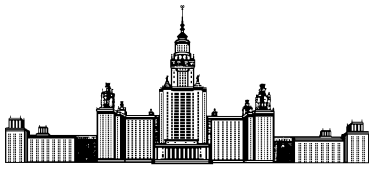

Московский государственный университет имени М. В. Ломоносова Факультет
Вычислительной математики и кибернетики
Кафедра Математических методов прогнозирования

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Структурные и статистические методы анализа эмоциональной окраски текста

Выполнила:
студентка 417 группы
Лукашкина Юлия Николаевна

Научный руководитель:
к.ф-м.н., доцент
Чехович Юрий Викторович

Москва, 2015

Содержание

Аннотация

Анализ эмоциональной окраски текста — это современный подход оценки мнения автора по отношению к объектам, которые описываются в тексте. В данной работе рассматриваются различные методы определения тональности теста, предложенные в литературе, производится их анализ и сравнение. Сравнение производится на нескольких современных наборах данных. Кроме того, предлагается модификация одного из методов, которая улучшает качество его работы. Также, в настоящей работе рассматриваются композиции нескольких алгоритмов машинного обучения, что позволяет добиться наиболее высокого качества среди всех рассмотренных подходов.

1 Введение

Анализ эмоциональной окраски текста — это современный подход оценки мнения автора по отношению к объектам, которые описываются в тексте. Данная задача становится всё более актуальной в последние годы в связи с возрастающим числом пользователей различных интернет-услуг. К такого рода услугам относятся магазины, сервисы для просмотра кинофильмов, прослушивания аудиозаписей, чтения электронных книг. Поскольку объём предоставляемой информации огромен, для успешного функционирования таких сервисов требуются качественные рекомендательные системы.

Задача выдачи релевантных рекомендаций решается различными методами машинного обучения, одним из которых и является анализ тональностей текстов, который позволяет определять отношение автора текста к описываемому в тексте предмету. Анализ тональности, применённый к отзывам пользователей на различный контент, позволяет выявлять отношения к этому контенту со стороны разных групп пользователей. Это позволяет предоставлять им в будущем более релевантные кинофильмы, аудиозаписи, книги и т.п.

В данной работе рассматривается частный случай задачи анализа тональности, а именно бинарный. Требуется определить, является ли отношение автора к описываемому контенту позитивным или негативным. Т.е. ставится задача двухклассовой классификации. В качестве исходных данных используются отзывы на кинофильмы, книги и электронику.

Двумя основными подходами в данной области являются т.н. *структурные* и *статистические* методы. В первой группе методов используется структура анализируемого текста, вторые же не оперируют информацией о структуре текста, а используют, например, частоты встречаемости слов или словосочетаний.

В настоящей работе рассматриваются различные предложенные в литературе методы, производится их анализ и сравнение. Кроме того, предлагается модификация одного из методов, которая улучшает качество его работы. Также, в настоящей работе рассматриваются композиции нескольких алгоритмов машинного обучения, что позволяет добиться наиболее высокого качества.

Данная работа имеет следующую структуру.

В разделе 1 приводится содержательная постановка задачи и вводятся основные обозначения.

В разделе 2 проводится обзор современных методов анализа тональности текста.

В разделе 3 описываются данные, на которых в дальнейшем будут проводится эксперименты.

Раздел 4 включает в себя описание проведенных экспериментов и их результаты. Также, в этом разделе описано сравнение различных подходов к анализу тональности текста.

1.1 Определения и обозначения

Задана коллекция текстовых документов D, множество употребляемых в них слов W. Каждый документ d из коллекции D представляет собой последовательность слов $W_d = (w_1, \ldots, w_{n_d})$ из словаря W, где n_d — длина документа d. Каждому документу можно поставить в соответствие его тональность $t \in T = \{0, 1\}$ (0 — негативный класс, 1 — позитивный).

Существует неизвестная целевая зависимость — отображение $y^* : D \to T$, значения которой известны только на объектах конечной обучающей выборки $D^m = \{(d_1, t_1), \ldots, (d_m, t_m)\}$. Требуется построить алгоритм $a : D \to T$, который приближал бы неизвестную целевую зависимость как на элементах выборки, так и на всём множестве D. Для оценки качества работы алгоритма будем использовать *индикаторную* функцию потерь, часто используемую в задачах классификации:

$$\mathcal{L}(t, t') = [t' \neq t]$$

Эмпирический риск — это функционал качества, характеризующий среднюю ошибку алгоритма a на выборке X^m:

$$Q(a, X^m) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a(x_i), y^*(x_i)).$$

2 Используемые методы

В последние годы число публикаций, освещающих проблему анализа тональности текста, существенно возросло. Рассмотрим некоторые популярные методы, которые используются в настоящее время.

В работе [7] исследовалась проблема определения эмоциональной окраски отзывов на кинофильмы. Рассматривалось несколько моделей представления документов выборки — *униграммная*, *биграммная*, в виде текстов, состоящих только из прилагательных и т.д.

При *униграммной* модели текста (так же известной как «мешок слов», bag–of–words) делается предположение о независимости слов, и, таким образом игнорируются любые связи между словами в предложении и между предложениями в целом. Документ $d \in D$ может быть представлен как $|W|$-мерный вектор $d = (w_1, \ldots, w_{|W|})$, где $|W|$ — размер словаря (число неповторяющихся слов), а $w_i,\ i = 1, \ldots, |W|$ — вес $i_{\text{ого}}$ слова в документе, вес рассчитывался по следующим формулам:

$$\begin{cases} w_i = 1, \text{ если } tf_i > 0, \\ w_i = 0, \text{ если } tf_i = 0, \end{cases} \qquad \text{или} \qquad w_i = tf_i,$$

где tf_i – частота встречаемости i-ого слова в документе. Данные способы определения веса слов в дальнейшем будем называть *частотным* и *бинарным* соответственно.

Результаты экспериментов, которые будут описаны далее показывают, что бинарное представление текста выигрывает по сравнению с частотным.

При *биграммной* модели (в общем случае – *n-граммной*) используется представление документа в виде n-грамм (упорядоченной подпоследовательности слов документа длины n). Использование данной модели позволяет, например, учитывать информацию о словосочетаниях.

Также авторы работы использовали мета-информацию, связанную с текстом, например, априорную информацию о частях речи слов. Однако такая дополнительная информация не улучшила качество классификации. В работе использовались

следующие классификаторы: SVM, байесовский классификатор, max-entropy-text-classifier. Эксперименты показали, что наилучший результат был получен с помощью SVM в рамках признакового пространства, задаваемого униграммной моделью.

Авторы [9] предложили алгоритм обучения без учителя для задачи определения тональности отзывов. В работе было высказано предположение, что наиболее важная информация об отзыве содержится в прилагательных и наречиях. Для проверки этого предположения к тесту была применена автоматическая разметка на части речи (part-of-speech tagger) для выделения фраз, содержащих прилагательные (с существительными) или наречия (с глаголами).

На следующем этапе определялась эмоциональная оценка каждой выделенной фразы. Она определялась близостью фразы к эталонным словам «excellent» и «poor». В качестве меры близости авторы использовали метрику *поточечной взаимной информации* (Pointwise Mutual Information), исходя из предположения, что высокая степень совместной встречаемости фраз указывает на их схожесть.

$$PMI(word_1, word_2) = \log_2 \frac{P(word_1,\ word_2)}{P(word_1)\ P(word_2)}$$

Значения метрики были приблизительно посчитаны с помощью поисковой системы Altavista [1] :

$$PMI(word_1, word_2) = \log_2 \frac{\text{hits}(word_1\ NEAR\ word_2)}{\text{hits}(word_1)\ \text{hits}(word_2)},$$

где $\text{hits}(query)$ – число ответов выдаваемых по данному запросу.

Тональность фразы *polarity* определялась как

$$polarity(phrase) = PMI(phrase, \text{«excellent»}) - PMI(phrase, \text{«poor»})$$

$$polarity(phrase) = \log_2 \frac{\text{hits}(phrase\ NEAR\ \ \text{«excellent»})\ \text{hits}(\text{«poor»})}{\text{hits}(phrase\ NEAR\ \ \text{«poor»})\ \text{hits}(\text{«excellent»})}$$

Оценка тональности отзыва получалась усреднением всех оценок тональностей фраз этого отзыва. Класс документа (позитивный или негативный) определялся с помощью порогового решающего правила.

[1] `http://www.altavista.com/sites/search/adv`

Описанный подход неплохо показал себя при решении ряда задач. Тем не менее, результаты, полученные им при решении проблемы определения тональности отзывов на кинофильмы, оказались ниже, чем у SVM [7].

В работе [4] данные представлялись моделью «мешка слов» ($d = (w_1, \ldots, w_n), \forall d \in D$). Однако в отличии от [7], где использовались простые подходы к взвешиванию слов (2), авторы предложили несколько более сложных весовых функций. В качестве основы использовалась *статистическая мера TF-IDF*:

$w_i = tf_i \times idf_i$,

- где tf_i — частота встречаемости i-го слова в документе;
- idf_i — обратная частота документа (inverse document frequency).

Авторы статьи рассмотрели разные модификации этой меры, например, модификация $\Delta(t)$ idf:

$$w_i = tf_i\, log_2(\frac{N_1}{df_{i,1}}) - tf_i\, log_2(\frac{N_2}{df_{i,2}}) = tf_i\, log_2(\frac{N_1\ df_{i,2}}{N_2\ df_{i,1}}),$$

- где N_j – число тренировочных документов в классе j,
- $df_{i,j}$ – число тренировочных документов класса j, которые содержат слово i.

Данный подход не предлагает никакого сглаживания, и поэтому возникают проблемы с словами, которые встречаются только в одном классе ($df_{ij} = 0$). Для решения этой проблемы был предложен сглаженный аналог:

$$w_i = tf_i\, log_2(\frac{N_1\ df_{i,2} + 0.5}{N_2\ df_{i,1} + 0.5})$$

Нормирование весов слов одного документа $d = (w_1, \ldots w_n), \quad \forall d \in D$ производилось по формулам:

$$\|d\| = \frac{1}{\sqrt{w_1^2 + \cdots + w_n^2}} \quad (l2), \qquad \text{или} \qquad \|d\| = \frac{1}{w_1 + \cdots + w_n} \quad (l1),$$

либо не проводилось совсем.

Вес слова в документе задавался тройкой $(tf, idf, norm)$ — где формулы для расчета компонент брались из таблицы 2. Была проведена серия экспериментов, в ходе

которых всевозможные представления весов слов использовались с нормировкой и без неё. Наилучшие результаты показали комбинации $o\Delta(k)n$, $b\Delta(t')n$.

Описанные выше модификации меры TF-IDF позволили добиться улучшения результатов классификации документов.

Аббревиатура	**TF**	**Аббревиатура**	**IDF**
n (natural)	tf	n (no)	1
l (logarithm)	$1 + \log(tf)$	t (idf)	$\log \frac{N}{df}$
a (augmented)	$0.5 + \frac{0.5tf}{max\_t(tf)}$	p (prob idf)	$\log \frac{N-df}{df}$
b(boolean)	$I[tf_i > 0]$	k (BM25 idf)	$\log \frac{N-df+0.5}{df+0.5}$
		$\Delta(t)$ (Delta idf)	$\log \frac{N_1 df_2}{N_2 df_1}$
L(log ave)	$\frac{1+\log(tf)}{1+\log(avg\_dl)}$	$\Delta(t')$ (Delta smoothed idf)	$\log \frac{N_1 df_2+0.5}{N_2 df_1+0.5}$
o(BM25)	$\frac{(k_1+1)tf}{k_1((1-b+b\frac{dl}{avg_dl})+tf}$	$\Delta(k)$	$\log \frac{(N_1-df_1+0.5)df_2+0.5}{(N_2-df_2+0.5)df_1+0.5}$

Таблица 1: avg_dl среднее число слов в документах, dl — длина документа, $k_1 = 1.2$, $b = 0.95$ — параметры были подобраны авторами статьи [4].

Новый подход к решению данной задачи был предложен в [5]. В его рамках каждый документ представлялся в виде графа. Вершинами этого графа являются предложения, рёбрами — т.н. *коэффициенты связи*, рассчитываемые с помощью различных эвристик. Кроме того, в каждом таком графе имеются две дополнительные вершины — субъективные и объективные полюса. Авторы описали алгоритм разбиения документа на объективные и субъективные предложения на основе поиска минимального разреза (сечения) этого графа, после чего применяли методы машинного обучения только к частям документа, содержащим эмоциональную окраску. В сравнении с предыдущей работой, такой подход позволил улучшить качество анализа (на 2-3% при использовании SVM) и в среднем сократить размер текста обучающих отзывов на 40%.

Все рассмотренные раннее подходы не учитывают структуру документа, информация о которой может улучшить точность классификации.

Знание о структуре документа позволяет, например, вводить различные веса для различных частей текста (введения, заключения и т.д). Большинство отзывов содержат значимую информацию в конце, поэтому, присваивая словам из заключения

больший вес, можно добиться повышение качества классификации. В общем, можно делить отзыв на некоторое количество частей и присваивать разные веса словам из разных частей.

Авторы статьи [8] предложили метод определения тональности отзыва с помощью подсчета некоторой метрики и использования порогового решающего правила. Для оценивания тональности отзыва использовался алгоритм *SO-CAL* (Semantic Orientation CALculator): тональность каждого слова вычислялась с помощью поисковой системы Google, путем определения меры совместной встречаемости данного слова и слов из размеченного словаря; учитывалось влияние слов-модификаторов (таких как «really», «(the) most», «pretty» и т.д.) на общую эмоциональную окраску фразы; также авторы алгоритма учитывали инвертирование тональности, например, с помощью слов «not», «never» и т.д. Данный подход позволил лучше оценивать тональность сложных отзывов.

В работе [2] авторы использовали теорию *риторической структуры* (Rhetorical Structure Theory). Данная теория основана на том, что текст разбивается на некоторые *части* (spans) и между ними определяются *риторические отношения*. Выделяют несколько типов риторических отношений: последовательность (sequence), противопоставление (contrast), конъюнкция (joint) и т.д. В общем, существует два способа построения сложного предложения: *паратаксис* [11] и *гипотаксис* [11]. При гипотаксисе одна часть текста является ядром *ядром* (nucleus), а остальные — *сателлитами* (satellite). Ядро — это наиболее информативная часть текста. Сателлиты — менее важные части текста, они зависят от ядер. При другом способе построения предложения (паратаксисе) все части текста одинаково значимы, следовательно они все рассматриваются как ядра. Другими словами, рассматриваются сложноподчиненные и сложносочиненные предложения. В теории риторической структуры простые предложения являются элементарными частями текста, они могут объединятся и образовать более сложные единицы. Данный подход позволяет описывать строение текста в виде иерархической структуры — RST-дерева [3]. Такой способ представления текста позволяет вводить различные веса для слов из разных частей, что позволяет более точно определять тональность документа в целом.

3 Описание данных

3.1 Данные

Для проведения экспериментов были выбраны два набора данных: Movie Review Data [2] и Multi-Domain Sentiment Dataset [3]. Movie Review Data содержит коллекцию отзывов на кинофильмы извлеченных из Internet Movie Database (IMDb). Были отобраны только положительные и негативные отзывы. Также, чтобы избежать доминирования специфической лексики было наложено ограничение на количество отзывов от одного автора (не более 20 отзывов). В итоге был получен набор данных из 2000 отзывов: 1000 положительных и 1000 негативных.

Multi-Domain Sentiment Dataset содержит отзывы покупателей на различные продукты, информация была получена с сайта Amazon.com. Данный набор данных состоит из отзывов на книги, dvd и электронику, по 2000 отзывов на каждую категорию, классы сбалансированы (1000 отзывов на каждый класс).

В таблице 2 приведены некоторые характеристики данных.

	кинофильмы	книги	dvd	электроника
средняя длина отзыва	629	166	160	103
средняя длина положительного отзыва	593	170	151	103
средняя длина негативного отзыва	665	162	169	102
размер словарика	39659	22503	21758	11450

Таблица 2: Характеристики данных.

3.2 Предобработка данных

Так как исходные данные являются необработанным текстом, была проведена предварительная обработка для приведения документов к нормализованному виду. Рассмотрим три этапа предобработки текста: приведение текста к нижнему регистру, стемминг, удаление редко и часто встречающихся слов.

[2] http://www.cs.cornell.edu/people/pabo/movie-review-data/

[3] http://www.cs.jhu.edu/~mdredze/datasets/sentiment/index2.html

Стемминг слов производится для приведения слова к его основе. То есть разные формы слова приводятся к единому виду. В настоящей работе использовался стеммер Портера[4].

Так как все наборы данных имеют большой размер словаря, не рассматривались признаки, которые встречаются реже чем заданное пороговое число раз. Для определения данного порога была проведена серия экспериментов. Если не рассматривать признаки, которые встречаются реже, чем три раза, то качество классификации не ухудшится, по сравнению с использованием всех признаков, см. рис. 1.

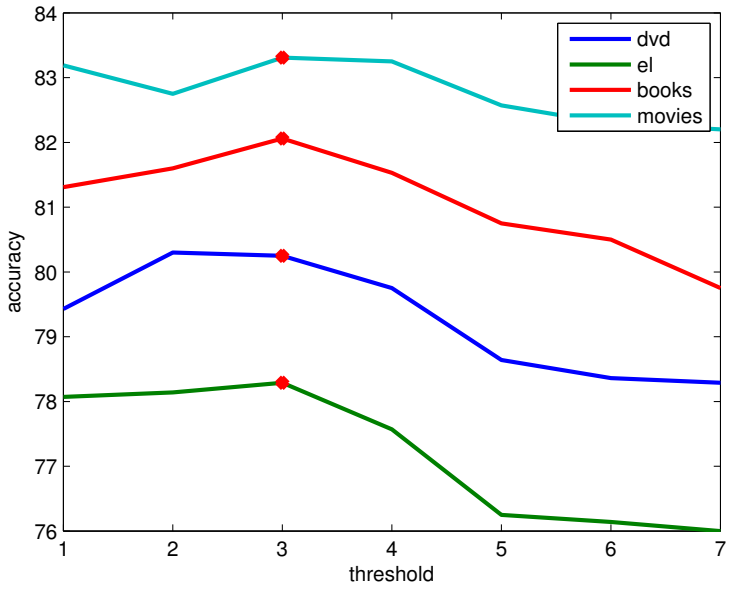

Рис. 1: Точность классификации для различных порогов на частоту встречаемости признаков. NB, модель (1,1).

Также не учитывались часто встречаемые слова (данное преобразование, в основном, затрагивает предлоги, союзы и т.д.).

[4] `http://www.nltk.org/api/nltk.stem.html`

4 Результаты экспериментов

4.1 Используемая система для экспериментов

Все вычисления производились на компьютере Toshiba Satellite. Использовался процессор 2 ГГц Intel Core i5, 4 Гб оперативной памяти 1600 МГц, операционная система Ubuntu 14.04.2 LTS.

4.2 Структура эксперимента

Эксперименты проводились по следующей схеме: каждая исходная выборка делилась на 2 части, 20% становились скрытой тестовой выборкой, а остальные 80% выборки фиксировались для проведения процедуры скользящего контроля с целью повышения устойчивости результата. Опишем процедуру скользящего контроля: вторая часть выборки случайным образом делилась на восемь подвыборок. Далее, было произведено восемь запусков, в ходе которых каждая подвыборка была использована в качестве тестовой один раз. Все остальные подвыборки, при этом, были использованы для обучения моделей. Итоговая точность классификации была получена усреднением результатов всех восьми запусков. Таким образом, в каждом эксперименте вычисляются две величины: точность на скользящем контроле с восьмью фолдами и точность на скрытой тестовой выборке.

4.3 Классификаторы

В данной работе были рассмотрены следующие классификаторы: линейный SVM, SVM с RBF ядром, двухслойная нейронная сеть с сигмоидальной функцией активации (NN) и наивный байесовский классификатор (NB). Они являются одними из самых распространенных классификаторов, используемых для решения задачи определения тональности текста. Поэтому именно они были выбраны для проведения экспериментов. Для SVM и нейронной сети была произведена настройка параметров по сетке. SVM — настройка параметра регуляризации C; SVM с RBF ядром — настройка параметра регуляризации C и ширины ядра γ; NN — подбор количества нейронов на скрытом уровне и настройка коэффициента скорости обучения.

4.4 Программная реализация

Эксперименты были реализованы на языке программирования Python 2.7.6 с использованием библиотек scikit-learn 14.1[5] и nltk[6]. Также была использована библиотека Vowpal Wabbit (VW)[7].

Программную реализацию можно условно разделить на две части. Первая часть включает в себя предобработку текста, а также разбиение коллекции на контрольную и обучающую выборки. Далее, для повторного воспроизведения экспериментов, предобработанные данные сохранялись в формате svmlight[8]. Вторая часть программы содержит код, реализующий непосредственно классификацию документов. Листинг программной реализации приведён в приложении А.

4.5 Используемые методы

Рассмотрим различные модели признакового представления данных: униграммную, биграммную и т.д. В таблице 3 приведены значения точности классификации этих моделей на скользящем контроле. Здесь и далее, под обозначением *(1, n)* подразумевается комбинация униграмм, биграмм и n-грамм. *(1, n) (ч)* — комбинация 1-n грамм, частотное представление весов (2). Для каждого набора данных жирным было выделено лучшее значение. К всем способам представления признаков была применена нормировка, по формуле (l2) (данная норма была выбрана после проведения серии экспериментов с различными видами нормировки). Точность классификации выборок с нормированными признаками выше, чем без нормировки.

Для всех наборов данных наилучшие результаты показывает 1-2 граммное и 1-3 граммное признаковое представление. Заметим, что увеличение числа n-грамм не дает прироста качества, а размеры признакового пространства при этом сильно увеличиваются. Бинарное представление весов слов документа дает более высокие показатели качества, по сравнению с частотным аналогом.

---

[5] `http://scikit-learn.org/stable/`

[6] `http://www.nltk.org/`

[7] `https://github.com/JohnLangford/vowpal_wabbit/wiki`

[8] `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.dump_svmlight_file.html`

	dvd	dvd	dvd	эл-ка	эл-ка	эл-ка	кни-ги	кни-ги	кни-ги	к/ф	к/ф	к/ф
	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB
(1,1)	79.5	80.25	80	79.05	78.29	78.19	80	81.56	82.06	85.56	86.81	83.31
(1,1) (ч)	80.4	81.15	82.05	80.3	79.42	80.98	81.5	82.31	82.98	86.18	87.1	84.7
(1,2)	**83.5**	83	80.75	**81.12**	79.14	80.44	83.31	83.62	84.94	85.63	**87.38**	84.44
(1,2) (ч)	82.12	81.97	78.8	79.92	78.2	79.74	82.5	81.8	83.44	84.1	86.9	82.14
(1,3)	82.5	82.75	81.25	80.62	79.57	79.5	83	83.69	**85.69**	84.31	87.19	85.19
(1,3) (ч)	81.9	82.05	80.22	79.5	78	77.91	82.1	84.1	84.14	83.6	86.12	83.8
(1,4)	83	83	81.75	80.75	79.43	79.88	83.06	83.69	85.1	85.44	86.8	85.44
(1,4) (ч)	81.5	82.1	79.85	79.44	76.4	77.6	80	81.7	82.78	83.2	85.1	84.14
(2,2)	78.25	77	80.25	79.31	77.79	74.88	80.19	81.19	83.25	83.75	82.31	84.44
(2,2) (ч)	77.5	76.1	79.1	78	76.14	74.04	79.4	80	81.75	82.13	80.4	82.28
(3,3)	68.75	72.5	72.25	72.56	69.79	67.94	73.69	75.69	76.88	79.88	78.44	80.75
(3,3) (ч)	66	70.7	70.65	70.12	65.1	64.19	72.05	73.8	74.57	77	77.5	79.1
(4,4)	61.75	65.5	63.75	64.25	63.21	62.25	65.5	68.19	63.81	71.5	72.12	74.94
(4,4) (ч)	60.5	63.2	60.4	62.8	62.1	60.5	63.97	66.91	62	69.41	70.14	72.2

Таблица 3: Точность классификации (в %) различных моделей представления признаков.

В таблице 4 выписаны результаты точности классификации на скользящем контроле отзывов на dvd. Для всех моделей представления данных классификатор SVM с RBF ядром даёт наихудшие результаты. На остальных наборах данных данная тенденция сохраняется. В дальнейшем откажемся от рассмотрения данного классификатора, если не оговорено другое.

В таблице 5 предоставлены результаты сравнения различных модификаций TF-IDF, описанных в разделе 2. Напомним, что использовалась модель представления документа весами слов: $d = (w_1, \ \ldots, \ w_n), \ \forall d \in D$. Вес слова w_i задавался тройкой $(tf, \ idf, \ norm)$, формулы для расчета компонент брались из таблицы 2. Так как общее число комбинаций большое в таблице приведены результаты только для

	(1,1)	**(1,2)**	**(1,3)**	**(1,4)**	**(2,2)**	**(3, 3)**	**(4, 4)**
NN	79.5	83.5	82.5	83	78.25	68.75	61.75
SVM	80.25	83	82.75	83	77	72.5	65.5
SVM RBF	77.5	76.9	77.25	76.75	75.5	67.9	63.5
NB	80	80.75	81.25	81.75	80.25	72.25	63.75

Таблица 4: Сравнение классификаторов. Набор данных dvd.

лучших троек. Экспериментально было установлено, что нормирование признаков может дать существенный (до 5%) прирост в качестве. Применение $l2$ нормировки дает более высокие результаты при классификации, чем применение $l1$ нормировки. Наилучшие результаты показали тройки $(n,\ t,\ l2)$ и $(l, \Delta(k),\ l2)$. В статьи [4] наилучшие результаты показали тройки $(l,\ \Delta(k),\ l2)$ и $(l,\ t,\ l2)$. Авторы статьи для оценки построенной модели использовали LOO контроль. Выбор LOO был связан с тем, что расчет значений TF и IDF чувствителен к размерам тренировочной выборки. В данной работе, как уже отмечалось ранее, для унификации результатов различных методов во всех экспериментах проводился контроль по восьми блокам.

	dvd	dvd	dvd	эл-ка	эл-ка	эл-ка	кни-ги	кни-ги	кни-ги	к/ф	к/ф	к/ф
	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB
(n,t)	83.25	83	83.19	80	81.79	81.69	82.13	85.88	**86.56**	86.31	**87.69**	86.38
(l,k)	79.5	82.5	82.8	79.38	81.71	81.8	81.25	83.5	84.14	85.75	86.12	86.87
(l,$\Delta(k)$)	**83.75**	83.25	83	**81.86**	80.75	80.14	83.1	82.75	80.63	86.19	87.44	86
(o,t)	77.75	79.75	80.5	76.75	78.75	80.43	79.44	83	83.56	82.69	84.69	83

Таблица 5: Точность классификации (в %) различных модификаций TF-IDF. Униграммы.

Сравним результаты различных модификаций TF-IDF с результатами классификации в случае представления признаков как бинарных или частотных униграмм. Точность классификации всех выборок выше у TF-IDF модели, чем у простой бинарной или частотной n-граммной модели (см. таблицы 3, 5).

Далее, рассмотрим один из простейших структурных подходов: будем использовать униграммную модель представления признаков и будем присваивать разные веса словам из различных частей текста. Документ будем делить на две, три, четыре и пять равных частей. В таблице 6 приведены наилучшие значения точности классификации на скользящем контроле и на скрытой тестовой выборке. Данный подход незначительно увеличивает точность классификации на наборе отзывов на электронику и кинофильмы. Для остальных двух наборов (отзывы на книги и dvd) результаты классификации без взвешивания лучше.

	dvd		эл-ка		книги		к/ф	
	cv	ht	cv	ht	cv	ht	cv	ht
без взвешивания	**80.25**	78.75	79.05	78.75	**82.06**	80.5	86.81	85.75
2 части	79.4	74.25	81.72	81.25	82.04	82	87.05	86.5
3 части	79.63	77.5	**82.1**	81.75	84	82.75	**87.22**	86.75
4 части	80.17	78.25	80.93	80	81.7	80.25	86.98	86
5 частей	79.9	76	80.05	77.75	81.41	79.5	85.63	82.5
позиция слова	78.8	74.75	81.84	79.75	80.02	77.5	85.4	84

Таблица 6: Точность классификации (в %) для униграммного представления с перевзвешиванием отдельных частей документа.

Рассмотрим структурный подход, описанный в статье [8]. Будем использовать реализацию парсера для RST предложенную Feng и Hirst[9]. Для вычисления тональности отдельного слова будем использовать корпус SentiWordNet[10]. Данный корпус позволяет для каждого слова рассчитать оценку за положительный и негативный класс. Тональность всего документа будем вычислять усреднением всех оценок, с учетом некоторых риторических отношений, таких как отрицание и усиление. Результаты эксперимента на указанных наборах данных предоставлены в таблице 7. Учёт одновременно и отрицания, и усиления увеличивает точность классификации.

[9] `http://www.cs.toronto.edu/~weifeng/software.html`

[10] `http://sentiwordnet.isti.cnr.it/`

Однако, по сравнению, например, с TF-IDF для униграмм результаты получились хуже.

	dvd		эл-ка		книги		к/ф	
	cv	ht	cv	ht	cv	ht	cv	ht
усиление	81.5	79.25	80.14	79.75	84.5	82.75	86.93	85.75
отрицание	82.84	80.75	80.6	79.75	85.8	82.25	**87.4**	86.5
усиление и отрицание	**83.1**	81.5	**81.23**	80.25	**86.4**	84.5	87.12	86.25

Таблица 7: Точность классификации (в %) структурного метода.

TF-IDF модель дает наилучшие результаты классификации, попробуем его ещё улучшить. Поставим эксперименты с биграммной моделью представлением документа. Вычисление весов будем проводить по приведенным ранее формулам. Результаты классификации для такого представления предоставлены в таблице 8. Здесь и далее, символ b в обозначениях означает биграммное представление. Для данных, содержащих отзывы на dvd и кинофильмы точность классификации увеличилась и является лучшим значением точности для этих наборов, среди рассмотренных ранее в данной работе.

	dvd	dvd	dvd	эл-ка	эл-ка	эл-ка	кни-ги	кни-ги	кни-ги	к/ф	к/ф	к/ф
	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB
(n,t)b	82.75	83.05	82.06	79.63	81.07	81.12	82.38	**85.44**	84.81	86.75	86.56	85.31
(l,k)b	78.57	82.75	81.44	78.06	**81.29**	80.74	82.88	84.12	82.7	85.44	87.06	85.34
(l,$\Delta(k)$)b	83.88	84.05	**84.25**	80.38	80.93	80.1	84.56	84.25	83.74	86.6	**87.92**	86.4
(o,t)b	79.25	79	80.12	75	79.93	80.12	80.75	83.5	80.75	84.69	85.38	83.7

Таблица 8: Точность классификации (в %) различных модификаций TF-IDF. Биграммы.

	dvd	dvd	dvd	эл-ка	эл-ка	эл-ка	кни-ги	кни-ги	кни-ги	к/ф	к/ф	к/ф
	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB	NN	SVM	NB
(1,2)	83.5	83	80.75	81.12	79.14	80.44	83.31	83.62	84.94	85.63	87.38	84.44
(1,3)	82.5	82.75	81.25	80.62	79.57	79.5	83	83.69	85.69	84.31	87.19	85.19
(n,t)	83.25	83	83.19	80	81.79	81.69	82.13	85.88	**86.56**	86.31	87.69	86.38
(l,$\Delta(k)$)	83.75	83.25	83	**81.86**	80.75	80.14	83.1	82.75	80.63	86.19	87.44	86
(l,$\Delta(k)$)b	83.88	84.05	**84.25**	80.38	80.93	80.1	84.56	84.25	83.74	86.6	**87.92**	86.4

Таблица 9: Сравнение различных моделей представления данных.

4.6 Композиция методов

В данной работе были рассмотрены два способа композиции классификаторов: голосование и композиция наивного байесовского классификатора и SVM. Рассмотрим подробнее каждый из этих способов.

Обучим несколько наивных байесовских классификаторов на различных моделях представления данных. С помощью процедуры скользящего контроля выберем те модели, на которых обученные классификаторы, дают наилучшие результаты. Так как байесовский классификатор оперирует вероятностями, можно получить оценку вероятности отнесения каждого объекта к определенному классу. Эти вероятности и будем подавать на вход SVM. В настоящей работе были рассмотрены различные комбинации моделей представления признаков, наилучшие комбинации выписаны в таблице 10.

Другой, рассмотренный в данной работе, метод композиции классификаторов — это голосование по большинству. Выберем наилучшие модели признаковых представлений и обучим на них различные классификаторы. Для всех документов каждый из классификаторов проголосует за свой класс, в качестве ответа выберем тот класс, за который наберется больше всего голосов.

Качество классификации при композиции нескольких методов машинного обучения является наилучшим на скользящем контроле. Посмотрим, как изменится точность классификации на скрытой выборке. В таблицах 10 — 14 содержатся резуль-

набор данных	комбинация	cv	ht
dvd	(n,t)+(n,t)b+(1,4)+(l,$\Delta(k)$)b	85.98	83.25
эл-ка	(1,2)+(n,t)+(l,k)+(n,t)b	82.14	81.25
книги	(1,3)+(1,4)+(n,t)	87	85.25
к/ф	(1,2) + (1,3) +(1,4)+ (n,t)	87.38	86.5

Таблица 10: NB + SVM.

набор данных	комбинация	cv	ht
dvd	(n,t)+(1,4)+(l,$\Delta(k)$)b	85.78	83.75
эл-ка	(n,t)+(l,k)+(n,t)b	81.7	80.25
книги	(1,3)+(1,4)+(n,t)	86.98	85
к/ф	(1,3) +(1,4)+ (n,t)	86.31	85.5

Таблица 11: Голосование NB.

таты классификации данных методов на скользящем контроле и на проверочной тестовой выборке. Как и ожидалось, точность классификации на проверочной выборке падает.

4.7 Выводы

- Наилучшие результаты удалось получить с помощью применения композиции алгоритмов. Оба рассмотренных метода построения композиции показали сопоставимые результаты (см. таблицу 14).
- Для наборов отзывов на электронику и кинофильмы наилучшие результаты были получены при использовании голосования по большинству с классификатором SVM.
- Для отзывов на dvd продукцию максимум точности классификации был достигнут, также, с помощью голосование по большинству, но уже с нейронной сетью.

набор данных	комбинация	cv	ht
dvd	(1,2)+(l,$\Delta(k)$)+(l,$\Delta(k)$)b	86.02	84.5
эл-ка	(1,2)+(1,4)+(l,$\Delta(k)$)	82.14	81
книги	(1,2)+(l,$\Delta(k)$)+(l,$\Delta(k)$)b	84.1	81.5
к/ф	(1,1)+(n,t)+(n,t)b	86.56	85.75

Таблица 12: Голосование NN.

набор данных	комбинация	cv	ht
dvd	(n,t)b+(1,4)+(l,$\Delta(k)$)b	85.31	82.25
эл-ка	(n,t)+(l,k)+(l,k)b	82.21	81.75
книги	(1,3)+(n,t)+(n,t)b	86.44	82.25
к/ф	(1,2)+(1,3)+(n,t)+(l,$\Delta(k)$)+(l,$\Delta(k)$)b	88.19	87.5

Таблица 13: Голосование SVM.

- Композиция наивного байесовского классификатора и SVM показала наилучшие результаты на наборе отзывов на книги.
- Во всех комбинациях были использованы классификаторы, обученные на нормированных признаках. Экспериментально было установлено, что нормирование признаков дает небольшой прирост качества (1–2 %).
- На всех наборах данных линейный SVM дает точность классификации выше, чем SVM с RBF ядром.
- Точность классификации всех выборок выше у TF-IDF модели, чем у простой бинарной (частотной) n-граммной модели.
- Сравнение простых бинарных и частотных моделей позволяет сделать вывод, о том, что частотное представление избыточно, и при бинарном представлении результаты получаются лучше.
- Среди бинарных n-граммных моделей наилучшие результаты получаются при представлении текста в виде комбинации униграмм и биграмм (в некторых слу-

набор данных	метод	cv	ht
dvd	(1,2)+(l,$\Delta(k)$)+(l,$\Delta(k)$)b, NN vote	86.02	84.5
эл-ка	(n,t)+(l,k)+(l,k)b, SVM vote	82.21	81.75
книги	(1,3)+(1,4)+(n,t), NB+SVM	87	85.25
к/ф	(1,2)+(1,3)+(n,t)+(l,$\Delta(k)$)+(l,$\Delta(k)$)b, SVM vote	88.19	87.5

Таблица 14: Наилучшие результаты.

чаях триграмм). Дальнейшее увеличение длин n-грамм приводит к ухудшению качества классификации.

5 Заключение

В данной работе были рассмотрены существующие подходы к классификации текстов по эмоциональной окраске, также было произведено сравнение качества работы этих подходов на реальных данных. В результате экспериментов, в ходе которых для каждой из рассмотренных моделей были подобраны оптимальные параметры, выяснилось, что наиболее высокого качества можно достичь при использовании композиции алгоритмов. Для каждого использованного набора данных базовое семейство алгоритмов подбиралось индивидуально.

Для получения хорошего качества классификации важно также правильно подобрать признаковое описание текстов. Предложенная в данной работе идея использования биграммной модели с представлением весов слов с помощью TF-IDF позволила получить наилучшие результаты на нескольких наборах данных среди всех рассмотренных признаковых представлений.

5.1 Положения выносимые на защиту

- Программная реализация различных методов, используемых для анализа эмоциональной окраски текстов;
- Реализация программного стенда для проведения экспериментов;

- Проведение экспериментов и сравнение различных методов на наборах реальных данных;
- Реализация алгоритмической композиции методов.

Список литературы

[1] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python.* "O'Reilly Media, Inc. 2009.

[2] B. Heerschop, F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong. Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1061–1070. ACM, 2011.

[3] W. Mann and S. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[4] G. Paltoglou and M. Thelwall. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1386–1395. Association for Computational Linguistics, 2010.

[5] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[6] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[7] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[8] M. Taboada, K. Voll, and J. Brooke. Extracting sentiment as a function of discourse structure and topicality. *Simon Fraser Univeristy School of Computing Science Technical Report*, 2008.

[9] P. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[10] R. Socher, A. Perelygin Recursive deep models for semantic compositionality over a sentiment treebank, In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, pages 1631–1642, 2013.

[11] К. Бюлер. Теория языка.Репрезентативная функция языка. *М.: Прогресс.*, 2000.

[12] А. Сусов. Моделирование дискурса в терминах теории риторической структуры. *Вестник Воронежского государственного университета. Серия: Филология. Журналистика*, (2):133–138, 2006.

A Исходный код экспериментов.

Listing 1: Модуль для предобработки текста и разделения его на две выборки

```
# makes text preprocessing
# makes partition to hidden test set and cv set
# saves files in svmlight format
import os
import pdb

import nltk
import numpy as np
import math

from nltk.stem import *
from sklearn.datasets import dump_svmlight_file
from sklearn.cross_validation import StratifiedKFold
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import normalize
from scipy.sparse import csr_matrix

# functions for TF-IDF model
def tf(corpus):
    return normalize(corpus, norm='l1', axis=1)

def num_docs_containing(index_word, corpus):
    return corpus[:, index_word].indices.shape[0]

def idf(corpus, corpus_labels, method):
    idf_data = np.zeros((1, corpus.shape[1]), dtype=np.float)
    if method == "delta-t" or method == "delta-t-":
        indices_positive = np.where(corpus_labels == -1)[0]
        indices_negative = np.where(corpus_labels ==  1)[0]
        N_positive = len(indices_positive)
        N_negative = len(indices_negative)
        for i in xrange(idf_data.shape[1]):
            df_positive = float(num_docs_containing(i,
                corpus[indices_positive, ]))
            df_negative = float(num_docs_containing(i,
                corpus[indices_negative, ]))
            if method == "delta-t":
                idf_data[0, i] = math.log((N_positive * df_negative)
                    / \
                                    (N_negative * df_positive))
            else:
                idf_data[0, i] = math.log((N_positive * df_negative
                    + \
                                   0.5) / (N_negative * df_positive +
                                      0.5))
    elif method == "n":
```

```
42          for i in xrange(idf_data.shape[1]):
43              idf_data[0, i] = float(num_docs_containing(i, corpus))
44      elif method == "t":
45          for i in xrange(idf_data.shape[1]):
46              idf_data[0, i] = math.log(corpus.shape[0] / \
47                                        float(num_docs_containing(i, corpus)))
48      elif method == "p" or method == "k":
49          for i in xrange(idf_data.shape[1]):
50              df = float(num_docs_containing(i, corpus))
51              if method == "p":
52                  idf_data[0, i] = math.log((corpus.shape[0] - df) /
                        df)
53              else:
54                  idf_data[0, i] = math.log((corpus.shape[0] - df +
                        0.5)/\
55                                            (df + 0.5))
56      elif method == "delta-k":
57          indices_positive = np.where(corpus_labels == -1)[0]
58          indices_negative = np.where(corpus_labels ==  1)[0]
59          N_positive = len(indices_positive)
60          N_negative = len(indices_negative)
61          for i in xrange(idf_data.shape[1]):
62              df_positive = float(num_docs_containing(i,
                    corpus[indices_positive, ]))
63              df_negative = float(num_docs_containing(i,
                    corpus[indices_negative, ]))
64              idf_data[0, i] = math.log(((N_positive - df_positive + \
65                                         0.5) * df_negative + 0.5) / \
66                                        ((N_negative - df_negative + 0.5) * \
67                                          df_positive + 0.5))
68      return idf_data
69
70  def tf_idf(data_set, idf_data, method):
71      d = csr_matrix(data_set.shape)
72      tf_data = tf(data_set)
73      if method == "n":
74          for i in xrange(data_set.shape[0]):
75              indices = data_set[i, ].indices
76              for ind in indices:
77                  d[i, ind] = tf_data[i, ind] * idf_data[0, ind]
78      elif method == "l":
79          for i in xrange(data_set.shape[0]):
80              indices = data_set[i, ].indices
81              for ind in indices:
82                  d[i, ind] = (1 + math.log(tf_data[i, ind])) * \
83                              idf_data[0, ind]
84      elif method == "a":
85          max_t = [tf_data[:, i].max() for i in
                xrange(tf_data.shape[1])]
86          for i in xrange(data_set.shape[0]):
```

```
87              indices = data_set[i, ].indices
88              for ind in indices:
89                  d[i, ind] = (0.5 + 0.5 * tf_data[i, ind] /
                        max_t[ind])*\
90                              idf_data[0, ind]
91      elif method == "o":
92          avg_dl = data_set.sum() / data_set.shape[0]
93          k1 = 1.2
94          b = 0.95
95          for i in xrange(data_set.shape[0]):
96              indices = data_set[i, ].indices
97              dl_to_avg = data_set[i, ].sum() / avg_dl
98              for ind in indices:
99                  tf_ind = (k1 + 1) * tf_data[i, ind] / \
100                             (k1 * (1 - b + b * dl_to_avg) + \
101                             tf_data[i, ind])
102                 d[i, ind] = tf_ind * idf_data[0, ind]
103
104     return d
105 # reading file from XML format
106 folder = "dvd"
107 f = open("../dataset/" + folder + "/positive.txt","r")
108 tmp = ""
109 data = []
110 its_text = False
111 end_of_text = False
112 for line in f:
113     if line == '<review_text>\n':
114         its_text = True
115         continue
116     if line == '</review_text>\n':
117         end_of_text = True
118     if its_text and not end_of_text:
119         tmp = tmp + line
120     if end_of_text:
121         data.append(tmp)
122         tmp = ""
123         end_of_text = False
124         its_text    = False
125 f.close()
126 #stemming
127 stemmer = PorterStemmer()
128 for i in xrange(len(data)):
129     doc_list = nltk.word_tokenize(data[i])
130     stemmed_doc = [stemmer.stem(word) for word in doc_list]
131     data[i] = " ".join(stemmed_doc)
132 labels = np.concatenate((-1 * np.ones(1000), np.ones(1000)))
133 # define parameters
134 params = {}
135 params["ngram"]  = (1,1)
```

```
params["min_df"] = 7
params["max_df"] = 0.9
params["bin"]    = True
params["method"] = "TF-IDF"
params["norm"]   = "l2"
params["tf"]  = "a"
params["idf"] = "t"
# for sentiword model
if params["method"] == "sentiword_struct":
    c = CountVectorizer(binary=params["bin"],
        ngram_range=params["ngram"], \
                    min_df=params["min_df"], stop_words=None,
                    tokenizer=nltk.word_tokenize,
                        max_df=params["max_df"])
    data = c.fit_transform(data)
    feature_names = c.get_feature_names()
    pos_score = []
    for f in feature_names:
        pos = 0
        synset = swn.senti_synsets(f[0])
        for s in synset:
            pos = pos + s.pos_score()
        if len(synset) != 0:
            pos = pos / len(synset)
        pos_score.append(pos)
    data = data.astype(np.float)
    for i in xrange(data_vec.shape[0]):
        ind = data[i, :].indices
        for index in ind:
            data[i, index] = (pos_score[index])
# subfolder, where the data will be saved to
subfolder = str(params["bin"]) + str(params["ngram"]) + \
            str(params["min_df"])
if params["method"] == "TF-IDF":
    subfolder = str(params["tf"]) + str(params["idf"]) + "_" + \
                str(params["min_df"]) + str(params["ngram"])
elif params["method"] == "sentiment-structural":
    subfolder = str("sentiment-structural" + params["bin"]) + \
                str(params["ngram"]) + str(params["min_df"])
subfolder = ''.join(subfolder.split())
if not os.path.exists("data_vw/" + folder + "/" + subfolder):
    os.mkdir("data_vw/" + folder + "/" + subfolder)

# hidden set partition
test_index  = set(range(200) + range(1000, 1200))
train_index = set(range(2000)) - test_index
hide_train  = [data[i] for i in train_index]
hide_test   = [data[i] for i in test_index]
y_train     = [labels[i] for i in train_index]
y_test      = [labels[i] for i in test_index]
```

```
c = CountVectorizer(binary=params["bin"], ngram_range=params["ngram"], \
                    min_df=params["min_df"], \
                    stop_words=None, tokenizer=nltk.word_tokenize,\
                    max_df=params["max_df"])
hide_train = c.fit_transform(hide_train)
hide_test  = c.transform(hide_test)
hide_train = hide_train.astype(np.float)
hide_test  = hide_test.astype(np.float)
if params["method"] == "TF-IDF":
    idf_data   = idf(hide_train, y_train, params["idf"])
    hide_train = tf_idf(hide_train, idf_data, params["tf"])
    hide_test  = tf_idf(hide_test,  idf_data, params["tf"])
# saving results
dump_svmlight_file(hide_train, y_train, f="data_vw/" + folder + "/" +
                   subfolder + "/hide_train.txt", zero_based=False)
dump_svmlight_file(hide_test, y_test,   f="data_vw/" + folder + "/" +
                   subfolder + "/hide_test.txt", zero_based=False)
if params["norm"] != "l2":
    train_normalized = normalize(hide_train, norm=params["norm"],
       axis=1)
    test_normalized  = normalize(hide_test,  norm=params["norm"],
       axis=1)
    dump_svmlight_file(train_normalized, y_train, f="data_vw/" +
                       folder + "/" + subfolder + "/" +
                          params["norm"] +
                        "_hid_train.txt", zero_based=False)
    dump_svmlight_file(test_normalized, y_test,   f="data_vw/" +
                       folder + "/" + subfolder + "/" +
                          params["norm"] +
                       "_hid_test.txt", zero_based=False)
# cv partition on 8 folds
indexes    = set(range(0, 2000)) - set(test_index)
data_cut   = [data[i] for i in indexes]
labels_cut = [labels[i] for i in indexes]
skf = StratifiedKFold(labels_cut, 8, random_state=1)
n_fold = 0
for train_index, test_index in skf:
    n_fold  = n_fold + 1
    X_train = [data_cut[i] for i in train_index]
    X_test  = [data_cut[i] for i in test_index]
    y_train = [labels_cut[i] for i in train_index]
    y_test  = [labels_cut[i] for i in test_index]
    c = CountVectorizer(binary=params["bin"], \
                        ngram_range=params["ngram"], \
                        min_df=params["min_df"],\
                        stop_words=None,
                            tokenizer=nltk.word_tokenize,\
                        max_df=params["max_df"])
    train_set = c.fit_transform(X_train)
```

```
228     test_set  = c.transform(X_test)
229     train_set = train_set.astype(np.float)
230     test_set  = test_set.astype(np.float)
231     if params["method"] == "TF-IDF":
232         idf_data = idf(train_set, y_train, params["idf"])
233         train_set  = tf_idf(train_set, idf_data, params["tf"])
234         test_set   = tf_idf(test_set, idf_data, params["tf"])
235     dump_svmlight_file(train_set, y_train, f="data_vw/" + folder +
236                        "/" + subfolder + "/train" + str(n_fold) +
237                        ".txt", zero_based=False)
238     dump_svmlight_file(test_set, y_test, f="data_vw/" + folder + "/"
        +
239                        subfolder + "/test" + str(n_fold) + ".txt",\
240                        zero_based=False)
241     if params["norm"] != "none":
242         train_normalized = normalize(train_set, norm=params["norm"],
            \
243                              axis=1)
244         test_normalized = normalize(test_set,  norm=params["norm"], \
245                             axis=1)
246         dump_svmlight_file(train_normalized, y_train, f="data_vw/" +
247                             folder + "/" + subfolder +  "/" +
248                             params["norm"] + "_train" +
249                             str(n_fold) + ".txt", zero_based=False)
250         dump_svmlight_file(test_normalized, y_test, f="data_vw/" +
251                             folder + "/" + subfolder + "/" +
252                             params["norm"] + "_test" +
253                             str(n_fold) + ".txt", zero_based=False)
```

Listing 2: Модуль для проведения классификации с помощью NB

```
1  #makes NB classification
2  import numpy as np
3  import pdb
4
5  from sklearn.datasets import load_svmlight_file
6  from sklearn.svm import LinearSVC
7  from sklearn.naive_bayes import MultinomialNB
8  from user_log import log
9
10 folder = "data_vw/dvd"
11 params = {}
12 params["ngram"]  = (1,1)
13 params["min_df"] = 7
14 params["max_df"] = 0.9
15 params["bin"]    = True
16 params["method"] = "TF-IDF"
17 params["norm"]   = "l2"
18 params["tf"]  = "a"
19 params["idf"] = "t"
20 # subfolder with the stored data
```

```
subfolder = str(params["bin"]) + str(params["ngram"]) + \
            str(params["min_df"])
if params["method"] == "TF-IDF":
    subfolder = str(params["tf"]) + str(params["idf"]) + "_" + \
                str(params["min_df"]) + str(params["ngram"])
elif params["method"] == "sentiment-structural":
    subfolder = str("sentiment-structural" + params["bin"]) + \
                str(params["ngram"]) + str(params["min_df"])
subfolder = ''.join(subfolder.split())
precision      = []
precision_norm = []
precision_hide = []
train_set, y_train = load_svmlight_file(folder + "/" + subfolder +
                     "/hid_train.txt", zero_based=False)
test_set, y_test   = load_svmlight_file(folder + "/" + subfolder +
                     "/hid_test.txt", zero_based=False, \
                     n_features=train_set.shape[1])
cl = MultinomialNB().fit(train_set, y_train)
predicted_labels = cl.predict(test_set)
tmp = 100 * float(sum(predicted_labels == y_test)) / len(y_test)
precision_hide.append(tmp)
if params["norm"] != "none":
    train_set, y_train = load_svmlight_file(folder + "/" + subfolder
       +
                         "/" + params["norm"] + "_hid_train.txt")
    test_set, y_test = load_svmlight_file(folder + "/" + subfolder +
                        "/" + params["norm"] + "_hid_test.txt", \
                        zero_based=False,
                           n_features=train_set.shape[1])
    cl = MultinomialNB().fit(train_set, y_train)
    predicted_labels = cl.predict(test_set)
    tmp = 100 * float(sum(predicted_labels == y_test)) / len(y_test)
    precision_hide.append(tmp)
for i in range(1,9):
    train_set, y_train = load_svmlight_file(folder + "/" +
                         subfolder + "/train" + str(i) + ".txt", \
                         zero_based=False)
    test_set, y_test = load_svmlight_file(folder + "/" +
                       subfolder + "/test" + str(i) + ".txt", \
                       zero_based=False, \
                       n_features=train_set.shape[1])
    cl = MultinomialNB().fit(train_set, y_train)
    predicted_labels = cl.predict(test_set)
    tmp = 100 * float(sum(predicted_labels == y_test)) / len(y_test)
    precision.append(tmp)
    if params["norm"] != "none":
        train_set, y_train = load_svmlight_file(folder + "/" +
                             subfolder + "/" + params["norm"] +
                             "_train" + str(i) + ".txt", \
                             zero_based=False)
```

```
69          test_set, y_test = load_svmlight_file(folder + "/" +
70                                  subfolder + "/" + params["norm"] +
71                                  "_test" + str(i) + ".txt",
                                       zero_based=False,\
72                                  n_features=train_set.shape[1])
73          cl = MultinomialNB().fit(train_set, y_train)
74          predicted_labels = cl.predict(test_set)
75          tmp = 100 * float(sum(predicted_labels == y_test)) / \
76                len(y_test)
77          precision_norm.append(tmp)
```

Listing 3: Модуль для проведения классификации с помощью SVM

```
1  # makes svm classification
2  import numpy as np
3  import pdb
4
5  from sklearn.datasets import load_svmlight_file
6  from sklearn.svm import LinearSVC
7  from sklearn.svm import SVC
8  from user_log import log
9
10 folder = "data_vw/dvd"
11 #define parameters
12 params = {}
13 params["ngram"]  = (1,1)
14 params["min_df"] = 7
15 params["max_df"] = 0.9
16 params["bin"]    = True
17 params["method"] = "TF-IDF"
18 params["norm"]   = "l2"
19 params["tf"]  = "a"
20 params["idf"] = "t"
21 # subfolder with the stored data
22 subfolder = str(params["bin"]) + str(params["ngram"]) + \
23             str(params["min_df"])
24 if params["method"] == "TF-IDF":
25     subfolder = str(params["tf"]) + str(params["idf"]) + "_" + \
26                 str(params["min_df"]) + str(params["ngram"])
27 elif params["method"] == "sentiment-structural":
28     subfolder = str("sentiment-structural" + params["bin"]) + \
29                 str(params["ngram"]) + str(params["min_df"])
30 subfolder = ''.join(subfolder.split())
31 precision      = []
32 precision_norm  = []
33 precision_hide = []
34 CS = [0.01, 0.05, 0.1, 0.2, 0.5, 1, 1.5, 2]
35 for coef in CS:
36     precision_c      = []
37     precision_l1_c   = []
38     precision_l2_c   = []
```

```
39      precision_hide_c = []
40      train_set, y_train = load_svmlight_file(folder + "/" + subfolder
           +
41                              "/hid_train.txt", zero_based=False)
42      test_set, y_test   = load_svmlight_file(folder + "/" + subfolder
           +
43                              "/hid_test.txt", zero_based=False, \
44                              n_features=train_set.shape[1])
45      cl = LinearSVC(C=coef).fit(train_set, y_train)
46      #cl = SVC(C=coef, gamma=gamma_coef).fit(train_set, y_train)
47      predicted_labels = cl.predict(test_set)
48      tmp = 100 * float(sum(predicted_labels == y_test)) / len(y_test)
49      precision_hide_c.append(tmp)
50      if params["norm"] != "none":
51          train_set, y_train = load_svmlight_file(folder + "/" +
52                                  subfolder + "/" + params["norm"] +
53                                  "_hid_train.txt")
54          test_set, y_test = load_svmlight_file(folder + "/" +
               subfolder +
55                                "/" + params["norm"] + "_hid_test.txt", \
56                                zero_based=False,
                                    n_features=train_set.shape[1])
57          cl = LinearSVC(C=coef).fit(train_set, y_train)
58          #cl = SVC(C=coef, gamma=gamma_coef).fit(train_set, y_train)
59          predicted_labels = cl.predict(test_set)
60          tmp = 100 * float(sum(predicted_labels == y_test)) /
               len(y_test)
61          precision_hide_c.append(tmp)
62      for i in range(1,9):
63          train_set, y_train = load_svmlight_file(folder + "/" +
64                                  subfolder + "/train" + str(i) + ".txt",
                                       \
65                                  zero_based=False)
66          test_set, y_test = load_svmlight_file(folder + "/" +
67                                subfolder + "/test" + str(i) + ".txt", \
68                                zero_based=False, \
69                                n_features=train_set.shape[1])
70          cl = LinearSVC(C=coef).fit(train_set, y_train)
71          #cl = SVC(C=coef, gamma=gamma_coef).fit(train_set, y_train)
72          predicted_labels = cl.predict(test_set)
73          tmp = 100 * float(sum(predicted_labels == y_test)) /
               len(y_test)
74          precision_c.append(tmp)
75          if params["norm"] != "none":
76              train_set, y_train = load_svmlight_file(folder + "/" +
77                                      subfolder + "/" + params["norm"] +
78                                      "_train" + str(i) + ".txt", \
79                                      zero_based=False)
80              test_set, y_test = load_svmlight_file(folder + "/" +
81                                    subfolder + "/" + params["norm"] +
```

```
                                    "_test" + str(i) + ".txt", \
                                    zero_based=False,\
                                    n_features=train_set.shape[1])
            cl = LinearSVC(C=coef).fit(train_set, y_train)
            #cl = SVC(C=coef, gamma=gamma_coef).fit(train_set,
                y_train)
            predicted_labels = cl.predict(test_set)
            tmp = 100 * float(sum(predicted_labels == y_test)) / \
                  len(y_test)
            precision_norm_c.append(tmp)
    precision.append(sum(precision_c) / len(precision_c))
    precision_norm.append(sum(precision_norm_c) /
        len(precision_norm_c))
    precision_hide.append(sum(precision_hide_c) /
        len(precision_hide_c))
```

Listing 4: Модуль для проведения экспериментов с композицей алгоритмов через голосование

```
# makes composition by voting
import sys
import pdb

import numpy as np

from sklearn.cross_validation import StratifiedKFold
from sklearn.svm import LinearSVC
from sklearn.svm import SVC

#folders that contain best predictions
best_ps     = ["ot_3(2,2)", "nt_3(2,2)", "True(1,4)3"]
folder      = "data_vw/books/"
n_predicted = 200
ps_folders  = []
for i in xrange(8):
    ps_folders.append(np.zeros((n_predicted, len(best_ps))))
ps_hid = np.zeros((n_predicted * 2, len(best_ps)))
for i in xrange(len(best_ps)):
    f = open(folder + best_ps[i] + "/nb/predictions.txt")
    predicted = []
    for line in f:
        index = line.index(':') + 1
        line = line[index : len(line) - 1]
        tmp = 1 - 2 * (float(line) > 0.5)
        predicted.append(tmp)
    f.close()
    ps_hid[:, i] = predicted
    for j in xrange(8):
        f = open(folder + best_ps[i] + "/nb/predictions_fold" +
                 str(j + 1) + ".txt")
        predicted = []
        for line in f:
```

```
            index = line.index(':') + 1
            line = line[index : len(line) - 1]
            tmp = 1 - 2 * (float(line) > 0.5)
            predicted.append(tmp)
        f.close()
        ps_folders[j][:, i] = predicted

precision = []
true_labels = np.concatenate((-1 * np.ones(len(predicted) / 2), \
                              np.ones(len(predicted) / 2)))
for fold in ps_folders:
    predicted = np.array(fold.sum(1))
    predicted[np.where(predicted > 0)] = 1
    predicted[np.where(predicted < 0)] = -1
    precision.append(100 * float(sum(predicted == true_labels)) / \
                     len(true_labels))

predicted = np.array(ps_hid.sum(1))
true_labels = np.concatenate((-1 * np.ones(len(predicted) / 2), \
                              np.ones(len(predicted) / 2)))
predicted[np.where(predicted > 0)] = 1
predicted[np.where(predicted < 0)] = -1
precision_hide = float(100 * sum(predicted == true_labels)) / \
                    len(true_labels)
precision = sum(precision) / len(precision)
```

Listing 5: Модуль для проведения экспериментов с композицей алгоритмов NB + SVM

```
# makes composition of NB and SVM
import numpy as np
import pdb

from sklearn.svm import LinearSVC
from sklearn.svm import SVC

best_ps =   [ "True(1,3)3", "True(1,4)3","ot_3(2,2)", "nt_3(2,2)"]
folder = "data_vw/books/"
name =  "fold"
hide_name =  "hid"
n_predicted = 200
ps_folders = []
for i in xrange(8):
    ps_folders.append(np.zeros((n_predicted, len(best_ps))))

ps_hid = np.zeros((n_predicted * 2, len(best_ps)))
for i in xrange(len(best_ps)):
    f = open(folder + best_ps[i] + "/nb/" + hid_name + ".txt")
    predicted = []
    for line in f:
        index = line.index(':') + 1
        line = line[index : len(line) - 1]
```

```
24          tmp = float(line)
25          predicted.append(tmp)
26      f.close()
27      ps_hid[:, i] = predicted
28      for j in xrange(8):
29          f = open(folder + best_ps[i] + "/nb/" + \
30                   name + str(j + 1) + ".txt")
31          predicted = []
32          for line in f:
33              index = line.index(':') + 1
34              line = line[index : len(line) - 1]
35              tmp = float(line)
36              predicted.append(tmp)
37          f.close()
38          ps_folders[j][:, i] = predicted
39
40  precision = []
41  precision_hide = []
42  CS = [0.001, 0.01, 0.05, 0.1, 0.5, 1, 2]
43  for coef in CS:
44      true_labels = np.concatenate((-1 * np.ones(len(predicted) / 2), \
45                    np.ones(len(predicted) / 2)))
46      precision_c = []
47      test_index = range(n_predicted / 4) + \
48                   range(n_predicted / 2, 3 * n_predicted / 4)
49      train_index = range(n_predicted / 4, n_predicted / 2) + \
50                    range(n_predicted * 3 / 4, n_predicted)
51      for fold in ps_folders:
52          train_set = fold[train_index, :]
53          test_set  = fold[test_index, :]
54          y_train, y_test = true_labels[train_index],
                true_labels[test_index]
55          cl = LinearSVC(C=coef).fit(train_set, y_train)
56          predicted_labels = cl.predict(test_set)
57          tmp = float(sum(predicted_labels == y_test)) / len(y_test)
58          precision_c.append(tmp)
59      precision.append(100 * sum(precision_c) / len(precision_c))
60      true_labels = np.concatenate((-1 * np.ones(len(predicted)), \
61                    np.ones(len(predicted))))
62      n_predicted = len(true_labels)
63      test_index = range(n_predicted / 4) + \
64                   range(n_predicted / 2, 3 * n_predicted / 4)
65      train_index = range(n_predicted / 4, n_predicted / 2) + \
66                    range(n_predicted * 3 / 4, n_predicted)
67      hid_train, hid_test = ps_hid[train_index, :], ps_hid[test_index,
            :]
68      y_train, y_test = true_labels[train_index],
            true_labels[test_index]
69      cl = LinearSVC(C=coef).fit(hid_train, y_train)
70      predicted_labels = cl.predict(hid_test)
```

```
71      tmp = 100 * float(sum(predicted_labels == y_test)) / len(y_test)
72      precision_hide.append(tmp)
```

Listing 6: Модуль для экспериментов с PMI.

```
1  #calculates PMI using google
2  def hits(word_x, word_y):
3      query =
           "http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=%s"
4      results = urllib.urlopen(query % word_x + " AROUND(10) " +
           word_y)
5      json_res = json.loads(results.read())
6      google_hits = int(json_res['responseData']\
7                        ['cursor']['estimatedResultCount'])
8      return google_hits
9
10 def so(word, hits_poor, hits_excellent):
11     return log((hits(word,"excellent")  * hits_poor) / \
12                (hits(word,"poor") * hits_excellent), 2)
```

Listing 7: Модуль для перевода из формата svmligth в VW формат

```
1  # script for making VW format file from svmlight format
2  # 1 - way to the svmlight
3  # 2 - way to the VW format file
4  old_file=$1
5  new_file=$2
6  perl -pe 's/\s/ | /' $old_file >> $new_file
```

Listing 8: Модуль для проведения эесперимента с NN VW

```
1  # script for running experiments
2  # with NN classification using VW
3  # INPUT:
4  # 1 - way to the train file
5  # 2 - way to the test file
6  # 3 - way to the result file
7  # 4 - way to the model file
8  # OUTPUT:
9  # PRECISIONS - an array of precisions
10 # first dimension - rate
11 # second dimension - number of hidden units
12 TRAIN=$1
13 TEST=$2
14 RESULT=$3
15 MODEL=$4
16 PASSES=250
17 INDEX_H_U=0
18 for HIDDEN_UNITS in  4 7 8 10 15 16 31
19 do
20   INDEX_R=0
21   for RATE in 0.4 0.5
```

```
22    do
23      sh experiment_one_fold.sh $TRAIN $TEST $PASSES $HIDDEN_UNITS
           $RATE $RESULT $MODEL
24      INDEX=0
25      filename=$RESULT
26      while read -r line
27      do
28          a[$INDEX]=$line
29          INDEX=$((INDEX+1))
30      done < "$filename"
31      SUM=$( IFS="+"; bc <<< "${a[*]}" )
32      LENGTH=${#a[*]}
33      unset a
34      PRECISION=`calc "$SUM*100/$LENGTH"`
35      PRECISION_RATE[$INDEX_R]=$PRECISION
36      INDEX_R=$(($INDEX_R+1))
37    done
38    PRECISIONS[$INDEX_H_U]=${PRECISION_RATE[*]}
39    INDEX_R_H=$(($INDEX_R_H+1))
40  done
```

Listing 9: Модуль для проведения эесперементов с NN VW

```
1  # script for running experiment for one fold
2  # INPUT:
3  # 1 - way to the train file
4  # 2 - way to the test file
5  # 3 - number of passes
6  # 4 - number of hidden units
7  # 5 - rate
8  # 6 - way to the result file
9  # 7 - way to the model file
10 TRAIN=$1
11 TEST=$2
12 PASSES=$3
13 HIDDEN_UNITS=$4
14 RATE=$5
15 RESULT=$6
16 MODEL=$7
17 PARAMS_TRAIN="--quiet -k -c -f $MODEL --binary --random_seed 1
       --passes $PASSES -l $RATE --nn $HIDDEN_UNITS"
18 PARAMS_TEST="--quiet -k -c -i $MODEL --random_seed 1 -p $RESULT"
19 vw -d $TRAIN $PARAMS_TRAIN
20 vw -t -d $TEST $PARAMS_TEST
21 rm $MODEL
```