МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«МЭИ»

А.С. Мохов, В.О. Толчеев, А.А. Бородкин

АНАЛИЗ И ОБРАБОТКА ТЕКСТОВЫХ ДАННЫХ

Практикум

по курсу «Интеллектуальные информационные системы»

для студентов, обучающихся по
направлению подготовки магистров
27.04.04 «Управление в технических
системах»

Москва

Издательство МЭИ

2019

УДК 519.254

ББК 22.19

М 861

Утверждено учебным управлением НИУ «МЭИ» в качестве учебного издания

Подготовлено на кафедре управления и информатики

Рецензенты: докт. техн. наук, докт. экон. наук, проф. А.И. Орлов;
докт. техн. наук, проф. Г.Ф. Филаретов;

Мохов А.С., Толчеев В.О., Бородкин А.А.

М 861 Анализ и обработка текстовых данных: практикум / А.С. Мохов, В.О. Толчеев, А.А. Бородкин. – М.: Издательство МЭИ, 2019. – 56 с.

ISBN

Содержит теоретический материал по методам обработки и анализа текстовых и фактографических данных для решения задач классификации и кластеризации, а также описания четырех лабораторных работ по курсу «Интеллектуальные информационные системы» с методическими указаниями по их выполнению. Предназначен для студентов АВТИ, обучающихся по направлению «Управление в технических системах».

УДК 519.254

ББК 22.19

ISBN

1 ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время для решения практических проблем в области обработки и анализа фактографической, текстовой и мультимедийной информации используются библиотеки с открытым исходным кодом. Такие библиотеки позволяют вместо разработки специальных программ для каждой конкретной задачи применять хорошо отлаженные и апробированные на практике «стандартные» библиотечные процедуры.

В лабораторном практикуме в качестве основного инструментария для анализа используется библиотека Scikit-Learn, предназначенная для машинного обучения и прогнозной аналитики на языке Python. В ее состав входят алгоритмы классификации и регрессии, кластеризации и визуализации данных, выделения информативных признаков и верификации результатов. Важным преимуществом библиотеки Scikit-Learn является наличие хорошо документированных материалов, поясняющих реализованные алгоритмы и способы настройки неизвестных параметров.

Наряду с основными сведениями из теории обработки и анализа данных нами приводятся подробные рекомендации по использованию процедур, реализованных в *Scikit-Learn.* Эти рекомендации позволяют пошагово выполнять задания лабораторных работ и применять на практике теоретические знания, полученные на лекциях.

В практикуме перед учащимися ставится задача не только реализовать типовые процедуры *Scikit-Learn*, но и научиться правильно выбирать алгоритмы выявления (и построения) системы признаков (*Feature Selection* и *Feature Engineering*), обосновывать применение тех или иных методов анализа данных с учетом специфики исследуемых выборок.

В практикуме дается теоретический материал по методам обработки и анализа текстовых и фактографических данных (документов и объектов) для решения задач классификации и кластеризации приводится описание четырех лабораторных работ, которые посвящены бинарной классификации фактографических данных, предварительной обработке текстовых данных, многоклассовой классификации текстовой информации и кластеризации фактографических данных. В качестве исходных выборок применяются как специализированные модули генерации выборок, так и «эталонные» общедоступные массивы.

Разработанный практикум предназначен для магистров кафедры управления и информатики, обучающихся по направлению подготовки 27.04.04 – «Управление в технических системах». При выполнении заданий используются знания и умения, полученные студентами в

бакалаврских курсах «Статистические методы инженерных исследований», «Методы оптимизации», «Методы обработки данных», «Программное обеспечение автоматизированных систем».

1 ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ К ЛАБОРАТОРНОЙ РАБОТЕ №1 «БИНАРНАЯ КЛАССИФИКАЦИЯ ФАКТОГРАФИЧЕСКИХ ДАННЫХ»

1.1 Постановка задачи классификации данных на примере текстовых документов

Приведем постановку задачи классификации на более конкретном примере объектов в виде текстовых документов. При этом, под свойствами объектов мы будем понимать термины документа.

Имеется $\mathbf{X} = \{\overrightarrow{X_1}, \ldots, \overrightarrow{X_j}, \ldots \overrightarrow{X_N}\}$ ($j=1,..,N$)– множество документов и фиксированное число классов $\mathbf{Q}=\{Q_1, \ldots, Q_g, \ldots Q_G\}$ (Q_g - метка g-го класса, $g=1,..,G$, G – число классов). Каждый документ X представляется в виде вектора в пространстве терминов (M – размерность пространства, которая определяется словарем терминов).

Существует некоторая зависимость, позволяющая по $\vec{X} \in \mathbf{X}$ предсказать метку класса $Q \in \mathbf{Q}$. Обозначим эту неизвестную зависимость a: $\vec{X} \rightarrow Q$. Далее она будет называться: целевая функция или решающее правило или классификатор. Ставится задача построения классификатора $\hat{a}$, максимально близкого к a на выбранном словаре терминов

Построение классификатора осуществляется на множестве документов **D**, размеченном экспертом («учителем»), $\mathbf{D} \in \mathbf{X}$. С помощью *алгоритма обучения J* требуется получить преобразование $J(\mathbf{D})=\hat{a}$, т.е. алгоритм обучения J получает на вход обучающее множество и формирует решающее правило $\hat{a}$. Результирующий классификатор может быть получен с помощью различных алгоритмов обучения. Решающее правило $\hat{a}$ и алгоритм обучения J часто не различают и используют единое обозначение – в нашем случае будем использовать $\hat{a}$.

При построении $\hat{a}$ необходимо учитывать, какая классификация проводится. Возможно три варианта:

- Бинарная (двухклассовая) классификация: $Q = \{0, 1\}$, т.е. документы распределяются по двум классам.
- Многоклассовая классификация в G непересекающихся классов.

- Многоклассовая классификация в G пересекающихся классов. В этом случае один и тот же объект $\overrightarrow{X_j}$ может быть отнесен на основе своего терминологического состава сразу к нескольким классам одновременно (отнесение документа к классу может быть как однозначным, так и задаваться определенной степенью уверенности классификатора).

1.1.1 Способы формирования обучающих и тестовых выборок

Ранее было отмечено, что для построения классификатора $\hat{a}$ необходимо иметь множество документов **D**, размеченное экспертом («учителем»). В связи с этим классификацию часто называют «обучением с учителем» («*Supervised learning*»). Далее будем считать, что эксперт (эксперты) безошибочно определяют метки классов, т.е. рассматривается задача классификации с «идеальным» учителем.

Для оценки неизвестной целевой функции необходимо получить репрезентативное множество **D**, которое достоверно отражает структуру и специфику исходного множества **X**. В ряде случаев, когда **X** велико, формирование **D** можно осуществлять «классическим» способом, извлекая документы из **X** случайным образом (без возвращения).

Рассмотрим способы обучения и тестирования (экзамена) классификатора $\hat{a}$, когда множество **D** достаточно велико. В этом случае **D** разбивается на обучающие и тестовые (экзаменационные) выборки. Чаще всего используются следующие два подхода:

1) *Оценка точности по тестовым выборкам.* Множество **D** случайным образом разбивается на две части. По первой части (обучающая выборка – около 60-70 процентов множества **D**) производится настройка неизвестных параметров классификатора, по второй части (тестовая выборка – около 30-40 процентов множества **D**) оценивается качество классификации.

2) *Оценка точности с помощью v–кратной перекрестной проверки (v–fold cross validation)*. Из множества ***D*** случайным образом формируются v-подвыборок размера n (обычно v=5 или v=10). При этом (v–1)-подвыборок объединяются в обучающую выборку, а одна подвыборка является тестовой. Обучение и тестирование повторяются v раз, после чего вычисляются усредненные показатели качества.

В случае, ограниченного числа документов (**D** имеет малый объем) обычно применяются две другие стратегии построения и тестирования классификатора $\hat{a}$.

3) *Оценка точности с помощью скользящего контроля (или «метод складного ножа», «Jackknife»).* Для обучения применяется вся выборка кроме одного документа, оставшийся документ используется

для проверки. Затем этот документ включается в общую выборку, а для контроля извлекается другой документ. Данная процедура повторяется для всех членов исходной выборки. Таким образом, объем обучающей выборки все время равен N–1 элементу, где N – объем множества **D**.

4) *Метод статистического моделирования (Bootstrap)*. Предложен для формирования обучающих и экзаменационных выборок в условиях ограниченного количества наблюдений с целью проведения многократного обучения и тестирования. При этом множество **D** принимается за «генеральную совокупность» и из него случайным образом производится составление обучающих и тестовых подвыборок (т.е. используется отбор с возвращением).

1.1.2 Показатели качества классификации

Рассмотрим способы определения качества классификации (близости $\hat{a}$ к a) по тестовой выборке в простейшем случае бинарной классификации. Составим *матрицу ошибок* (*Confusion Matrix*, матрица неточностей, см. таблицу 1). Строки этой таблицы соответствуют результатам классификации с помощью классификатора $\hat{f}$, а столбцы экспертно указанным меткам класса в тестовой выборке.

Таблица 1.1

Матрица ошибок

	Фактический класс Q=1	*Фактический класс Q=0*
Предсказанный класс $\hat{Q}=1$	***TP*** Истинно положительные (True Positive)	***FP*** Ложно положительные (False Positive)
Предсказанный класс $\hat{Q}=0$	***FN*** Ложно отрицательные (False Negative)	***TN*** Истинно отрицательные (True Negative)

TP – число правильно определенных меток класса *Q=1*;
TN – число правильно определенных меток класса *Q=0*;
FN – число неправильно определенных примеров, когда класс документа *Q=1* проклассифицирован как $\hat{Q}=0$ (ошибка первого рода);
FP – число неправильно определенных меток, когда класс документа *Q=0* проклассифицирован как $\hat{Q}=1$ (ошибка второго рода).

Таким образом, матрица ошибок показывает, сколько документов интересующего нас («положительного») класса *Q=1* были отнесены к («отрицательному») классу *Q=0* (и наоборот).

В теории вероятностей и математической статистике *FP* (*False Positive*) и *FN* (*False Negative*) называются соответственно ошибками I-го рода (ложная тревога, ложно положительное срабатывание, т.е. отвергается правильная гипотеза H_0) и II-го рода (пропуск цели, ложно отрицательное срабатывание, т.е. принимается неправильная гипотеза H_0).

На основе матрицы ошибок рассчитываются следующие показатели качества:

1) *Правильность* δ (*Accuracy*, также употребляются термины верность, аккуратность, точность) и *ошибка классификации* Δ:

$$\delta = \frac{TP+TN}{TP+FN+FP+TN} \quad (1.1)$$

$$\Delta = 1 - \delta = \frac{FN+FP}{TP+FN+FP+TN} \quad (1.2)$$

2) *Полнота R* (*Recall*) и *точность P* (*Precision*). Точность классификатора – это доля документов действительно принадлежащих данному классу относительно всех документов, которые классификатор отнес к этому классу (т.е. доля документов, которым присвоена метка $\hat{Q}=1$ и при этом они действительно относятся к этому классу $Q=1$). Полнота системы – это доля правильно проклассифицированных документов, принадлежащих классу $Q=1$, относительно всех документов этого класса в экзаменационной выборке. Таким образом, полнота показывает долю правильно распознанных положительных примеров – *TPR* (*True Positives Rate*)**.**

$$R = \frac{TP}{TP+FN} \text{ - полнота} \quad (1.3)$$

$$P = \frac{TP}{TP+FP} \text{ - точность} \quad (1.4)$$

Увеличение полноты обычно приводит к потере точности и наоборот. Поэтому часто используется комбинация этих двух показателей, которые объединяются в *F-меру*:

$$F = \frac{(\beta^2+1)PR}{\beta^2 P+R} \quad (1.5)$$

Здесь β – эмпирически определяемый параметр, позволяющий задавать *полноте* и *точности* разные веса. Чаще всего на практике параметр β принимают равным единице, такую меру называют F_1:

$$F_1 = \frac{2PR}{P+R} \quad (1.6)$$

3) *ROC-кривая* (*Receiver Operating Characteristic*, кривая ошибок).

По таблице 1 определим специфичность *Sp*, которая отражает долю правильно распознанных отрицательных примеров ($Q=0$):

$$S_p = \frac{TN}{TN + FP} \quad (1.7)$$

Чувствительность *Se* (или полнота, см. формулу 3) и специфичность *Sp* можно рассматривать как *правильность* применительно к отдельному классу.

Рассчитаем также *FPR* (*False Positive Rate*) – долю ложных положительных классификаций, которая показывает, сколько от общего числа документов $\hat{Q}=0$, оказались предсказанными неверно.

$$FPR = 1 - S_p = \frac{FP}{TN + FP} \quad (1.8)$$

Определив чувствительность и специфичность, можно отразить результаты проверки классификаторов в двумерном *ROC*-пространстве, откладывая по оси ординат значения *Se*, а по оси абсцисс – значения (1 – *Sp*). *ROC*-кривая (*ROC = Receiver Operating Characteristic*, «кривая ошибок») представляет зависимость доли истинно положительных результатов классификации (true positive rate TPR, чувствительности) от доли ложно положительных оценок, равной (1 – *Sp*). Или, используя ранее введенные обозначения, ROC-кривая отражает взаимосвязь *TPR* от *FPR*. При ее построении изменяется порог отсечения от 0 до 1 (например, с шагом 0.01), рассчитываются значения чувствительности *Se* и специфичности *Sp*, сроится график зависимости.

ROC-кривая всегда начинается из точки (0,0) и заканчивается в (1,1). Чем больше площадь под кривой (чем выше лежит кривая), тем лучше качество классификации. На рисунке 1 представлен пример успешной классификации. Если площадь под кривой равна 0.5, то это соответствует случайному выбору класса.

Минимизировать издержки ошибок классификации можно с помощью выбора точки отсечения (*Cut-off Point*) – порогового значения, разделяющего классы. Цель *ROC*-анализа заключается в том, чтобы подобрать такое значение точки отсечения, которое позволит модели с наибольшей точностью распознавать положительные или отрицательные примеры и выдавать наименьшее количество ложноположительных или ложноотрицательных ошибок.

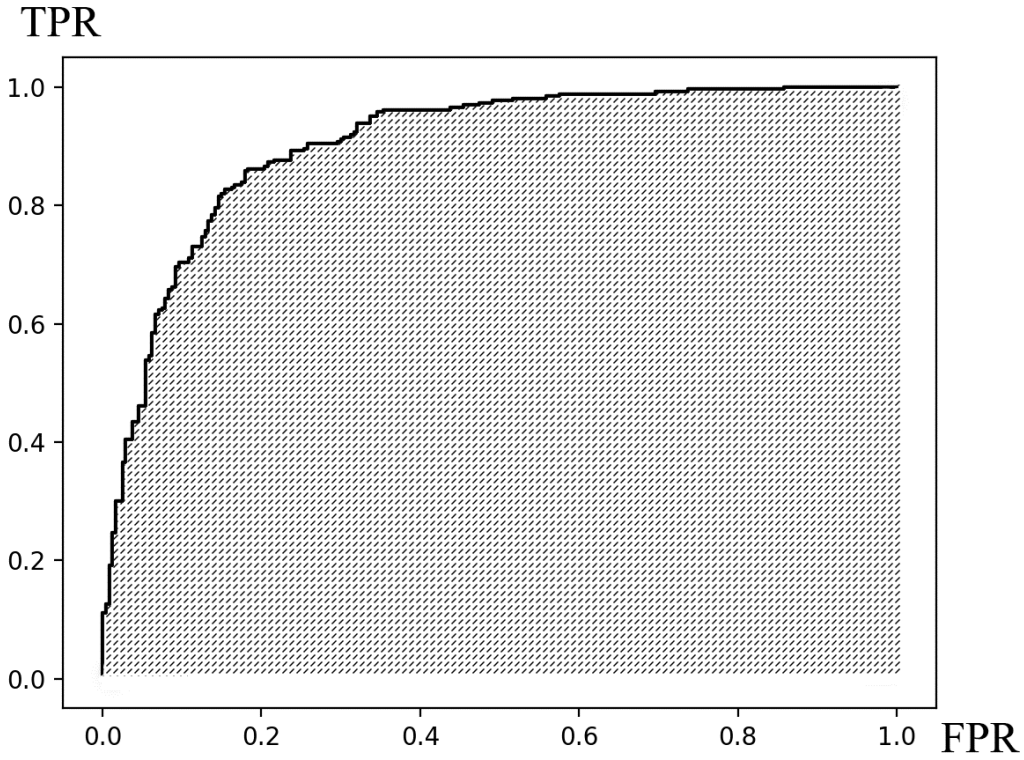

Рис. 1. Кривая ошибок и площадь под кривой

Для сравнения эффективности различных классификаторов удобно применять некоторую интегральную характеристику, в качестве которой используют площадь *AUC* (*Area Under Curve*), вычисленную под *ROC*-кривой.

Часто на практике обучение бинарного классификатора проводится на несбалансированных выборках, которые имеют различное число примеров из каждого класса (*Imbalanced Dataset*). При обработке фактографических данных такие выборки возникают, например, в задачах кредитного скоринга. В этом случае количество примеров, соответствующих «плохим» заемщикам, составляет малую часть выборки. Несбалансированность также возникает при фильтрации спама, выявлении сетевых атак, выделении «разыскиваемых» лиц на изображениях видеокамер и т.п. Основная проблема обучения на несбалансированных выборках заключается в том, что бинарные классификаторы на реальных данных будут «иметь склонность» относить новые наблюдения к «большому» классу. В таких случаях необходимо тщательно выбирать способы обучения классификатора, в частности можно рекомендовать использовать метод бутстреп (*Bootstrap*), позволяющий увеличить (сократить) размер класса путем статистического моделирования. Кроме того, надо учитывать, что ряд показателей качества дает искажения на

несбалансированных выборках (например, *ROC*-кривая, которая использует при расчете долю неверно распознанных наблюдений относительно общего числа отрицательных, а также показатель *accuracy*).

1.1.3 Базовые методы классификации

Рассмотрим базовые методы классификации, применяющие различные подходы к построению решающего правила [1].

В ***наивном байесовском методе*** (НБМ) используется предположение о независимости признаков между собой, поэтому вероятность принадлежности объекта $\vec{X}$, описываемого признаками $x^{(i)}$ ($i=1,..,M$), к классу Q вычисляется по формуле:

$$P(Q|\vec{X}) = P(Q)\prod_{i=1}^{M} P\left(x^{(i)}\middle|Q\right) \quad (1.9)$$

Здесь P(Q) - априорная вероятность принадлежности объекта $\vec{X}$ к классу Q. $P\left(x^{(i)}\middle|Q\right)$– условная вероятность того, что признак $x^{(i)}$ появится в объектах класса (т.е. оценка, насколько признак характерен для класса).

В зависимости от конкретной задачи, метод наивного Байеса имеет несколько реализаций.

Если признаки объектов независимы и нормально распределены, то вероятность $P\left(x^{(i)}\middle|Q\right)$ может быть оценена по нормальному (Гауссовскому) распределению:

$$\hat{P}\left(x^{(i)}\middle|Q\right) = \frac{1}{\sqrt{2\pi\sigma_Q^2}}\exp\left(-\frac{(x_i-\mu_Q)^2}{2\sigma_Q^2}\right) \quad (1.10)$$

Параметры μ_Q и σ_Q оцениваются с использованием максимального правдоподобия. Такая реализация метода наивного Байеса называется Гауссовской.

В задачах анализа текстов обычно используется мультиномиальная реализация метода наивного Байеса, которая учитывает количество повторений каждого слова, но не учитывает, каких слов нет в документе[1]. При этом, расчет оценок $\hat{P}\left(x^{(i)}\middle|Q\right)$ и $\hat{P}(Q_g)$ проводится по обучающей выборке следующим образом:

$$\hat{P}(Q_g) = \frac{N_g}{N} \quad (1.11)$$

[1] https://logic.pdmi.ras.ru/~sergey/teaching/mlstc12/sem01-naivebayes.pdf

$$\hat{P}\left(x^{(i)}\middle|Q\right) = \frac{N_{ig}}{N_g} \qquad (1.12)$$

Здесь N_g – число документов обучающей выборки, принадлежащих классу Q_g, N_{ig} – частота встречаемости слова i в документах класса Q_g в обучающей выборке.

Многие слова из общего словаря могут не встречаться в документах класса, чтобы избежать нулевых множителей в формуле (12) используется уточненная формула, где α – параметр сглаживания:

$$\hat{P}\left(x^{(i)}\middle|Q\right) = \frac{\alpha + N_{ig}}{\alpha M + N_g} \qquad (1.13)$$

Обычно α = 1, что соответствует сглаживанию по Лапласу. Если α > 1, такое сглаживание называется сглаживанием по Лидстоуну[2]

В ***методе ближайшего соседа*** (МБС) решение об отнесении документа к тому или иному классу принимается не по всей выборке, а лишь по документу, который находится в непосредственной близости от классифицируемого объекта.

Для определения степени соседства используются меры близости. Чаще всего применяется евклидово расстояние между двумя документами $\vec{X}_j$ и $\vec{X}_l$:

$$d\left(\vec{X}_j, \vec{X}_i\right) = \sqrt{\sum_{i=1}^{M} (x_j^{(i)} - x_l^{(i)})^2} \qquad (1.14)$$

или косинусная мера близости:

$$d\left(\vec{X}_j, \vec{X}_i\right) = \frac{\sum_{i=1}^{M} x_j^{(i)} x_l^{(i)}}{\sqrt{\sum_{i=1}^{M} (x_j^{(i)})^2 \sum_{i=1}^{M} (x_l^{(i)})^2}} \qquad (1.15)$$

где $x_j^{(i)}$ – вес i-го термина в j-ом документе; $x_l^{(i)}$ – вес i-го термина в l-ом документе.

В ***методе к-БС*** определяется не один, а группа документов-соседей, наиболее близких к классифицируемому тексту. Число соседей ***к*** является параметром, настраиваемым на стадии обучения (или задаваемым экспертно). Решение принимается путем простого голосования ***к*** соседей.

В ***методе деревьев решений*** (*МДР, Decision Trees, DT*) проводится последовательное разделение исходного множества документов на основе значений специально выбираемого термина $x^{(i)}$. Строится дерево, содержащее *нетерминальные узлы* (узлы проверок, «родительские» вершины), в которых происходит разбиение по

[2] https://scikit-learn.org/0.21/modules/naive_bayes.html

термину $x^{(i)}$, и *терминальные узлы* (узлы ответа), в которых должны находиться тексты одного класса. В общем случае предикторы $x^{(i)}$ могут быть измерены в номинальной, порядковой и количественной шкале.

На практике чаще всего используются бинарные деревья решений, в которых принятие решения в нетерминальном узле осуществляется путем сравнения веса термина с пороговым значением. При этом нетерминальный узел делится на две части – узел, в котором значение признака больше или равно величине порога (потомок – *right*) и узел, в котором значение признака меньше порога (потомок – *left*).

Выбор способа выявления *наиболее информативного признака*, по которому проводится разбиение, является ключевым элементом алгоритма. *МДР* относится к так называемым «жадным» алгоритмам (*Greedy Algorithms*), для которых невозможно вернуться на предыдущий шаг и изменить разбиение, поменяв ранее выбранный $x^{(j)}$.

В настоящее время разработано значительное число различных алгоритмов обучения МДР, отличающихся способом выбора классообразующего признака $x^{(j)}$. Алгоритмы обычно измеряют однородность целевой переменной на подмножествах, используя некоторые метрики (критерии), а затем определяют качество разбиения.

Пусть в результате очередного разбиения по признаку $x^{(j)}$ в сравнении с пороговым значением t, подвыборка T_m разделилась на подвыборки T_l и T_r.

Для определения качества такого разбиения обычно используется следующий критерий ошибки, который необходимо минимизировать:

$$Q(T_m,j,t) = \frac{|T_l|}{|T_m|} H(T_l) + \frac{|T_r|}{|T_m|} H(T_r) \qquad (1.16)$$

Здесь $|T_l|$, $|T_r|$ и $|T_m|$ - размеры соответствующих подвыборок, $H(T_l)$ – критерий, который измеряет качество подмножества T_l – насколько сильный разброс ответов имеет место в подвыборке T_l. Аналогично $H(T_r)$ измеряет разброс ответов в подвыборке T_r.

В качестве таких критериев H обычно используют критерий Джини (*Gini impurity*) и прирост информации (*Information gain*).

Критерий Джини

$$Gini\ (T) = 1 - \sum_{k=1}^{K} p_k^2 \qquad (1.17)$$

где p_k - вероятность (относительная частота) класса k в рассматриваемой подвыборке. Критерий Джини может быть интерпретирован как вероятность ошибки случайного классификатора, который выдает случайный класс $k \in \{1,\ldots,K\}$, при этом, вероятность выдать класс $k = p_k$.

Прирост информации

$$IG\ (T) = -\sum_{k=1}^{K} p_k\, ln(p_k) \qquad (1.18)$$

Данный критерий показывает меру отличия распределения классов от вырожденного – от распределения, в котором энтропия равна 0. В таком распределении нет ничего неожиданного, и нам всегда известен класс. Если распределение равномерное (т.е. вероятность получить каждый класс в этой выборке одинаковая), то энтропия будет максимальной – мы не можем предсказать, что мы получим.

Таким образом, в качестве основных настраиваемых параметров в *МДР* используются:

- *глубина дерева* – количество узлов, расположенных на разных уровнях;

- *критерий выбора наиболее информативного признака.*

2 ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ К ЛАБОРАТОРНОЙ РАБОТЕ №2 «ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ТЕКСТОВЫХ ДАННЫХ»

Построению классификатора предшествует предварительная обработка текстовой информации. Основной особенностью работы с текстовыми данными является большая размерность пространства признаков (терминов). Вместе с тем, не все признаки несут в себе полезную информацию, способствующую точному анализу, что в первую очередь связано с особенностями естественного языка, на котором написаны тексты. Предварительная обработка существенным образом влияет на результирующую точность и включает совокупность специальных процедур.

Наиболее важной задачей, после того как проведено разбиение текста на отдельные слова (векторизация текста), является сокращение числа слов и их взвешивание. Одним из основных способов решения задачи уменьшения размерности является отсечение стоп-слов (стоп-слова – слова, не несущие смысловой информации, например, местоимения, предлоги, артикли и т.д.). Еще

один широко используемый способ сокращения словаря выборки - операции выделения основы слова – лемматизация и стемминг.

Лемматизация – процесс приведения слова к его нормальной форме. Нормальной формой слова является форма единственного числа в именительном падеже для русского языка, и форма единственного числа в инфинитиве для английского языка.

Стемминг – процесс нахождения основы слова. При этом основа слова не обязательно должна совпадать с морфологическим корнем слова.

После проведения векторизации, стемминга (или лемматизации), отсечения стоп-слов одним из наиболее важных этапов предварительной обработки является взвешивание терминов, или, другими словами, определение их меры информативности. Чаще всего для взвешивания используют *tf*-взвешивание (*term-frequency*):

$$x_j^{(i)} = f_{ij}, \tag{2.1}$$

которое учитывает лишь частоту *f* появления термина *i* в документе *j*, а также *tf–idf* - взвешивание (*term frequencies – inverse document frequencies*), в котором вес слова *i* в документе *j* пропорционален числу вхождений слова в данный документ и обратно пропорционален общему числу документов в выборке, в которых также содержится это слово:

$$x_j^{(i)} = f_{ij} \log\left(\frac{N}{N_i}\right), \tag{2.2}$$

Здесь N_i – число документов выборки, в которых встретилось слово *i*.

3 ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ К ЛАБОРАТОРНОЙ РАБОТЕ №3 «КЛАССИФИКАЦИЯ ТЕКСТОВЫХ ДАННЫХ»

3.1 Методы многоклассовой классификации

Рассмотрим принцип принятия решения в наиболее эффективных методах, используемых для многоклассовой классификации.

Метод случайного леса (*МСЛ*, *Random forest, RF*). Метод основан на бутстреп - процедуре генерации повторных выборок из исходного набора данных (случайное извлечение с возвращением) Идея *МСЛ* заключается в построении большого числа деревьев

решений, каждое из которых формируется на выборке, полученной с помощью бутстрепа. Деревья решений весьма чувствительны к небольшому изменению обучающих данных, т.к. обладают высокой дисперсией решений (при низком смещении). Это означает, что деревья решений, получаемые на бутсреп-выборках, будут существенно различаться. В отдельности полученные деревья вряд ли окажутся точными. Однако при их объединении в коллектив результаты существенно улучшаются. Среднее коллективное решение о метке класса, обладающее более низкой дисперсией, называется *бэггингом* (сокр. от ***b**ootstrap **agg**regat**ing***).

Для усиления различий (*Diversity*) в *МСЛ* разбиение в каждом узле осуществляется на фиксированном числе терминов. Это означает, что для каждого узла дерева случайным образом выбирается m из M подлежащих рассмотрению признаков-предикторов (информативных терминов). Таким образом, с вероятностью $(M-m)/M$ блокируются потенциально доминирующие признаки, стремящиеся войти в каждое дерево. Если не использовать фиксированный набор признаков, то, несмотря на использование бутстреп-выборок, все деревья решений в итоге будут достаточно похожи друг на друга. Как следствие, значение *Diversity* станет незначительным, предсказания коррелированными, а снижение дисперсии решений небольшим. Благодаря частичному блокированию наиболее информативного термина, разбиение нетерминальных узлов будет проводиться с помощью других терминов и вариации в получаемых деревьях возрастут.

Выбор малого значения m при построении случайного леса полезен при наличии большого числа сильно коррелирующих признаков. Если случайный лес строится с использованием $m=M$, то вся процедура сводится к бэггингу.

Настраиваемыми параметрами в *МСЛ* являются:

- *число деревьев решений*, входящих в коллектив (ансамбль, комитет),

- *глубина дерева* – количество узлов, расположенных на разных уровнях;

- *критерий выбора НИТ* - критерий Джини, критерий максимума энтропии

Основная идея ***метода опорных векторов* (*Support Vector Machines – SVM*)** заключается в поиске разделяющей гиперплоскости с максимальным зазором в пространстве [6]. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей имеющиеся классы. Разделяющей поверхностью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем

больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

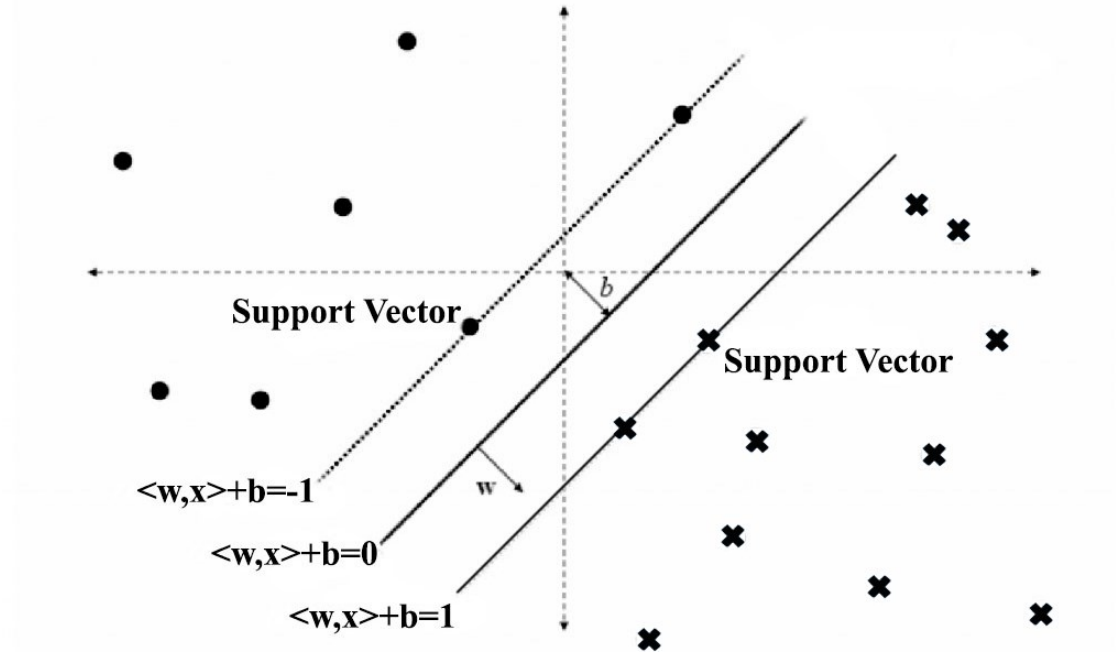

Рис 2. Метод опорных векторов

По обучающей выборке в ***SVM*** строится классифицирующая функция $\hat{a}(x)$ в виде:

$$\hat{a}(x) = sign(\langle w, x\rangle + b), \qquad (3.1)$$

где <w,x>— скалярное произведение, w — нормальный вектор к разделяющей гиперплоскости, b — вспомогательный параметр. Документы, для которых $\hat{a}(x)$ = 1 попадают в один класс, а документы с $\hat{a}(x)$ = -1 –в другой. Выбор именно такой функции неслучаен: любая гиперплоскость может быть задана в виде $<w,x> + b = 0$ для некоторых w и b .

Далее выбираются такие w и b, которые максимизируют расстояние до каждого класса. Это расстояние равно $\frac{1}{||\boldsymbol{w}||}$. Проблема нахождения максимума $\frac{1}{||\boldsymbol{w}||}$ эквивалентна проблеме определения минимума $\|\boldsymbol{w^2}\|$. Данная задача может быть представлена в виде задачи оптимизации:

$$\begin{cases} \frac{1}{2}\|\boldsymbol{w^2}\| \underset{w,b}{\rightarrow} min \\ y_j(< w, x_j > +b) \geq 1 \end{cases} \qquad (3.2)$$

Здесь $y_j \in \{-1; 1\}$ - метка класса, принимающая одно из двух возможных значений.

Для случая произвольных данных $y_j(< w, x_j > + b) \geq 1$ не может быть выполнено для всех документов, в этом случае добавляется ослабляющие коэффициенты $\varepsilon_j \geq 0$, а также коэффициент регуляризации $C \geq 0$, который определяет компромисс между количеством ошибок на обучающей выборке и простотой линейного решающего правила. Более подробно про регуляризацию см. раздел «Регуляризация модели». Таким образом, критерий оптимизации перепишется в виде:

$$\begin{cases} \frac{1}{2}||w||^2 + \mathrm{C}\sum_{j=1}^{N} \varepsilon_j \underset{w,b,\varepsilon}{\longrightarrow} \min \\ y_j\big(\langle w, x_j\rangle + b\big) \geq 1 - \varepsilon_j \\ \varepsilon_j \geq 0, \end{cases} \tag{3.3}$$

Задачу условной оптимизации (3.3) можно переписать как задачу безусловной оптимизации:

$$\varepsilon_j \geq \max\Big(0{,}1 - \ y_j\big(\langle w, x_j\rangle + b\big)\Big) \triangleq l_{hinge}\Big(y_j, f\big(x_j\big)\Big) \tag{3.4}$$

В задаче (3.3) требуется минимизировать значения ε_j. При фиксированных ***w, b*** этот минимум достигается при

$$\varepsilon_j = l_{hinge}\Big(y_j, f\big(x_j\big)\Big)$$

Таким образом, задача (3.3) эквивалентна следующей задаче:

$$\sum_{j=1}^{N} l_{hinge}\Big(y_j, f\big(x_j\big)\Big) + \frac{1}{2C}\|w\|^2 \underset{w,b}{\longrightarrow} min \tag{3.5}$$

В этой задаче первое слагаемое соответствует ошибке классификатора на обучающей выборке, измеряемой с помощью функции потерь l_{hinge}. Второе слагаемое является регуляризатором, штрафующим излишнюю перенастройку классификатора на обучающую выборку. [7]

Таким образом, настраиваемыми параметрами в методе опорных векторов являются вид функции потерь и вид регуляризации (см. раздел «Регуляризация модели»).

Логистическая регрессия (*логит-регрессия,* ЛР). Это разновидность множественной линейной регрессии, которая описывает связь между независимыми переменными (в нашем случае – терминами) и зависимой переменной (меткой класса).

Ставится задача определить вероятности принадлежности вектора $\vec{X}$ к конкретному классу *y=Q*. Далее рассмотрим наиболее

простой случай – бинарную логистическую регрессию (Q = {0, 1}). Вместо предсказания номинальных величин – меток классов, будем предсказывать (непрерывную) переменную *y*, принимающую значения в диапазоне [0, 1] в зависимости от значений независимых переменных.

Будем искать решение в виде регрессионной зависимости:

$$z = f(X, B) = b_0 + b_1 x^{(1)} + b_2 x^{(2)} + \ldots + b_M x^{(M)} \qquad (3.6)$$

Для приведения этой величины в диапазон [0, 1] используем логит-преобразование:

$$P = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \qquad (3.7)$$

Здесь P – вероятность того, что произойдет определенное событие (например, документ $\vec{X}$ относится к классу *Q=1*); z – уравнение множественной регрессии, заданное формулой (25).

Таким образом, в ЛР оценивается не само значение зависимой количественной переменной *y*, а вероятность принадлежности текста, заданного своими терминами, к конкретному классу. Зависимость (26) – логистическая (сигмоидальная) функция, отражающая связь между вероятностью события P и величиной *z*, показана на рисунке 3.

В отличие от множественной линейной регрессии, в которой для определения оценок параметров чаще всего используется метод наименьших квадратов, в ЛР применяется метод максимального правдоподобия и рассчитывается *функция правдоподобия* (*ФП*, *Likehood Function*). Для облегчения поиска максимума вычисляется логарифм функции правдоподобия. Максимизация *ФП* проводится с помощью градиентных методов или метода Ньютона.

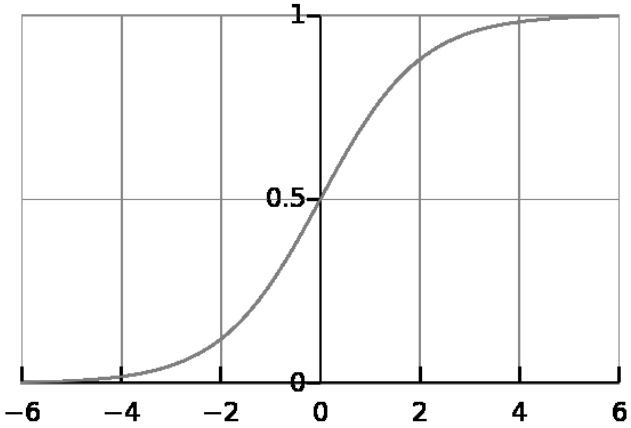

Рис. 3. Логистическая зависимость

При мультиномиальной логистической регрессии переменная *y* может относиться к произвольному числу классов. Обычно такие задачи сводятся к ранее рассмотренному случаю: для каждого класса строится уравнение бинарной логистической регрессии.

В качестве настраиваемых параметров в логистической регрессии используются:

Метод нахождения экстремума – градиентный спуск, метод Ньютона и другие

Вид регуляризации – L_1-регуляризация (лассо) или L_2-регуляризация (гребневая) – см. раздел «Регуляризация модели».

Таким образом, логистическая регрессия, как и метод деревьев решений, является линейным классификатором, проводя разбиение текстов в многомерном пространстве терминов с помощью плоскостей.

3.1.1 Регуляризация модели

Одной из существенных проблем, возникающих при настройке параметров классификаторов, является переобучение, при котором ошибка на обучающих выборках практически равна нулю, однако наблюдаются достаточно большие значения ошибки обобщения, получаемых на тестовых выборках. Этот эффект свидетельствует о создании избыточно сложной модели. В случае с логистической регрессией переобучение заключается в слишком сильной подгонке неизвестных параметров (коэффициентов регрессии) b_i в формуле (23) к имеющимся обучающим данным. Чаще всего о переобучении свидетельствуют очень большие величины b_i.

Для нивелирования эффекта переобучения разработаны различные подходы, один из них предусматривает проведение регуляризации. Для ЛР регуляризация заключается во введении нового (требующего предварительного задания) параметра λ (*regularization parameter*). Это позволяет не допускать появление очень больших значений b_i за счет добавления штрафа (за рассчитанные коэффициенты) к функции ошибки.

При построении регрессии расчет ошибок проводится различными способами. При минимизации суммы квадратов значений b_i (мера L_2) получаем *гребневую* (*ридж, Ridge*) регуляризацию. В отечественной специализированной литературе такой подход обычно называется *регуляризацией по Тихонову – Tikhonov regularization*. При расчете абсолютных значений ошибок (мера L_1) получаем *лассо* регуляризацию (*Lasso – Least Absolute Shrinkage and Selection Operator*).

При использовании лассо регуляризации часть коэффициентов b_i обращается в *0*, т.е. происходит сокращение признакового пространства. Гребневая регуляризация ограничивает значения b_i, но не приводит к их полному обнулению. Основной недостаток лассо-регуляризации заключается в осложнении использования численных

методов для расчета градиента. Ридж–регуляризация не имеет подобных ограничений.

Применение регуляризации способно улучшить точность классификации (особенно в случае наличия малой выборки и большого числа признаков) и повысить устойчивость метода по отношению к изменениям в исходных данных. Однако при этом возникает необходимость задания дополнительного параметра λ, значение которого должно быть определено до проведения регуляризации.

3.1.2 Микро- и макро-усреднение показателей качества

В лабораторной работе №1 были рассмотрены несколько базовых классификаторов. Расчет показателей качества в случае многоклассовой классификации, как правило, сводится к подсчету показателей качества, вычисленных для бинарного случая. При этом используется *микро- и макро-усреднение*. Для выборки из G классов необходимо рассмотреть G двухклассовых задач, каждая из которых заключается в отделении своего класса от остальных. При этом используются различные характеристики (TP, FP, и т.д.). При *микро-усреднении* сначала эти характеристики усредняются по всем классам, а затем вычисляется итоговые показатели качества, например, точность, полнота или F-мера. При *макро-усреднении* сначала определяется итоговый показатель для каждого класса, а затем результаты усредняются по всем классам.

Если классы несбалансированы, то при микро-усреднении они практически никак не будут влиять на результат, поскольку их вклад в средние значения *TP, FP, FN* и *TN* будет незначителен. В случае же с макро-усреднением расчет проводится для величин, которые уже не чувствительны к соотношению размеров классов, и поэтому каждый класс внесет равный вклад в итоговую метрику.

4 ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ К ЛАБОРАТОРНОЙ РАБОТЕ №4 «КЛАСТЕРИЗАЦИЯ ДАННЫХ»

4.1 Задача кластеризации данных

Кластеризация является важным этапом разведочного анализа текстовых данных, позволяя определять группы (кластеры) похожих документов, оценивать размер и компактность кластеров [1,4,5]. Кластеризация также дает возможность проведения редукции (сокращения) данных, которая предусматривает определение

нескольких наиболее типичных представителей для каждого кластера, обнаружения нетипичных объектов, создания иерархии кластеров. Так как кластеризация представляет собой обучение без учителя («*Unsupervised learning*»), то не существует единственно правильного разбиения исследуемого множества на группы. Ценность группировки заключается в получении «разведочных» оценок о структуре выборки и ее особенностях. В ряде случаев при совпадении результатов кластеризации, выполненной разными методами, можно говорить о выявлении закономерностей, которые характеризуют исследуемую выборку (выборки) [4].

Имеется два подход к кластеризации: «*плоская*» (*Flat clustering*) и *иерархическая* (*Hierarchical clustering*). В «плоской» кластеризации кластеры строятся как однородные группы, между которыми отсутствует связь. При выполнении иерархической кластеризации проводится построение иерархии (дерева) вложенных кластеров на основе вычисления специальной меры близости между группами.

По аналогии с методами классификации выделяют *жесткую* и *мягкую* кластеризацию. Жесткое разделение документов на кластеры предполагает, что каждый документ принадлежит только одной группе (непересекающиеся кластеры). В случае, когда текст может быть отнесен к нескольким кластерам на равных основаниях, он приписывается, например, к кластеру с меньшим индексом. При мягком разделении на кластеры один и тот же документ может быть приписан к нескольким кластерам (пересекающиеся кластеры).

Дадим формальную постановку задачи «плоской» кластеризации с жестким отнесением документов к кластерам.

Имеется $\mathbf{X}$ – множество документов, каждый документ $\vec{X}$ представляется в виде вектора в пространстве терминов (M – размерность пространства, которая определяется словарем терминов). Введена мера близости $\boldsymbol{\rho}$ между документами. Требуется разбить множество $\mathbf{X}$ на подмножества, называемые кластерами, так что бы кластер состоял из документов близких по мере $\boldsymbol{\rho}$, при этом документы разных кластеров существенно отличились.

В общем случае количество кластеров неизвестно и определяется в ходе кластеризации. В качестве $\boldsymbol{\rho}$ в задачах анализа текстовых документов чаще всего используются евклидово расстояние или косинусная мера близости (см. формулы (14) и (15)).

Оценка качества кластеризации менее формализованная процедура, чем при классификации. Обычно для этого используются различные критерии, основанные на расстояниях между всеми элементами кластера (кластеров) или расстояниях между некоторыми «типичными» представителями кластеров – центроидами [5]. Наиболее часто применяются средняя сумма внутриклассовых

попарных расстояний, средняя сумма межклассовых попарных расстояний, среднее расстояние до центроида от элементов кластера, расстояние между центроидами различных кластеров.

4.1.1 Иерархические методы кластеризации

Существует два подхода, реализующих иерархическую кластеризацию, – агломеративный и дивизимный. Под *агломеративным* подходом подразумевается процедура последовательного формирования сначала кластеров, состоящих из групп наиболее тесно связанных объектов, а затем кластеров следующего уровня путем присоединения “тяготеющих” к ним объектов. Такая процедура при движении от вершин к корню итерируется, пока кластеризация не охватит всего множества объектов. При *дивизимной* иерархической процедуре на 1-м уровне иерархии находится один класс, включающий все объекты (корень дерева), а на наивысшем уровне число классов равно числу классифицируемых объектов. Агломеративная и дивизимная процедуры соответственно представлены на рис. 3.

Иерархическая структура объектов геометрически представляется в виде *дендрограммы* (иерархия с численными уровнями). Такое представление результата работы алгоритма наглядно иллюстрирует, как N объектов распределились по группам, находящимся на последовательных уровнях иерархии (каждый уровень соответствует числовому порогу, указывающему степень сходства объектов в группах этого уровня).

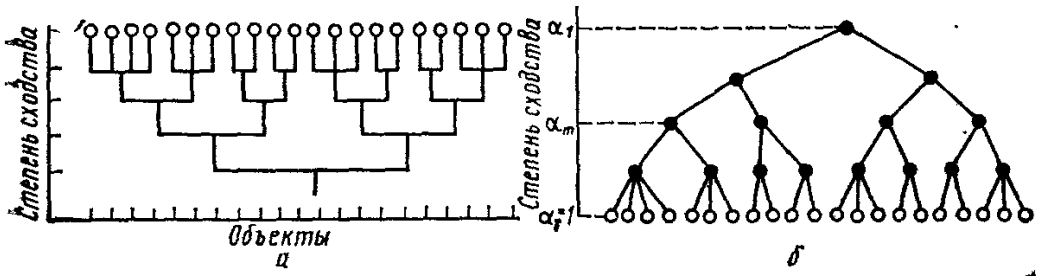

Рис. 4. Агломеративная и дивизимная процедуры

При объединении кластеров в иерархических процедурах можно использовать различные меры близости между ними.

- **Одиночная связь (метод ближайшего соседа).** В этом методе расстояние между двумя кластерами определяется расстоянием между двумя наиболее близкими объектами (ближайшими соседями) в различных кластерах (рисунок 5, расстояние 2).

- **Полная связь (метод наиболее удаленных соседей).** В этом методе расстояния между кластерами определяются наибольшим расстоянием между любыми двумя объектами в различных кластерах (т.е. “наиболее удаленными соседями” см. рисунок 5, расстояние 3).
- **Невзвешенное попарное среднее.** В этом методе расстояние между двумя различными кластерами вычисляется как среднее расстояние между всеми парами объектов в них.
- **Взвешенное попарное среднее.** Метод идентичен методу невзвешенного попарного среднего, за исключением того, что вычисляется размер соответствующих кластеров (т.е. число объектов, содержащихся в них) и эта величина используется в качестве весового коэффициента. Предлагаемый метод обычно используется, когда предполагаются неравные размеры кластеров.
- **Невзвешенный центроидный метод.** В этом методе расстояние между двумя кластерами определяется как расстояние между их центрами тяжести (рисунок 5, расстояние 1). Центр тяжести или центроид рассчитывается по следующей формуле:

$$\overrightarrow{C_k} = \frac{1}{N_k}\sum_{j=1}^{N_k}\overrightarrow{X_j} \qquad (4.1)$$

Здесь $\overrightarrow{C_k}$ – центроид кластера, $\overrightarrow{X_j}$ - объект данного кластера

- **Взвешенный центроидный метод.** Данный метод идентичен предыдущему, за исключением того, что при вычислениях центроидов используются веса для учёта разницы между размерами кластеров (т.е. количеством объектов в них).
- **Метод Уорда (*Ward*).** Этот метод отличается от всех других методов, поскольку он использует методы дисперсионного анализа для оценки расстояний между кластерами. Метод минимизирует сумму квадратов расстояний для любых двух (гипотетических) кластеров, которые могут быть сформированы на каждом шаге. В целом метод представляется очень эффективным, однако в нем присутствует тенденция к созданию кластеров малого размера.

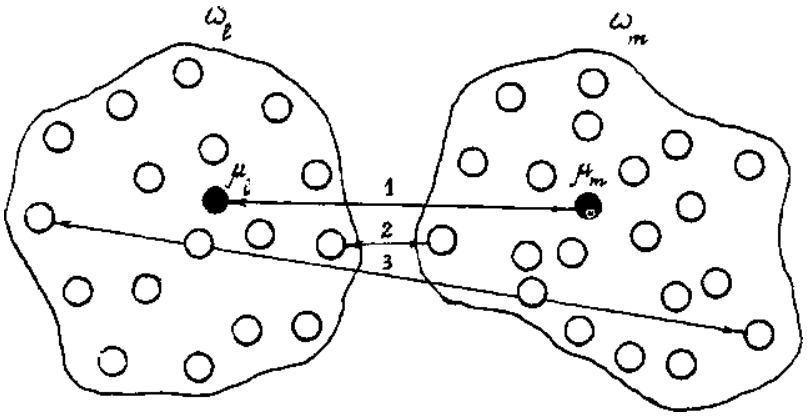

Рис. 5. Способы объединения кластеров

Использование различных метрик расстояния и способов объединения объектов в кластеры будет порождать заметно различающиеся результаты. В этой связи обратим внимание на такую важную характеристику кластеризации (классификации) как устойчивость [4]. Здесь под устойчивостью кластеризации (классификации) понимается получение почти идентичного разбиения выборки на кластеры (классы), применяя различные методы. Таким образом, устойчивость результатов свидетельствует, что с помощью кластерного анализа удалось нащупать естественные группировки, имеющиеся в генеральной совокупности, и полученные закономерности не меняются существенным образом при извлечении новых выборок.

4.1.2 Метод k-средних

Суть алгоритма *k*-средних (*k-means*) заключается в минимизации расстояний от всех элементов кластеров до их центроидов [8].

$$\sum_{g=1}^{k}\sum_{j=1}^{N_g}(\vec{X}_j - \vec{C}_g)^2 \to min \qquad (4.2)$$

где k – заданное число кластеров, N_g – множество всех элементов кластера g, $\vec{C}_g$– центроид g-го кластера.

Основными параметрами метода k-средних являются мера близости и число кластеров, которые должны быть заданы заранее. Общая схема алгоритма состоит из следующих шагов:

1. Выбираем k начальных центроидов (случайным образом или экспертно).
2. Для каждого документа с помощью введенной меры близости определяем, к какому центроиду $\vec{C}_g$он ближе и относим к этому кластеру.
3. Пересчитываем центроиды для вновь образовавшихся кластеров.

4. Повторяем шаги 2-3 до тех пор, пока центроиды не перестанут изменяться или не будет выполнено заданное число итераций.

5 ЛАБОРАТОРНЫЕ РАБОТЫ

5.1 Лабораторная работа №1. Бинарная классификация фактографических данных

Цель работы: получить практические навыки решения задачи бинарной классификации данных в среде *Jupiter Notebook*. Научиться загружать данные, обучать классификаторы, проводить классификацию, оценивать точность полученных моделей.

Задание:

1) В среде *Jupiter Notebook* создать новый проект (*Notebook*).
2) Импортировать необходимые для работы библиотеки и модули.
3) Загрузить данные в соответствие с вариантом.
4) Вывести первые 15 элементов выборки (координаты точек и метки классов).
5) Отобразить на графике сгенерированную выборку. Объекты разных классов должны иметь разные цвета.
6) Разбить данные на обучающую (*train*) и тестовую (*test*) выборки в пропорции 75% – 25% соответственно.
7) Отобразить на графике обучающую и тестовую выборки. Объекты разных классов должны иметь разные цвета.
8) Реализовать модели классификаторов, обучить их на обучающем множестве. Применить модели на тестовой выборке, вывести результаты классификации:
   - Истинные и предсказанные метки классов;
   - Матрицу ошибок (*confusion matrix*);
   - Значения полноты, точности, *f1*-меры и аккуратности;
   - Значение площади под кривой ошибок (*AUC ROC*);
   - Отобразить на графике область принятия решений по каждому классу и объекты тестовой выборки

В качестве методов классификации использовать:

а) Метод к-ближайших соседей (*n_neighbors* = {1, 3, 9, 13}).

b) Наивный байесовский метод (гауссовская реализация).

c) Случайный лес (*n_estimators* = {5, 10, 20, 50}).

9) По каждому пункту работы занести в отчет программный код и результат выполнения данного кода.

10) По результатам п.8 занести в отчет таблицу с результатами классификации всеми методами и выводы о наиболее подходящем методе классификации ваших данных.

11) Изучить, как изменится качество классификации, если на тестовую часть выделить 10% выборки, 35% выборки. Для этого повторить п.п. 6 – 10.

Варианты заданий к лабораторной работе приведены в таблице 5.1.

Таблица 5.1

Варианты заданий к лабораторной работе №1

Вариант	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Вид классов	*blobs*	*blobs*	*blobs*	*blobs*	*moons*	*moons*	*moons*	*moons*
Random_state	34	28	41	23	41	23	77	15
cluster_std	1.5	4.5	3	5	-	-	-	-
noise	-	-	-	-	0.25	0.3	0.25	0.2
Centers	2	2	2	2	-	-	-	-

Вариант	**9**	**10**	**11**	**12**
Вид классов	*classification*	*classification*	*classification*	*classification*
Random_state	78	58	15	23
class_sep	0.45	0.7	0.6	0.35

Для всех вариантов, использующих для генерации make_classification, дополнительные параметры*:* *n_features*=2, *n_redundant*=0, *n_informative*=1, *n_clusters_per_class*=1,

Все параметры генераторов выборок *blobs, moons, classification* описаны в документации, ссылки на которую приведены в методических указаниях к данной лабораторной работе.

Контрольные вопросы

1) Постановка задачи классификации данных. Что такое бинарная классификация?

2) Общий алгоритм решения задачи классификации данных.

3) Чем отличаются обучающая и тестовая выборки? Какие существуют способы формирования обучающей и тестовой выборок?
4) Как рассчитываются значения полноты и точности?
5) Как рассчитывается значение площади под кривой ошибок и что оно характеризует?
6) Что показывает и как рассчитывается матрица ошибок?
7) Алгоритм и особенности метода к-ближайших соседей.
8) Алгоритм и особенности метода случайного леса.

5.2 Методические указания к лабораторной работе №1

Основной библиотекой, которую рекомендуется использовать в данном курсе работ, является библиотека *scikit-learn* (*http://scikit-learn.org*). Библиотека *scikit-learn* предоставляет реализацию целого ряда алгоритмов для обучения с учителем (*Supervised Learning*) и обучения без учителя (*Unsupervised Learning*) через интерфейс для языка программирования *Python*.

В данной работе потребуются следующие модули:

confusion_matrix - *http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html*

classification_report – *http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html*

accuracy_score - *http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html*

metrics - *http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score*

train_test_split - *http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html*

LogisticRegression - *http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html*

KNeighborsClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

RandomForestClassifier - http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

MultinomialNB - https://scikit-learn.org/0.21/modules/generated/sklearn.naive_bayes.MultinomialNB.html

А также, в зависимости от варианта задания – один из модулей встроенных генераторов выборок –

make_blobs - http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

make_moons - http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html

make_classification - http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

Кроме того, в работе будет использоваться библиотека *NumPy http://www.numpy.org*, позволяющая работать с многомерными массивами и высокоуровневыми математическими функциями.

Программный код для импорта указанных модулей может выглядеть следующим образом:

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier


from sklearn.naive_bayes import MultinomialNB
```

Для построения графиков рекомендуется использовать библиотеку *matplotlib* и ее модуль *pyplot https://matplotlib.org/users/pyplot_tutorial*

А для отображения на графике области принятия решения - готовую функцию *plot_2d_separator*, которой нужно передать на вход объект classifier – модель классификатора и X – массив входных данных:

```
import matplotlib.pyplot as plt
def plot_2d_separator(classifier, X, fill=False, line=True, ax=None, eps=None):
    if eps is None:
        eps = 1.0 #X.std() / 2.
    x_min, x_max = X[:, 0].min() - eps, X[:, 0].max() + eps
    y_min, y_max = X[:, 1].min() - eps, X[:, 1].max() + eps
    xx = np.linspace(x_min, x_max, 100)
    yy = np.linspace(y_min, y_max, 100)
    X1, X2 = np.meshgrid(xx, yy)
    X_grid = np.c_[X1.ravel(), X2.ravel()]
    try:
        decision_values = classifier.decision_function(X_grid)
        levels = [0]
        fill_levels = [decision_values.min(), 0, decision_values.max()]
    except AttributeError:
        # no decision_function
        decision_values = classifier.predict_proba(X_grid)[:, 1]
        levels = [.5]
        fill_levels = [0, .5, 1]
    if ax is None:
        ax = plt.gca()
    if fill:
        ax.contourf(X1, X2, decision_values.reshape(X1.shape),
        levels=fill_levels, colors=['cyan', 'pink', 'yellow'])
    if line:
        ax.contour(X1, X2, decision_values.reshape(X1.shape), levels=levels,
colors="black")
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)
    ax.set_xticks(())
    ax.set_yticks(())
```

5.2.1 Генерация выборки

Сгенерируем данные, с которыми будем работать. В нашем случае это будут 2 класса, каждый из которых на плоскости имеет форму «шарика» (*blob*). Передадим в качестве параметра *centers = 2* – количество классов-шаров, *random_state = 66* – основа, используемая для генерации случайных чисел, cluster_std = *4* – стандартное отклонение классов, shuffle = *1* – перемешиваем объекты внутри выборки.

В массив с именем X сохраним координаты каждого объекта выборки, а в массив y – метки классов.

```
X, y = make_blobs (centers = 2 , random_state = 66, cluster_std  = 4, shuffle = 1)
```

Посмотрим, что из себя представляют массивы *X* и *y*. Выведем первые 15 элементов каждого из массивов.

```
print ("Координаты точек: ")
print (X[:15])
print ("Метки класса: ")
print (y[:15])
```

```
Координаты точек:
[[-14.80437794  -7.18377798]
 [ -2.60632729   0.20074702]
 [ -7.67410393  -0.92722787]
 [-18.73964269  -1.88968606]
 [ -3.7511755    3.11333437]
 [ -6.13559977  -8.39517379]
 [ -8.10457133   6.15227722]
 [  4.91341461  -2.95516942]
 [ -2.86156125  10.56078045]
 [  0.52303829   3.14548666]
 [  3.53563356   5.80649298]
 [ -6.05018557  -2.10920558]
 [  0.8102374    1.86943425]
 [ -8.0759995   -0.91582206]
 [ -9.37948058 -10.35367349]]
Метки класса:
[0 1 0 0 1 0 1 0 1 1 1 0 1 1 0]
```

Рис. 6. Координаты точек

Первый элемент выборки с координатами *[-14.80437794 -7.18377798]* относится к классу 0,

второй элемент *[-2.60632729 0.20074702]* – к классу 1 и т.д.

Отобразим на графике сгенерированные данные.

В качестве координат точек передадим первый (индекс 0) и второй (индекс 1) столбец массива *X*, для указания цвета точки (параметр *c*) используем метку класса из массива y.

```
plt.scatter (X[:,0], X[:,1], c=y)
plt.show
```

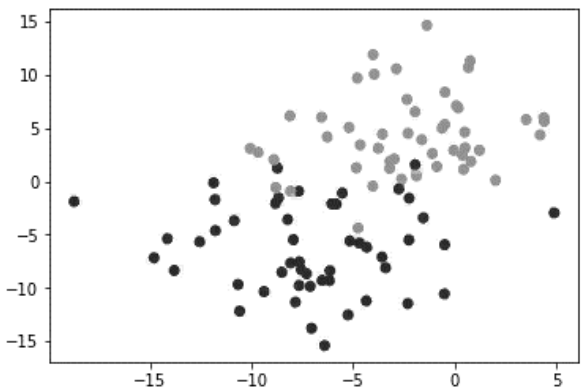

Рис. 7. Обучающая выборка

Видно, что объекты двух классов пересекаются между собой.

Разобьем выборку на обучающее и тестовое множества, используя функцию train_test_split. В качестве аргументов передаем массив *X*, массив *y*, *test_size* = 0.25 – означает, что на тестовую часть пойдет 25% всей выборки, соответственно, на обучающую – 75%, а также указываем, что разбиение будет случайным, но воспроизводимым (random_state = *1*). Если параметр random_state = None, то разбиение будет невоспроизводимым.

Функция train_test_split записывает результаты разбиения в 4 переменные. Назовем их *X*_train, X_test, y_train, y_test. В первую и вторую переменную будут записаны координаты объектов из обучающей и тестовой выборки соответственно, а в третью и четвертую – метки классов объектов из обучающей и экзаменационной выборки соответственно:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1,)

Таким образом, в переменной *X_train* лежат координаты, а в *y_train* – метки классов соответствующих объектов из тестовой выборки. Эти переменные будут использоваться в дальнейшем для обучения модели.

X_test, y_test – соответственно координаты и метки классов объектов тестовой выборки. Эти переменные мы будем использовать для оценки точности модели.

5.2.2 Обучение модели и классификация

Для обучения модели и последующей классификации с использованием модулей библиотеки *scikit-learn* используется стандартная процедура. Разберем на примере обучение и классификацию данных методом к-ближайших соседей:

1) Импортировать требуемый модуль, если этого не было сделано ранее

from sklearn.neighbors import KNeighborsClassifier

2) Создать переменную - модель классификатора, указав при необходимости параметры классификации. В нашем случае мы задаем два параметра – количество ближайших соседей = 1 и евклидову метрику.

knn = KNeighborsClassifier(n_neighbors=1, metric = ‘euclidean’)

Для большинства классификаторов, если не задавать никаких параметров, они будут выбраны по умолчанию. Список доступных параметров можно посмотреть в документации, в нашем случае - *http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html*

Посмотреть доступные метрики расстояний также можно в документации на *DistanceMetric: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.DistanceMetric.html*

3) Обучить модель, используя метод fit(), передав в него координаты объектов и метки классов обучающей выборки

```
knn.fit(X_train, y_train)
```

4) Оценить качество модели, используя метод predict()и тестовую выборку.

```
prediction = knn.predict(X_test)
```

Стоит отметить, что в метод predict() подаются только координаты объектов (X_test) без истинных меток класса (y_test). В общем случае, когда модель полностью настроена, в данный метод могут передаваться «боевые» данные – объекты, которые нужно проклассифицировать.

В нашем случае, в переменную prediction метод вернул предсказанные метки классов для каждого объекта из переменной X_test.

Зная истинные метки классов (переменная y_test) мы можем оценить, насколько точно работает наша модель.

Самое простое – вывести на экран истинные и предсказанные ответы:

```
print ('Prediction and test: ')
print (prediction)
print (y_test)
```

Кроме того, можно оценить матрицу неточностей (*confusion matrix*) используя функцию confusion_matrix, и передав в нее истинные и предсказанные ответы:

```
print ('Confusion matrix: ')
print (confusion_matrix(y_test, prediction))
```

Для оценки аккуратности классификации можно использовать функцию accuracy_score:

```
print ('Accuracy score: ', accuracy_score(prediction, y_test))
```

Для оценки показателей полноты-точности и *f1*-меры воспользуемся функцией classification_report:

```
print(classification_report(y_test, prediction))
```

Оценить показатель *AUC ROC* можно следующим образом:

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, prediction)
```

Кроме того, воспользовавшись функцией *plot_2d_separator,* описанной выше, можно наглядно отобразить на графике область принятия решений по каждому классу:

```
plt.xlabel("first feature")
plt.ylabel("second feature")
plot_2d_separator(knn, X, fill=True)
plt.scatter(X[:, 0], X[:, 1], c=y, s=70)
```

```
Prediction and test:
[1 1 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0]
[1 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 0 0 1 0 1 0 0 0]
Confusion matrix:
[[10  3]
 [ 2 10]]
Accuracy score:  0.8
             precision    recall  f1-score   support

          0       0.83      0.77      0.80        13
          1       0.77      0.83      0.80        12

avg / total       0.80      0.80      0.80        25
```

Out[28]: <matplotlib.collections.PathCollection at 0xcfb5690>

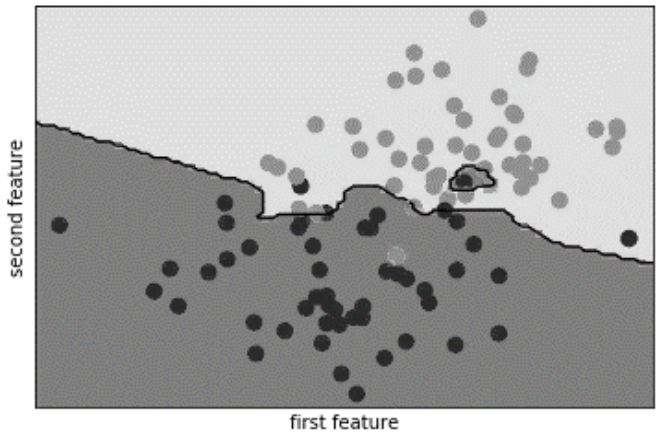

Рис. 8. Показатели качества классификации и область принятия решений

5.3 Лабораторная работа №2. Предварительная обработка текстовых данных

Цель работы: получить практические навыки обработки текстовых данных в среде *Jupiter Notebook*. Научиться проводить предварительную обработку текстовых данных и выявлять параметры обработки, позволяющие добиться наилучшей точности классификации.

Задание:

1) В среде *Jupiter Notebook* создать новый проект (*Notebook*).
2) Импортировать необходимые для работы библиотеки и модули.
3) Загрузить обучающую и экзаменационную выборку в соответствие с вариантом.
4) Вывести на экран по два документа каждого класса.
5) Применить стемминг, записав обработанные выборки (тестовую и обучающую) в новые переменные.
6) Провести векторизацию выборки:
    a. Векторизовать обучающую и тестовую выборки простым подсчетом слов (*CountVectorizer*) и значеним *max_features* = 10000.
    b. Вывести и проанализировать первые 20 наиболее частотных слов всей выборки и каждого класса по отдельности.
    c. Применить процедуру отсечения стоп-слов и повторить пункт *b*.
    d. Провести пункты *a – c* для обучающей и тестовой выборки, для которой проведена процедура стемминга.
    e. Векторизовать выборки с помощью *TfidfTransformer* (с использованием *TF* и *TF-IDF* взвешиваний) и повторить пункты *b-d*.
7) По результатам пункта 6 заполнить таблицы 5.2 и 5.3 наиболее частотными терминами обучающей выборки.

Таблица 5.2

Наиболее частотные термины без стемминга

	Без стемминга					
	Count		*TF*		*TF-IDF*	
№	Без стоп-слов	С стоп-словами	Без стоп-слов	С стоп-словами	Без стоп-слов	С стоп-словами
1						
2						
…						
20						

Таблица 5.3

Наиболее частотные термины со стеммингом

	Со стеммингом					
	Count		*TF*		*TF-IDF*	
№	Без стоп-слов	С стоп-словами	Без стоп-слов	С стоп-словами	Без стоп-слов	С стоп-словами
1						
2						
…						
20						

8) Используя конвейер (*Pipeline*) реализовать модель наивного байесовского классификатора и выявить на основе показателей качества (значения полноты, точности, *f1*-меры и аккуратности), какая предварительная обработка данных обеспечит наилучшие результаты классификации. Должны быть исследованы следующие характеристики:

   - Наличие –отсутствие стемминга.
   - Отсечение – не отсечение стоп-слов.
   - Количество информативных терминов (*max_features*).
   - Взвешивание: *Count, TF, TF-IDF*.

9) По каждому пункту работы занести в отчет программный код и результат выполнения данного кода.

10) По результатам классификации занести в отчет выводы о наиболее подходящей предварительной обработке данных (наличие стемминга, взвешивание терминов, удаление стоп-слов, количество информативных терминов).

Варианты заданий к лабораторной работе приведены в таблице 5.4, названия классов указаны в таблице 5.5.

Таблица 5.4

Варианты заданий к лабораторной работе №2

Вариант	Классы
1	2, 3, 8
2	6, 10, 11
3	1, 9, 17
4	7, 12, 18
5	4, 14, 18
6	1, 15, 16
7	3, 7, 13
8	5, 16, 20
9	6, 17, 19
10	3, 5, 15
11	7, 14, 20
12	2, 12, 13

Таблица 5.5

Названия классов

№ класса	Название класса
1	*'alt.atheism'*
2	*'comp.graphics'*
3	*'comp.os.ms-windows.misc'*
4	*'comp.sys.ibm.pc.hardware'*
5	*'comp.sys.mac.hardware'*
6	*'comp.windows.x'*
7	*'misc.forsale'*
8	*'rec.autos'*
9	*'rec.motorcycles'*
10	*'rec.sport.baseball'*
11	*'rec.sport.hockey'*
12	*'sci.crypt'*
13	*'sci.electronics'*
14	*'sci.med'*
15	*'sci.space'*
16	*'soc.religion.christian'*
17	*'talk.politics.guns'*
18	*'talk.politics.mideast'*
19	*'talk.politics.misc'*
20	*'talk.religion.misc'*

Контрольные вопросы

1) Особенности задачи классификации текстовых данных.
2) Этапы предварительной обработки данных.
3) Алгоритм наивного байесовского метода.
4) Как влияет размер словаря терминов на точность классификации?
5) Какие способы выявления информативных терминов вам известны?
6) Как влияет способ взвешивания терминов на точность классификации?

5.4 Методические указания к лабораторной работе №2

В данной работе мы продолжаем работать с библиотекой *scikit-learn* (*http://scikit-learn.org*), и хотим выяснить ее возможности при работе с текстовыми документами.

Ниже приведены новые модули, которые будут использованы в данной работе.

fetch_20newsgroups - *http://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_20newsgroups.html* - загружает набор данных «*20 news groups*», состоящий приблизительно из 18000 сообщений на английском языке по 20 тематикам, разбитым на обучающую и тестовую выборки.

Векторизаторы текста:

CountVectorizer - *http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html*

TfidfTransformer - *http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer*

Pipeline - *http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html* - конвейерный классификатор

MultinominalNB - http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html - Мультиномиальный наивный байесовский метод

5.4.1 Загрузка выборки

Данные в выборке «*20 news groups*» имеют следующий вид:

From: af774@cleveland.Freenet.Edu (Chad Cipiti)
Subject: Good shareware paint and/or animation software for SGI?
Organization: Case Western Reserve University, Cleveland, OH (USA)
Lines: 15
Reply-To: af774@cleveland.Freenet.Edu (Chad Cipiti)
NNTP-Posting-Host: hela.ins.cwru.edu

Does anyone know of any good shareware animation or paint software for an SGI machine? I've exhausted everyplace on the net I can find and still don't hava a nice piece of software.

Thanks alot!

Chad
--
Knock, knock. *Chad Cipiti*
Who's there? *af774@cleveland.freenet.edu*
cipiti@bobcat.ent.ohiou.edu
It might be Heisenberg. *chad@voxel.zool.ohiou.edu*

Сообщение состоит из заголовка (*header*), основной части, подписи или сноски (*footer*), а также может содержать в себе цитирование предыдущего сообщения (*quotes*).

Модуль fetch_20newsgroups позволяет выбирать интересующие тематики и удалять ненужные части сообщений. Для того чтобы выбрать сообщения по интересующим тематикам, необходимо передать список тематик в параметр categories. Для того чтобы удалить ненужные части сообщений, нужно передать их в параметр remove. Кроме того, важным параметром fetch_20newsgroups является subset - тип выборки – обучающая или тестовая.

Выберем сообщения по тематикам «Атеизм» и «Компьютерная графика», а также укажем, что нас не интересуют заголовки, цитаты и подписи:

```
categories = ['alt.atheism', 'comp.graphics']
remove = ('headers', 'footers', 'quotes')
twenty_train = fetch_20newsgroups(subset='train', shuffle=True,
random_state=42, categories = categories, remove = remove )
twenty_test = fetch_20newsgroups(subset='test', shuffle=True,
random_state=42, categories = categories, remove = remove )
```

Возвращаемый набор данных — это *scikit-learn* совокупность: одномерный контейнер с полями, которые могут интерпретироваться как признаки объекта (*object attributes*). Например, *target_names* содержит список названий запрошенных категорий, *target* - тематику сообщения, а *data* – непосредственно текст сообщения:

```
print (twenty_train.data[2])
```

Does anyone know of any good shareware animation or paint software for an SGI machine? I've exhausted everyplace on the net I can find and still don't hava a nice piece of software.

Thanks alot!

Chad

5.4.2 Векторизация

Чтобы использовать машинное обучение на текстовых документах, первым делом, нужно перевести текстовое содержимое в числовой вектор признаков.

Предобработка текста, токенизация и отбрасывание стоп-слов включены в состав модуля *CountVectorizer*, который позволяет создать словарь характерных признаков и перевести документы в векторы признаков:

```
vect = CountVectorizer(max_features = 10000, stop_words = 'english')
```

Создадим объект-векторизатор vect со следующими параметрами:

max_features = 10000 - количество наиболее частотных терминов, из которых будет состоять словарь

stop_words = 'english' – на данный момент модулем поддерживается отсечение английских стоп-слов. Кроме того, здесь можно указать список стоп-слов вручную. Если параметр не указывать, будут использованы все термины словаря.

Также, в работе может потребоваться настройка следующих параметров:

max_df - *float* в диапазоне [0.0, 1.0] или *int*, по умолчанию = 1.0. При построении словаря игнорирует термины, частота которых в документе строго превышает заданный порог (стоп-слова для конкретного корпуса). Если *float*, параметр обозначает долю документов, если целое число – то абсолютное значение.

min_df - *float* в диапазоне [0.0, 1.0] или *int*, по умолчанию = 1.0. При построении словаря игнорируйте термины, частота которых в документе строго ниже заданного порога. В литературе это значение также называется порогом. Если *float*, параметр обозначает долю документов, если целое число – то абсолютное значение.
После того как объект-векторизатор создан, необходимо создать словарь характерных признаков с помощью метода *fit()* и перевести документы в векторы признаков с помощью метода *transform(),* подав на него обучающую выборку:

```
vect.fit(twenty_train.data)

train_data = vect.transform(twenty_train.data)
test_data = vect.transform(twenty_test.data)
```

Также, можно отметить, что эти два действия могут быть объединены одним методом *fit_transform().* Однако, в этом случае нужно учесть, что для перевода тестовой выборки в вектор признаков, по-прежнему нужно использовать метод *transform().*

```
train_data = vect.fit_transform(twenty_train.data)
test_data = vect.transform(twenty_test.data)
```

Если для тестовых данных также воспользоваться методом *fit_transform()*, это приведет к перестроению словаря признаков и неправильным результатам классификации.

Следующий блок кода позволит вывести первые 10 терминов, упорядоченных по частоте встречаемости:

```
x=list(zip(vect.get_feature_names(),
np.ravel(train_data.sum(axis=0))))
def SortbyTF(inputStr):
    return inputStr[1]
x.sort(key=SortbyTF, reverse = True)
print (x[:10]))
```

5.4.3 TF- и TF-IDF взвешивание

CountVectorizer позволяет лишь определять частоту встречаемости термина во всей выборке, но такой подход к выявлению информативных терминов не всегда дает качественный результат. На практике используют более продвинутые способы, наибольшее распространение из которых получили *TF*- и *TF-IDF* взвешивания.
Воспользуемся методом *fit()* класса *TfidfTransformer(),* который переводит матрицу частот встречаемости в TF- и TF-IDF веса.

```
tfidf = TfidfTransformer(use_idf = True).fit(train_data)
train_data_tfidf = tfidf.transform(train_data)
```

Отметим, что в метод *fit()* нужно передавать не исходные текстовые данные, а вектор слов и их частот, полученный с помощью метода *transform()* класса *CountVectorizer*.
Для того, чтобы получить *tf-idf* значения, необходимо установить параметр *use_idf = True*, в противном случае на выходе мы получим значения *tf*

5.4.4 Классификация

После того как мы провели векторизацию текста и обучение модели, классификация для текстовых данных выглядит абсолютно идентично классификации объектов в первой лабораторной работе.

Задача обучения модели заключается не только в выборе подходящих данных обучающей выборки, способных качественно охарактеризовать объекты, но и в настройке многочисленных параметров метода классификации, предварительной обработке данных и т.д.
Рассмотрим, какие возможности предлагаются в библиотеке *scikit-learn* для автоматизации и упрощения данной задачи.

5.4.5 Pipeline

Чтобы с цепочкой *vectorizer => transformer => classifier* было проще работать, в *scikit-learn* есть класс *Pipeline* (конвейер), который функционирует как составной (конвейерный) классификатор.

```
from sklearn.pipeline import Pipeline
```

Промежуточными шагами конвейера должны быть преобразования, то есть должны выполняться методы *fit()* и *transform()*, а последний шаг – только *fit().*

При этом, *pipeline* позволяет устанавливать различные параметры на каждом своем шаге. Таким образом, проделанные нами действия по векторизации данных, взвешиванию с помощью *TF-IDF* и классификации методом К-БС с использованием *pipeline* будут выглядеть следующим образом:

```
text_clf = Pipeline([('vect', CountVectorizer(max_features= 1000, stop_words
= 'english')),
                     ('tfidf', TfidfTransformer(use_idf = True)),
                     ('clf', KNeighborsClassifier (n_neighbors=1)),])
```

Названия *vect, tfidf* и *clf* выбраны нами произвольно. Мы рассмотрим их использование в следующей лабораторной работе. Теперь обучим модель с помощью всего 1 команды:

```
text_clf = text_clf.fit(twenty_train.data, twenty_train.target)
```

И проведем классификацию на тестовой выборке:

```
prediction = text_clf.predict(twenty_test.data)
```

5.5 Лабораторная работа №3. Классификация текстовых данных

Цель работы: получить практические навыки решения задачи многоклассовой классификации текстовых данных в среде *Jupiter Notebook*. Научиться проводить предварительную обработку текстовых данных, настраивать параметры методов классификации, оценивать точность полученных моделей.

Задание:

1) Загрузить выборки по варианту из лабораторной работы №2.
2) Используя объект класса *GridSearchCV*, произвести предварительную обработку данных и настройку методов классификации в соответствие с заданием, вывести наилучшие значения параметров и результаты классификации модели (полнота, точность, *f1*-мера и аккуратности) с данными параметрами. Настройку проводить как на данных со стеммингом, так и на данных, на которых стемминг не применялся.
3) По каждому пункту работы составить отчет, в который включить программный код и результат выполнения данного кода.
4) По результатам классификации занести в отчет выводы о наиболее подходящем методе классификации данных с указанием параметров метода и описанием предварительной обработки данных.

Варианты заданий к лабораторной работе приведены в таблице 5.6.

ёТаблица 5.6

Варианты заданий к лабораторной работе №3

Вариант	Методы
1	*KNN, RF, LR*
2	*RF, MNB, SVM*
3	*KNN, DT, SVM*
4	*RF, MNB, KNN*
5	*LR, KNN, MNB*
6	*DT, KNN, LR*
7	*RF, LR, SVM*
8	*SVM, DT, LR*
9	*RF, LR, KNN*
10	*DT, SVM, LR*
11	*MNB, DT, KNN*
12	*RF, SVM, LR*

Параметры, которые необходимо настроить:

Помимо параметров предварительной обработки, таких как: взвешивание, отсечение стоп-слов, количество информативных терминов, для каждого метода классификации необходимо настроить следующие параметры:

К-ближайших соседей (*KNN*):

- количество ближайших соседей,
- метрика (евклидова, городских кварталов)

Дерево решений (*DT*):

- критерий (параметр *criterion: 'gini', 'entropy'*),
- глубина дерева (параметр *max_depth* от 1 до 5 с шагом 1, далее до 100 с шагом 20).

Случайный лес (*RF*):

- количество деревьев решений,
- критерий (параметр *criterion: 'gini', 'entropy'*),
- глубина дерева (параметр *max_depth* от 1 до 5 с шагом 1, далее до 100 с шагом 20).

Логистическая регрессия (*LR*):

- метод нахождения экстремума (*параметр solver: 'newton-cg', 'lbfgs', 'sag', 'liblinear'*),
- регуляризация (параметр penalty: 'L_1', 'L_2')

Обратить внимание, что разные виды регуляризации работают с разными методами нахождения экстремума.

Метод опорных векторов (*SVM*):

- функция потерь (параметр *loss: 'hinge', 'squared_hinge'*),
- регуляризация (параметр penalty: 'L_1', 'L_2')

Обратить внимание, что разные виды регуляризации работают с разными функциями потерь

Мультиномиальный Наивный Байесовский метод (*MNB*)

- параметр сглаживания α (параметр *alpha* {0,1;1;2}

Контрольные вопросы

1) Алгоритм и особенности метода опорных векторов.
2) Алгоритм и особенности метода логистической регрессии.
3) Алгоритм и особенности метода деревьев решений.
4) Что такое регуляризация?
5) Что такое метрика расстояния? Какие метрики вам известны?

5.6 Методические указания к лабораторной работе №3

В данной работе мы продолжаем работать с библиотекой *scikit-learn* (*http://scikit-learn.org*), и хотим выяснить ее возможности при работе с текстовыми документами.

Ниже приведены новые модули, которые будут использованы в данной работе.

GridSearchCV *http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html* - полный перебор по сетке заданных значений параметров для классификации

Decision Tree - *http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html* – Метод деревьев решений

MultinominalNB - *http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html* - Мультиномиальный Наивный Байесовский метод

LogisticRegression - *http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html* - Логистическая регрессия

5.6.1 Настройка параметров с использованием grid search

С помощью конвейерной обработки (*pipelines*), рассмотренной в лабораторной работе №2, стало гораздо проще указывать параметры обучения модели, однако перебирать все возможные варианты вручную даже в нашем простом случае весьма затратно. В случае, рассмотренном в лабораторной работе №2, имеется четыре настраиваемых параметра: *max_features, stop_words, use_idf* и *n_neighbors:*

```
text_clf = Pipeline([('vect', CountVectorizer(max_features= 1000, stop_words
= 'english')),
                     ('tfidf', TfidfTransformer(use_idf = True)),
                     ('clf', KNeighborsClassifier (n_neighbors=1)),])
```

Вместо поиска лучших параметров в конвейере вручную, можно запустить поиск (методом полного перебора) лучших параметров в сетке возможных значений. Сделать это можно с помощью объекта класса *GridSearchCV*. Объект класса *GridSearchCV* осуществляет полный перебор всех заданных пользователем параметров в указанном диапазоне, подавая их на вход классификатора, проводя обучение и тестирование модели, а также позволяет выбрать среди значений параметров те, при которых достигается наилучшее качество классификации.

```
parameters = {'vect__max_features': (100,500,1000,5000),
              'vect__stop_words': ('english', None),
              'tfidf__use_idf': (True, False)
              'clf__n_neighbors': (1,3,5,7)}
```

Для того чтобы задать сетку параметров необходимо создать переменную-словарь, ключами которого являются конструкции вида: «НазваниеШагаКонвейера__НазваниеПараметра», а значениями – кортеж из значений параметра:

Далее необходимо создать объект класса *GridSearchCV*, передав в него объект *pipeline* или классификатор, список параметров сетки, а также при необходимости, задав прочие параметры, такие так количество задействованных ядер процессора *n_jobs*, количество фолдов кросс-валидации *cv* и другие

```
gs_clf = GridSearchCV(text_clf, parameters, n_jobs=-1, cv=3)
```

Теперь объект *gs_clf* можно обучить по всем параметрам, как обычный классификатор, методом *fit().*

После того как прошло обучение, узнать лучшую совокупность параметров можно, обратившись к атрибуту *best_params_:*

```
gs_clf.best_params_
```

Аккуратность классификации – обратившись к параметру best_score_*:*
*gs_clf.best_score_*

А результаты по всей сетке параметров - *grid_scores_*
*gs_clf.grid_scores_*

5.7 Лабораторная работа №4. Кластеризация данных

Цель работы: получить практические навыки решения задачи кластеризации фактографических данных в среде *Jupiter Notebook*. Научиться настраивать параметры методов и оценивать качество полученного разбиения.

Задание:

1) Загрузить выборки согласно варианту задания.

2) Отобразить данные на графике в пространстве признаков. Поскольку решается задача кластеризации, то подразумевается, что априорная информация о принадлежности каждого объекта истинному классу неизвестна, соответственно, на данном этапе все объекты на графике должны отображаться одним цветом, без привязки к классу.

3) Провести иерархическую кластеризацию выборки, используя разные способы вычисления расстояния между кластерами: расстояние ближайшего соседа (*single*), дальнего соседа (*complete*), Уорда (*Ward*). Построить дендрограммы для каждого способа. Размер графика должен быть подобран таким образом, чтобы дендрограмма хорошо читалась.

4) Исходя из анализа дендрограмм, выбрать лучший способ вычисления расстояния между кластерами.

5) Для выбранного способа по дендрограммам определить количество кластеров в имеющейся выборке. Отобразить разбиение на кластеры и центроиды на графике в пространстве признаков (объекты одного кластера должны отображаться одним и тем же цветом, центроиды всех кластеров – также одним цветом, отличным от цвета кластеров)

6) Рассчитать сумму квадратов расстояний до центроида, сумму средних внутрикластерных попарных расстояний и сумму межкластерных попарных расстояний для данного разбиения. Сделать вывод о качестве разбиения.

7) Провести кластеризацию выборки методом *k*-средних для *k* [1, 10].

8) Сформировать три графика: зависимость суммы квадратов расстояний до центроида, суммы средних внутрикластерных попарных расстояний и суммы межкластерных попарных расстояний от количества кластеров. Исходя из результатов, выбрать наилучшее количество кластеров.

9) Составить сравнительную таблицу результатов разбиения иерархическим методом и методом *k*-средних.

Варианты заданий к лабораторной работе приведены в таблице 5.7.

Для всех вариантов дополнительно указать параметр *n_samples*=100 - объем выборки N.

Таблица 5.7

Варианты заданий к лабораторной работе №4

Вариант	**1**	**2**	**3**	**4**	**5**	**6**
Вид классов	*blobs*	*blobs*	*blobs*	*blobs*	*blobs*	*blobs*
Random_state	34	28	41	23	18	68
cluster_std	2.1	2	2	2	2	2
noise	-	-	-	-	-	-
Centers	7	7	7	6	8	6
Вариант	**7**	**8**	**9**	**10**	**11**	**12**
Вид классов	*classification*	*classification*	*classification*	*classification*	*classification*	*classification*
Random_state	55	36	3	68	5	27
class_sep	1.5	1.2	2	1	1.5	1
Для всех вариантов, использующих для генерации make_classification, дополнительные параметры: *n_features*=2, *n_redundant*=0, *n_informative*=2, *n_clusters_per_class*=1, *n_classes* = 4						

Контрольные вопросы

1) Постановка задачи кластеризации данных. Отличие от задачи классификации.

2) Какие способы вычисления расстояний между кластерами вам известны?

3) Что такое дендрограмма? Алгоритм построения дендрограммы.
4) Алгоритм метода *k*-средних.
5) Особенности метода *k*-средних, его достоинства и недостатки.
6) Способы оценки качества кластеризации данных.

5.8 Методические указания к лабораторной работе №4

В данной работе мы продолжаем работать с библиотекой *scikit-learn* (*http://scikit-learn.org*), и рассмотрим ее возможности для решения задачи кластеризации данных.

Ниже приведены новые модули, которые будут использованы в данной работе.

euclidean_distances - *https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.euclidean_distances.html* - евклидово расстояние между парой векторов.

KMeans - *https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html* – Метод *k*-средних

Кроме того, будут использованы следующие модули из библиотеки *Scipy* (*https://scipy.org*)

Linkage - *https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html* - Проведение иерархической кластеризации.

Dendrogram - *https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.dendrogram.html* - построение дендрограмм

fcluster - *https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.fcluster.html* - формирование кластеров из матрицы связей

Для вычисления критериев качества кластеризации могут быть полезными следующие методы массивов *numpy:*

shape - *https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.shape.html* – возвращает размерность массива

reshape - *https://docs.scipy.org/doc/numpy/reference/generated/numpy.reshape.html#numpy.reshape* – изменение размерности массива

mean - *https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.mean.html* - возвращает среднее значение элементов массива

where - *https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.where.html* - возвращает массив элементов, удовлетворяющих заданным условиям.

5.8.1 Кластеризация данных

Загрузка новых модулей может выглядеть следующим образом:

```
from sklearn.metrics.pairwise import euclidean_distances
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn.cluster import KMeans
```

Кроме того, необходимо подгрузить модуль *pyplot* для построения графиков и модуль *numpy* для обработки данных.

5.8.2 Иерархическая кластеризация

Для проведения иерархической кластеризации в данной работе мы воспользуемся модулем *linkage* из библиотеки *scipy*. В качестве параметров необходимо указать выборку данных и параметр *method* - метод вычисления расстояния между кластерами. На выходе модуль возвращает матрицу расстояний между всеми объектами. Например, для метода дальнего соседа:

```
mergings = linkage(X, method='complete')
```

Теперь, в переменной *mergings* хранится матрица расстояний. Чтобы построить по ней дендрограмму, воспользуемся модулем *dendrogram*, обязательным параметром которого является матрица расстояний, и отобразим дендрограмму на графике:

```
dendrogram(mergings)
plt.show()
```

Для того, чтобы разбить объекты на кластеры согласно матрице расстояний, воспользуемся модулем *fcluster*, передав матрицу расстояний, порог и критерий формирования кластеров. Например,

при указании критерия *‘distance’* и порога = *10*, данные будут разбиты на кластеры, расстояние между объектами которых не превышает 10:

```
T = fcluster (mergings, 10, 'distance')
print (T)
```

При указании критерия *‘maxclust’* и порога = 4, данные будут разбиты не более чем на 4 кластера.

Результатом будет массив длиной равной количеству объектов выборки, каждый элемент которого представляет собой номер кластера, к которому объект относится.

5.8.3 Оценка качества кластеризации

Для того чтобы оценить качество кластеризации, предлагается рассчитать следующие функционалы качества:

Сумма квадратов расстояний до центроида (*inertia*):

$$F_0 = \sum_{k \in Y} \sum_{l=1}^{N_k} \rho(\vec{X}_l, \mu_k)^2 \tag{5.1}$$

Сумма средних внутрикластерных расстояний:

$$F_1 = \sum_{k \in Y} \frac{1}{N_k} \sum_{l=1}^{N_k} \rho(\vec{X}_l, \mu_k) \tag{5.2}$$

Среднее межкластерное расстояние между центроидами:

$$F_2 = \frac{1}{N_k} \sum_{j,k \in Y} \rho(\mu_j, \mu_k) \tag{5.3}$$

Для расчета данных характеристик предварительно необходимо рассчитать центроиды каждого кластера, для чего предлагается следующая функция, позволяющая рассчитать координаты центроидов при условии наличия двух кластеров:

```
import numpy as np
def update_cluster_centers(X, c):
    ix = np.where(c==1)
    mu[0,:] = np.mean(X[ix,:], axis=1)
    ix = np.where(c==2)
    mu[1,:] = np.mean(X[ix,:], axis=1)
    return mu
```

На вход функции подается два массива одинаковой длины: *X* – исходная выборка объектов, *c* – номер кластера каждого объекта из выборки. При необходимости расчета центроидов более чем двух кластеров, предложенную функцию необходимо будет модифицировать.

Вывести на экран координаты центроидов, рассчитанные с помощью предложенной функции, можно следующим образом:

```
mu = np.array([[0.0,0], [0,0]])
mu = update_cluster_centers(X, T)
print(mu)
```

Для дальнейшего расчета функционалов качества удобно использовать функции *shape, reshape, mean, where* библиотеки *numpy*, а также функцией *euclidean_distances* из *sklearn*

5.8.4 K-средних

Для кластеризации методом k-средних с помощью модулей библиотеки *sklearn* используется алгоритм, схожий с алгоритмом проведения классификации данных: создаем модель кластеризации с помощью модуля *KMeans*(), указав предполагаемое количество кластеров, настраиваем ее на выборке методом *fit()*. После этого, с помощью метода *predict()* можно записать результаты кластеризации в переменную. Метод *predict()* возвращает массив длиной равной количеству объектов выборки, каждый элемент которого представляет собой номер кластера, к которому объект относится.

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
all_predictions =  kmeans.predict(X)
print (all_predictions)
```

С помощью метода *inertia_* можно вывести значение суммы квадратов расстояний до центроида, а координаты центроидов – с помощью метода *cluster_centers_*.

6 СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основной

1. Маннинг К.Д., Рагхаван П., Шютце Х. Введение в информационный поиск. – М.: «Вильямс», 2014. – 528с.
2. Флах П. Машинное обучение. – М.: «ДМК», 2015. – 399с.

Дополнительный

3. Орлов А.И. Прикладная статистика. М.: Экзамен, 2006. – 671 с.
4. Толчеев В.О. Современные методы обработки и анализа текстовой информации. М: Издательство МЭИ, 2006. – 75 с.
5. Лепский А.Е., Броневич А.Г. Математические методы распознавания образов. Курс лекций. Таганрог: Изд-во ЮФУ, 2009 – 152 с.
6. Уткин Л.В. Машинное обучение. Деревья решений. *(http://www.levvu.narod.ru/Machine_Learning_LTU_4.pdf)*
7. Чистяков С.П. Случайные леса: обзор. Труды Карельского научного центра РАН № 1. 2013. С. 117–136.
8. Chapelle O., Vapnik V., Bousquet O., Mukherjee S. Choosing Multiple Parameters for Support Vector Machines // Machine Learning. 2002. № 46 (1-3). P. 131-159.
9. http://www.machinelearning.ru/wiki/images/2/25/SMAIS11_SVM.pdf - Метод опорных векторов для стандартной задачи классификации.

Учебное издание

Мохов Андрей Сергеевич

Толчеев Владимир Олегович

Бородкин Артем Александрович

АНАЛИЗ И ОБРАБОТКА ТЕКСТОВЫХ ДАННЫХ

Практикум

Редактор Д.Р. Чернова

Компьютерная верстка З.Х. Айнетдиновой

Подписано в печать 27.07.2018.		Печать офсетная	Формат 60×84/16
Печ. л. 3,5	Тираж 45 экз.	Изд. № 18у-032	Заказ №

Оригинал-макет подготовлен в РИО НИУ «МЭИ».

111250, г. Москва, ул. Красноказарменная, д. 14.

Отпечатано в типографии НИУ «МЭИ».

111250, г. Москва, ул. Красноказарменная, д. 1