или окружении) пользователя. Основной идеей является то, что один и тот же пользователь, находясь в разных контекстах, может воспринимать информацию по-разному и иметь разные нужды [1].. Дизайн пользовательского интерфейса влияет на количество усилий, прилагаемых пользователем для достижения своих целей, а также на скорость освоения программной системы. Многие современные системы позволяют самостоятельно настраивать пользовательский интерфейс. Безусловно, это гораздо дешевле для производителей ПО, но пользователи, не являющиеся продвинутыми, могут не разобраться с возможностями ручной доработки интерфейса [2]. Кроме того, использование данного подхода не представляется возможным в рамках протокола USSD.. Системы с автоматической адаптацией дают возможность пользователю не задумываться о вопросах интерфейса: достаточно начать пользоваться приложением, и программа подстроится под пользователя сама.. Простейшим примером таких систем может служить браузер, который выносит наиболее часто посещаемые сайты на главную страницу или дополняет адрес при вводе первых символов.. Алгоритмы, применяемые в подобных системах, могут сильно отличаться друг от друга, но можно выделить ряд характеристик необходимых для оценки алгоритма. Наиболее важными из них являются: точность предлагаемых изменений относительно реакции пользователя, степень изменения и предсказуемости интерфейса, частота выполняемых адаптаций. Точность играет максимальную роль, повышение точности – смысл разработки новых алгоритмов.. Каждый алгоритм адаптации так или иначе использует математическую модель пользователя. Для построения такой модели применяются статистические методы, использующие исторические данные, полученные в результате сбора информации.. Чтобы адаптация была возможной, система должна собирать информацию о пользователе в процессе взаимодействия с ним. Правильный выбор модели является крайне важным этапом разработки алгоритма, так как хранение лишней информации ресурсозатратно, а отсутствие какой-либо информации чревато потерей времени.. Существует два принципиально разных подхода к разработке адаптивных интерфейсов: индивидуальный и стереотипный. Индивидуальный подход удобен в случаях, когда нет необходимости хранить и обрабатывать информацию о каждом пользователе на сервере, а наоборот, можно вынести часть логики на клиентские машины. Стереотипный подход основан на классификации пользователей на несколько классов.. Модель пользователя можно представить в разных формах, наиболее простой из которых является представление в виде вектора интересов. Возможные услуги нумеруются от 1 до N, и для каждой функции на основе статистических данных рассчитывается ее вес. Такой вектор достаточно просто построить, но для этого необходимо большое количество данных.. Поскольку данная работа посвящена разработке метода автоматической адаптации USSD-меню, стоит немного рассказать о данной технологии.. USSD (Unstructured Supplementary Service Data) – сессионно ориентированный протокол передачи неструктурированных данных, реализованный для сетей стандарта GSM. Эта технология позволяет осуществить высокоскоростное взаимодействие с пользователем в условиях отсутствия интернета [2].. Для открытия сессии пользователь набирает команду на телефоне вида \*...#. Такой формат позволяет телефону понять, что нужно произвести не звонок, а USSD запрос. Запросы могут быть как единичными (например, запрос баланса), так и интерактивными. Если ответ пользователя не был получен сервером в течении 60-ти секунд, то сессия обрывается. Максимальная длина контента может составлять 182 символа.. Эта технология имеет ряд преимуществ: она является бесплатной для абонентов, кроме того защищенность данного протокола позволяет использовать ее для предоставления услуг, чувствительных к защищенности информации.. При выборе алгоритма адаптации USSD-меню важно учитывать привычки пользователей. Резкое изменение меню усилит недовольство абонентов и усложнит процесс обучения.. Для повышения предсказуемости адаптацию часто делят на 2 этапа. На первом этапе новые пункты меню дублируются старыми. Это сделано для того, чтобы пользователь по привычке мог выбрать старый вариант расположения кнопок. На втором этапе, когда пользователь начинает все чаще пользоваться функцией, расположенной в новом месте, старые пункты удаляются. К сожалению, если пользователь слишком консервативен, данный вид адаптации не принесет никаких результатов.. Этот подход обладает рядом преимуществ. Во-первых, он гарантированно не увеличивает длину путей ни до одной из функций. При этом пользователю, знакомому с оригинальным вариантом интерфейса и уже запомнившему пути до нужных ему функций сервиса, будет легко найти их в адаптированном варианте: из путей пропадают лишь некоторые промежуточные элементы. Во-вторых, при необходимости технической поддержки пользователя, можно указать расположение требуемых им функций в оригинальном меню.. При проведении адаптаций ставится цель - снизить общее число переходов, необходимое пользователю для достижения конечной цели. Это в свою очередь должно повысить удовлетворенность пользователей.. В предыдущих работах [2-3] кластеризация производилась по вектору интересов. И это давало хорошие результаты.. В процессе изучения литературы посвященной вопросу создания адаптивных интерфейсов, бала изучена статья [1], в которой предлагалось разделить модель пользователя на 3 части: информация о пользовательском опыте, информация об устройстве, информация о контексте. Поскольку информацию об устройстве нельзя получить в рамках протокола USSD, было решено использовать информацию о пользовательском опыте и контексте.. На этапе анализа были сформированы следующие ограничения для предлагаемого алгоритма:. Первым этапом является выделение групп пользователей по интересам. В качестве метода классификации пользователей решено использовать нейронную сеть.. Искусственные нейронные сети призваны решать множество различных задач. К задачам, решаемым при помощи данной технологии можно отнести: распознавание образов, аппроксимацию, оптимизацию [4].. Нейронные сети Кохонена, которые были использованы при реализации программного модуля, созданы для решения задачи кластеризации. На вход нейронной сети подаются данные о кластеризуемом объекте. Число выходов равно числу кластеров (как это представлено на рис. 1). Стоит так же отметить, что нейронные сети Кохонена не нуждаются в обучающей выборке. Данный класс сетей реализует метод обучения без учителя. Число входов равно размерности вектора пользовательских интересов.. . Рис. 1, Иллюстрация работы нейронной сети. После кластеризации пользователей по интересам, необходимо перейти ко второму этапу.. Будем считать, что каждая сессия выполняется в рамках одного контекста. Это решение обусловлено спецификой протокола USSD: разница по времени между двумя запросами в рамках одной сессии не может превышать одной минуты. Следовательно, информацию, специфичную для сессии (время отклика, id вышки сотовой связи, время суток, и т.д.), можно использовать как информацию о контексте.. Для повышения точности определения контекста было принято решение рассматривать каждую группу пользователей отдельно. Это решение основывается на том, что люди, имеющие схожие интересы, должны иметь схожие контексты выполнения.. Для каждой группы происходит выборка всех сессий, которые были открыты пользователями данного кластера. Затем полученные сессии кластеризуются на 7-9 групп. Этого должно хватить, поскольку пользователи открывшие данные контексты имеют схожие группы интересов, следовательно, имеют схожие контексты выполнения. Этот процесс выполняется для каждого кластера полученного на первом этапе (см. рис. 2).. . Рис. 2, Иллюстрация второго этапа кластеризации. Опишем задачу оптимизации.. Пусть N – количество услуг, K–количество пользователей, `n\_(ij)` ` ` — количество вызовов i-й услуги пользователем j, а `d\_i` —количество переходов необходимое для активации i-й услуги.. Тогда задача оптимизации записывается в виде:. `sum\_(i=1)^Nsum\_(i=1)^N n\_(ij) d\_i -> min`. Решать данную задачу перебором всех возможных вариантов меню не представляется возможным. Алгоритм решения данной задачи гарантирует что не увеличиваются на каждой итерации. Т.о. на каждом шаге мы гарантированно не ухудшаем результат.. В рамках данной работы предлагается использование следующего алгоритма.. Сначала происходит выборка тех сессий, которые являлись успешными, то есть в рамках которых пользователи подключили те или иные услуги. Затем определяется количество активаций каждой функции пользователями текущего кластера.. Построим граф меню и найдем в этом графе пустые слоты.. Обозначим граф меню как `G = (V, E)` . На данном этапе можем не учитывать тот факт что граф является ориентированным. Соответственно `v\_(i)` обозначим вершину под номером i. Каждой вершине присвоим вес `w\_(i)` , определим это число следующим образом.. Если `v\_(i)` - лист. `w\_(i) = sum\_(j=1)^K n\_(ij)`. Иначе,. `w\_(i) = sum w\_(j)d\_(i)`. Для всех j, таких что вершина j входит в поддерево с корнем в i.. Иными словами, вес внутренних вершин `v\_(i)` равен сумме всех листов исходного дерева, входящих в поддерево с корнем в `v\_(i)` , умноженное на длину пути до вершины `v\_(i)`.. Пронумеруем все слоты, обойдя граф в ширину. Отсортируем список вершин по весу, присвоим i-ому слоту поддерево i.. Этот процесс запускается для каждого кластера.. ` `. ` `. После проведения всех операций необходимо произвести тестирование. Задача тестирования является достаточно важной. В большинстве предыдущих работ этому процессу уделялось мало времени и при получении результатов не учитывалась вероятность выбора пользователем более длинного пути.. Реализовать алгоритм случайного выбора того или иного пути пользователей не так трудно. Но для этого необходимо разработать метод получения вероятности, с которой пользователь выберет более длинный путь.. Вероятность можно получить из логов, включив в формулу число активаций той или иной функции: чем чаще данная функция была подключена пользователем, тем больше вероятность что пользователь выберет более длинный путь.. Для каждого пользователя кластера известны заказываемые им услуги и количество их заказов. Для заказа любой услуги можно подсчитать количество нажатий клавиш. Далее, считается минимальное количество нажатий клавиш для заказа соответствующих услуг в модифицированном меню. Эти величины суммируются по всем пользователям кластера, находится их отношение и вычитается из единицы:. `D=1- ( sum\_(j=1)^K n\_(j) )/( sum\_(j=1)^K m\_(j) ) `. . Где K – колличество пользователей кластера, `n\_(j)` — количество запросов совершаемых пользователем в оптимизированном меню, `m\_(j)` — количество запросов совершаемых пользователем в первоначальном меню.. Было решено проверить алгоритм оптимизации на следующих данных.. После проведения оптимизации меню для каждого кластера, было получено 100 различных модификаций исходного меню (10 основных кластеров и 10 подкластеров для каждого из них). Для каждой модификации было подсчитано сокращение количества запросов.. Первый пункт позволяет получить информацию о качестве исходного пользовательского меню, второй позволит нам сравнить эффективность использования двойной кластеризации.. После выполнения всех действий были получены следующие результаты:. Все пользователи. Выделенные группы пользователей (без учета контекстов). Выделенные группы пользователей (с учетом контекстов). В результате тестирования видно, что алгоритм значительно сократил количество запросов. Также видно, что использование информации о контексте вносит заметный вклад в решение поставленной задачи.. Более подробные результаты показаны на диаграммах 1 и 2:. . Рис.3 Диаграмма «Оптиизация количества запросов (C учетом контекстов)». . Конечно, результаты - несколько ниже чем у предшественников. Во-первых, это связано с разнородностью используемых для адаптации данных, во-вторых – с изначально более качественным меню.. . В данной работе были рассмотрены проблемы, которые возникают при автоматической адаптации мобильных USSD-интерфейсов. Также был разработан метод, позволяющий адаптировать меню на основе данных, собранных в процессе работы пользователей с сервисом. Более того, была создана программная система, реализующая данный метод и позволяющая его протестировать.. Тестирование показало, что разработанный метод, использующий двухэтапную кластеризацию, позволяет сократить среднее количество действий пользователей на 34%.