Санкт-Петербургский государственный университет
Факультет прикладной математики — процессов управления

Кафедра технологии программирования

Голованов Егор Олегович

Применение методов машинного обучения в области обработки естественного языка

Выпускная квалификационная работа бакалавра

Направление 010400
Прикладная математика и информатика

Научный руководитель:
к. физ.-мат. наук, доцент
Просолупов Е. В.

Рецензент:
д. физ.-мат. наук, профессор
Матросов А. В.

Санкт-Петербург
2017

Оглавление

1. Введение

Если взглянуть на нынешний уровень доступа к информации, то можно обнаружить, что найти нужную информацию без каких-либо специальных средств почти невозможно.

Во многих случаях, чтобы как-то работать с этими данными, приходится сначала их классифицировать или кластеризовывать. У читателя может возникнуть вопрос, что это такое и зачем вообще это нужно?

Определение. Кластеризация (или кластерный анализ) — задача разбиения множества объектов на группы, называемые кластерами. Внутри каждого кластера должны оказаться похожие объекты, а объекты различных групп должны быть как можно более отличны.

Цели кластеризации

- Понимание структуры данных. Разбиение множества объектов на группы позволяет упростить их дальнейшую обработку, например, зная структуру данных, можно применять какой-то конкретный алгоритм для каждого кластера.
- Обнаружение новизны. Можно найти объект, который невозможно отнести ни к одной из групп.

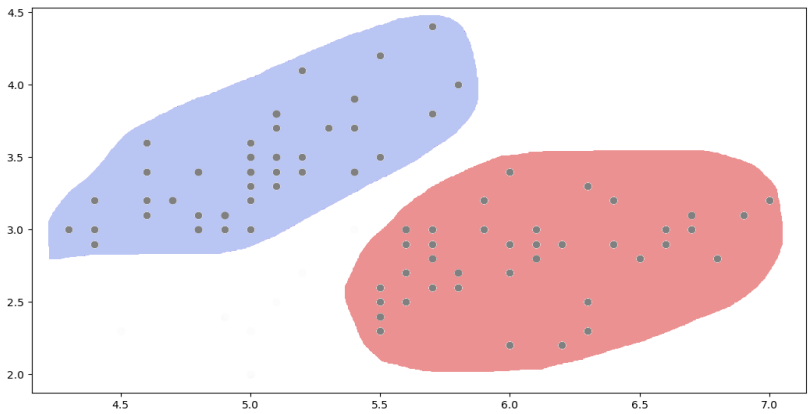

Рис. 1: Кластеризация

Определение. Классификация — задача состоит в присвоении каждому объекту из заданного множества, одной из заранее заданных групп.

Цели классификации

- Понимание структуры данных нового объекта. Присвоение группы объекту явно определяет его свойства и структуру. Эти свойства можно учесть при дальнейшей обработке.
- Предсказание данных. На основе известных данных можно предсказывать вероятность принадлежности объекта классификации к какому-либо классу.

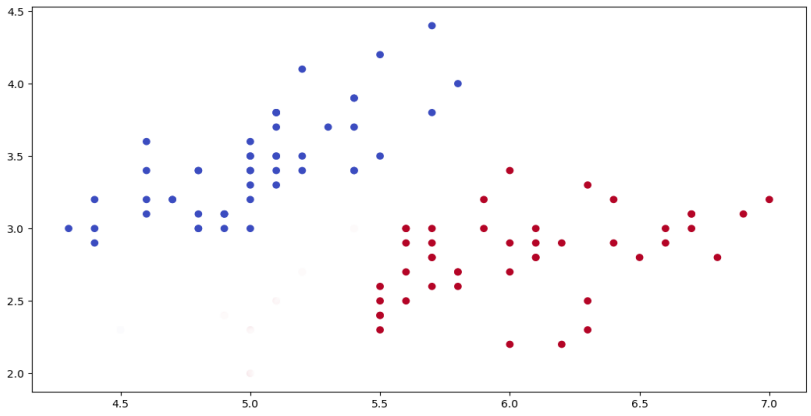

Рис. 2: Классификация

Отличием кластеризации от классификации является то, что перечень групп четко не задан и определяется в процессе работы алгоритма.

Многие крупные корпорации, такие как Яндекс или Google для обработки данных используют классификацию. Классификация является достаточно простой, но при этом часто решаемой задачей Data Mining. Компания Insecurityinsight, которая занимается мониторингом противоправных действий в отношении медицинких работников в горячих точках, по запросу которой реализован один из методов, рассмотренных в текущей работе, хочет выделять части преступления из статей(кто? когда? где? совершил преступление). Но для начала, им необходимо

научиться понимать, является ли статья криминальной или нет. Далее в этой работе будут рассмотрены только методы, решающие задачу *классификации*.

1.1. Формализация и постановка задачи

Пусть существует множество классов C = $\{c_1, c_2, \ldots, c_m\}$ и множество объектов X = $\{x_1, x_2, \ldots, x_i, \ldots\}$. Также существует функция $f: X \to C$, то есть такая функция, которая бы каждому объекту определяла класс. При этом, пусть для некоторых объектов $X_l = \{x_1, \ldots, x_l\}$ ($X_{learning}$) известны значения $y_i = \mathrm{f}(x_i)$. Тогда все пары (x_i, y_i) называются обучающей выборкой. Если $|\mathrm{C}| = 2$, то такая задача называется *бинарной классификацией*.

Возникает следующая проблема - объекты X имеют непредсказуемую природу. Элементом множества X может быть изображение, текст и т.д. Так как данная работа ориентируется на NLP(natural language processing), то будем считать, что элемент множества X - это текст (документ). Но к таким объектам нельзя напрямую применить существующий математический аппарат.

Рассмотрим задачу классификации подробнее. В задаче классификации можно выделить следующие этапы:

- Обработка текста
- Выделение признаков из текста
- Выбор классификатора
- Обучение классификатора и его практическое применение

Именно на этапе выделения признаков производится формирование пространства и перевод каждой статьи в это пространство. После этого математический аппарат использовать уже представляется возможным. Рассмотрим каждый из этапов отдельно.

1.2. Обработка текста

Этап обработки текста нужен, чтобы все слова, которые имеют одинаковую смысловую нагрузку, привести к одному виду, а также удалить слова, которые лишь мешают классификации документа. Для этого используются следующие методы:

- Стемминг слов
- Удаление стоп-слов
- Приведение слов к нижнему регистру

При стемминге происходит приведение слов к их основе. Основа слова - та часть слова, которая выражает его лексическое значение. Так, слова ”final” и ”finalize” преобразуются к ”final”.

Следующим этапом происходит удаление стоп-слов. Стоп-слова - это слова, которые не несут смысловой нагрузки, но при этом часто употребляются в тексте. Примером могут служить слова ”be”, ”was”, ”is” и т.д.

Также стоит приводить слова к одному и тому же регистру, иначе слова ”Victim”, ”victim” и ”ViCtim”, обозначающие одно и тоже, будут считаться разными.

При обработке любого документа в данной работе все слова приводятся к нижнему регистру, удаляются стоп-слова и используется стеммер Портера. Алгоритм Портера последовательно применяет ряд правил для отсечения суффиксов и окончаний слов, основываясь на особенностях языка. Вследствие этого, алгоритм достаточно быстр и при этом имеет достаточно высокую точность. Также любые слова, длина которых меньше либо равна двум символам, будут удалены из этого документа. Это объясняется тем, что в английском языке крайне мало слов, которые бы имели такую длину и при этом могли бы как-то охарактеризовать документ.

1.3. Выделение признаков

Существует множество моделей для преобразования текста в вектор, принадлежащий n-мерному пространству, лишь некоторые часто используемые:

- Bag of words
- Bag of n-grams
- TF-IDF

После того, как элемент множества X преобразуется через одну из вышеперечисленных моделей, получается n-мерный вектор, который характеризует этот элемент. Теперь, говоря о множестве X, будем понимать не первоначальные объекты, а их вектора, т.е. $X \subset R^n$.

Bag of words

Этот метод является одним из самых распространенных в практике. Суть метода заключается в том, что каждое слово рассматривается отдельно и "кладется в мешок". Таким образом, теряется первоначальный порядок слов и документ превращается в набор ("мешок") слов. При формировании "мешка" может учитываться количество вхождений слова, или же просто признак наличия или отсутствия. Также иногда применяют нормализацию, т.е. подсчет доли каждого слова во всех словах документа. Важно понимать, что каждому слову соответствует размерность пространства.

Пример. Рассмотрим как из предложения "Man was seriously injured and he was hospitalized" сформируется bag of words. Каждому слову в этом предложении поставим в соответствие частоту вхождения, т.е. "Man" = 1, "was" = 2, ..., "hospitalized" = 1. Таким образом, это предложение преобразуется в вектор $(1, 2, 1, ..., 1)$.

Bag of n-grams

Подход похож на bag of words, но вместо слов используются n-граммы. n-грамма - последовательность из n слов. Преимущество этого подхода

в том, что n-граммы сохраняют семантику. Этот подход имеет смысл использовать только в том случае, если предполагается, что появление текущего слова зависит только от n предыдущих.

Пример. Рассмотрим предложение "Man was seriously injured and he was hospitalized". Используем биграммы, тогда "мешок" будет состоять из ["Man", "Man was", "was seriously", "seriously injured", ..., "was hospitalized", "hospitalized"]. В общем случае n-граммы могут встречаться в тексте несколько раз, некоторые алгоритмы классификации могут учитывать эту информацию, другие - нет.

TF-IDF

TF-IDF (от англ. term frequency-inverse document frequency) состоит из трех этапов. После того, как слово проходит через эту модель, получается некоторая оценка этого слова, которая совмещает в себе и его важность, и его частоту вхождений.

Первым этапом выполняется подсчет *доли* вхождений каждого слова (TF).

$$TF(word) = \frac{W(word)}{A} \ ,$$

W(word) - количество вхождений слова word в документ;
A - количество всех слов в документе.

Вторым этапом выполняется измерение того, насколько важно это слово (IDF). При вычислении TF все слова считаются одинаково важными. Однако такие слова, как "say", "just" встречаются в документе достаточно часто, при этом они не характеризуют его.

$$IDF(word) = \log \frac{D}{DW(word)} \ ,$$

D - общее число документов;
DW(word) - количество документов, которые содержат слово word.

Третий этап совмещает первые две оценки.

$$RES(word) = TF(word) * IDF(word)$$

Пример. Допустим, документ содержит 100 слов, а слово "weapon" встретилось в документе 5 раз. TF("weapon") = 0.05. Пусть, при этом у нас есть 1000 документов, а слово "weapon" встретилось всего лишь в 10 из них. IDF("weapon") = 4,6. В итоге оценка TF-IDF("weapon") = 0.23.

Модель TF-IDF можно применять и к Bag of words, и к Bag of n-grams. В случае с Bag of words ничего не требуется изменять. Если же применять TF-IDF к Bag of n-grams, то во всех формулах нужно заменить word на n-грамму.

Получившееся пространство можно дополнить метрикой. Многие алгоритмы классификации нуждаются в расстоянии между двумя документами. Поэтому рассмотрим некоторые функции расстояния, которые в дальнейшем понадобятся.

Метрика Минковского

$$d(x, y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}, \quad x, y \subset R^n$$

При p = 2, является Евклидовым расстоянием.

Косинусное сходство

$$d(x, y) = \frac{\sum_{i=1}^{n} x_i * y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} * \sqrt{\sum_{i=1}^{n} y_i^2}}, \quad x, y \subset R^n$$

Расстояние Махаланобиса

$$d(x, y) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{s^2}, \quad x, y \subset R^n$$

Здесь s_i - среднеквадратическое отклонение x_i от y_i в обучающей выборке.

Классификаторы

В данной работе будут рассматриваться и сравниваться следующие классификаторы:

1. Наивные алгоритмы
2. Байесовский классификатор
3. Метод опорных векторов
4. Метод k-ближайших соседей

Все эти классификаторы требуют обучения перед их использованием. Для обучения классификаторов был проанализирован сайт theguardian.com и, с помощью простого парсера, были извлечены все предоставляемые этим сайтом криминальные статьи, а также часть статей, которые не являются криминальными (спорт, технологии, жизнь, стиль).

1.4. Методы оценки результата алгоритма

При работе алгоритмов *бинарной* классификации может возникнуть одна из следующих ситуаций:

		Настоящие значения	
		Криминальные	Не криминальные
Предсказанные значения	Криминальные	True positive	False positive
	Не криминальные	False negative	True negative

Введем обозначения, соответствующие всем возможным результатам:

- **tp** - количество true positive результатов при классификации;
- **fp** - количество false positive результатов при классификации;
- **tn** - количество true negative результатов при классификации;
- **fn** - количество false negative результатов при классификации.

На основе этих данных можно рассчитать некоторые общие оценки для классификации, которые не зависят от самого алгоритма.

Точность

Эта оценка показывает, насколько точно происходит классификация, т.е. сколько из предсказанных криминальных документов оказалась действительно крминальными. Чем ближе это значение к 1, тем больше true positive результатов в сравнении с false positive.

$$Precision = \frac{tp}{tp + fp}$$

True positive rate

Показывает, сколько документов из общего числа реальных криминальных документов были отнесены к криминальным. Также TPR часто

называют полнотой. Чем ближе значение TPR к 1, тем больше true positive результатов в сравнении с false negative.

$$TPR = \frac{tp}{tp + fn}$$

False positive rate

Показывает, сколько документов из общего числа реальных некриминальных документов были отнесены к криминальным.

$$FPR = \frac{fp}{fp + tn}$$

F-мера

Одновременно оценивает оба параметра - $Precision$ и TPR

$$F_{measure} = \frac{1}{\alpha * \frac{1}{Precision} + (1 - \alpha) * \frac{1}{TPR}} \qquad \alpha \in (0, 1)$$

С помощью α можно увеличивать или уменьшать значимость параметров. Если $\alpha = 0.5$, то такая F-мера называется сбалансированной.

ROC-кривая

Кривая, позволяющая оценить качество бинарной классификации. Она показывает зависимость TPR от FPR. Если классификатор может оценить вероятность принадлежности объекта к каждому классу, то будем использовать количественную интерпретацию ROC-кривой - AUC. AUC (англ. area under curve) - количественная оценка, которая означает площадь под ROC-кривой.

2. Наивные алгоритмы

Раздел наивных алгоритмов описывается здесь для того, чтобы увидеть, насколько простая реализация, которая почти не использует математический аппарат, классифицирует документы. Так как иначе, может быть и не стоит даже смотреть на более сложные реализации. Также, в конце будет оценена временная сложность каждого алгоритма. Все эти оценки позволят понять, насколько можно использовать наивные алгоритмы в реальной жизни. В этой главе будут последовательно рассмотрены три алгоритма, которые последовательно улучшаются.

Допустим, в нашем распоряжении существует обучающая выборка, состоящая только из криминальных статей. После преобразования этих статей в вектора, отобразим их в пространстве.

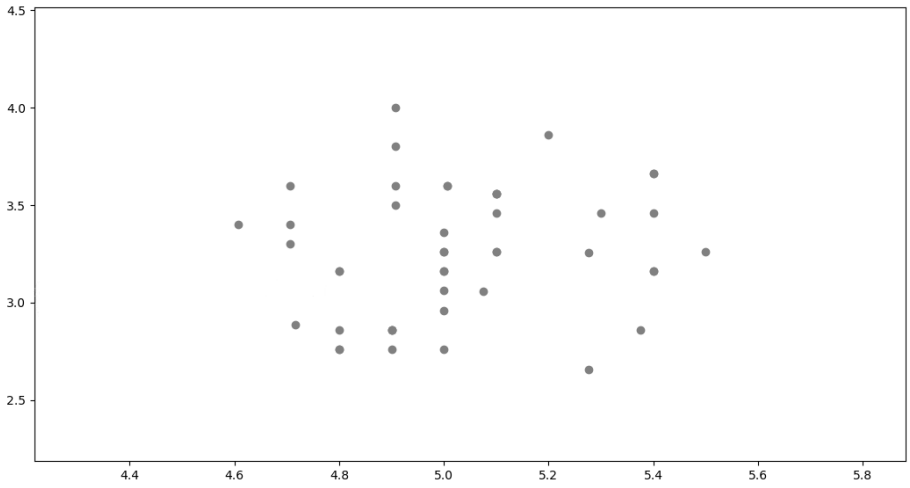

Рис. 3: Документы в двумерном пространстве

Первая мысль, которая приходит, посмотрев на рис.3 - необходимо найти область, которая включает в себя все эти точки. Далее, для того чтобы классифицировать новый документ достаточно посмотреть, попадает ли он в эту область или нет. Если документ попадает в область, то его следует классифицировать как криминальный, иначе - как некриминальный. Для нахождения такой области существует похожая задача в вычислительной геометрии, а именно нахождение выпуклой

оболочки множества.

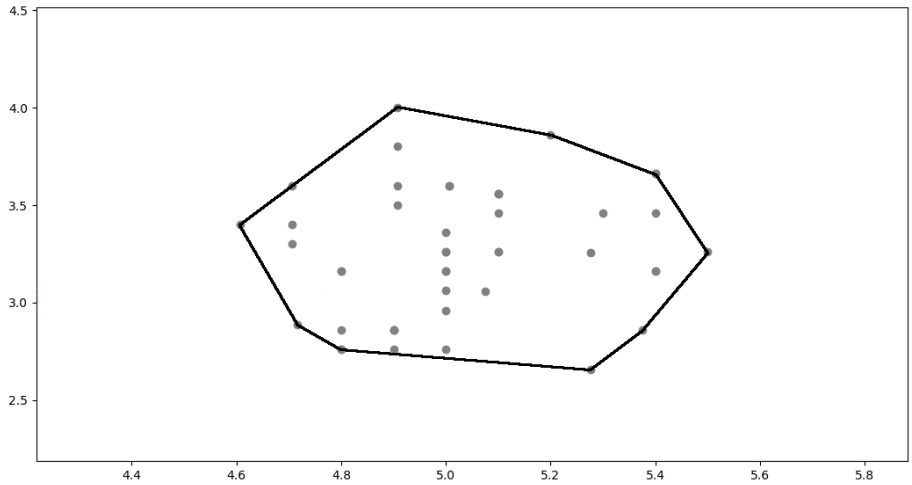

Рис. 4: Выпуклая оболочка

Все известные алгоритмы нахождения оболочки множества очень плохо маштабируются в случае увеличения размерности пространства. А именно, асимптотическая сложность алгоритма Quickhull, который работает для любых размерностей, равна

$$O(\frac{n^{[d/2]}}{[d/2]!}),$$

где n - количество точек множества,
d - размерность пространства.
Таким образом, такие алгоритмы не получится использовать в случае, когда размерность пространства больше 100.

2.1. Шар

Исходя из вышесказанного, можно сделать вывод, что нахождение выпуклой оболочки трудоемко, поэтому рассмотрим следующий способ. Будем искать **выпуклое множество**, содержащее все точки обучающей выборки. Найдем центр масс всех точек из обучающей выборки и будем считать, что вес каждой точки одинаков.

$$M = \sum_{x \in X_l} \frac{x}{n}$$

Теперь нужно ввести расстояние в наше пространство. Выбор расстояния достаточно сильно влияет на точность классификации, что будет показано в дальнейшем. Пока что рассмотрим Евклидово.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad x, y \in R^n$$

После этого, найдем максимально удаленную точку от центра масс. Будем считать это расстояние радиусом шара.

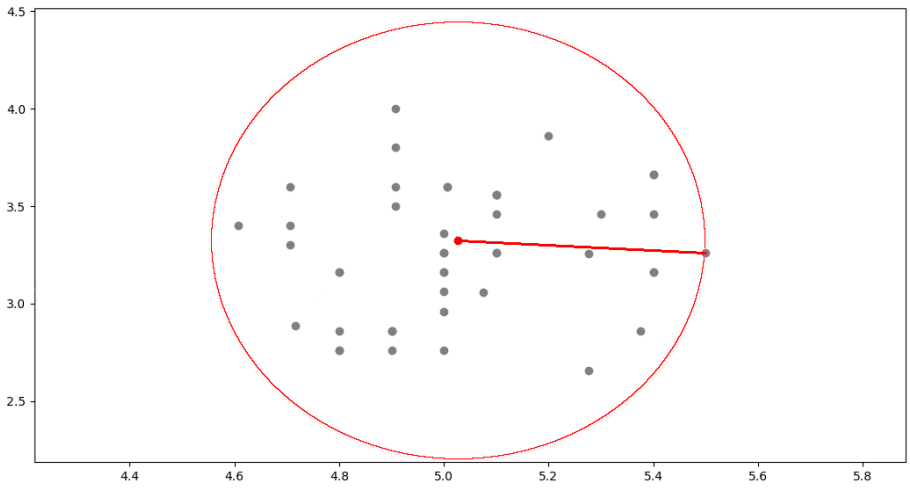

Рис. 5: Шар, как выпуклое множество

Теперь этот шар можно рассматривать как область, внутри которой лежат только точки, которые характеризуют криминальные статьи, а

вне этого шара - все не являющиеся криминальными.

Оценка асимптотической сложности и результатов

Оценим асимптотическую сложность каждого шага алгоритма. Обозначим размерность пространства за n. Размер обучающей выборки обозначим за l. Далее необходимо преобразовать каждый документ из обучающей выборки в вектор из этого пространства. Сложность этого шага варьирутеся от метода преобразования. При этом, этот шаг не относится к самому алгоритму, а лишь подготовка к нему. Нахождение центра масс точек требует, как видно из формулы, $O(l * n)$ операций. Операция подсчета Евклидова расстояния между двумя точками требует $O(n)$ операций. Тогда, чтобы найти максимально удаленную точку от центра масс, потребуется $O(l*n)$ операций. В итоге, временная сложность обучения классификатора будет равна

$$O(l * n)$$

Для самой классификации, очевидно, потребуется лишь находить расстояние от новой точки до центра масс, сложность такого шага равна O(n).

Этот и все последующие алгоритмы реализованы на языке Python. Далее приведены результаты работы этого алгоритма. Число документов для классификации - 10000 (5000 криминальных и 5000 некриминальных документов).

При использовании Евклидова расстояния и модели Bag of words:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 100$	0.5	0.9912	0.9912	0.66469
$\|X_l\| = 1000$	0.50010	1.0	0.9996	0.66675

При использовании косинусного сходства и модели Bag of words:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 100$	0.55087	0.9854	0.8034	0.70668
$\|X_l\| = 1000$	0.5001	1.0	0.9996	0.66675

Плюсы:

- Скорость
- Простота реализации

Минусы:

- Низкая полнота алгоритма
- Нет четкой теоретической обоснованности

Очевидны недостатки этого подхода. Шар покрывает все криминальные статьи, но при этом покрывает еще бесконечное множество точек, которые не являются криминальными. Так происходит по тому, что радиус шара определялся как максимальное расстояние до центра масс. Это ведет к тому, что если существует "аномальная" точка, то шар получается слишком большим. Под "аномальной" точкой понимается такая точка, которая достаточно сильно отклоняется от большинства других точек. Т.е. расстояние от неё до ближайшей точки множества много больше, чем у остальных точек. Данную проблему видно и из результатов. TPR алгоритма близок к 1, т.е. алгоритм хорошо классифицирует криминальные статьи. Но и FPR близок к 1. Это означает что алгоритм и некриминальные статьи определяет как криминальные.

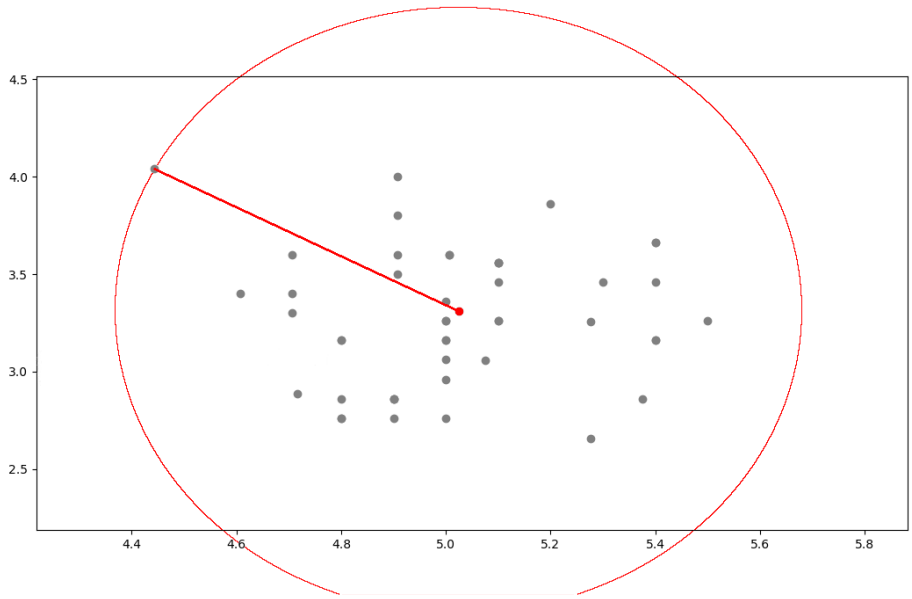

Рис. 6: Шар с аномальной точкой

2.2. Максимальное отклонение по координатам

Как было сказано выше, основная проблема предыдущего подхода была в том, что шар захватывал слишком большую область. Рассмотрим следующий способ. Вместо радиуса шара будем счиать максимальное отклонение по координатам от центра масс. Центр масс находим тем же способом.

$$H = (h_1, ..., h_n)$$
$$h_i = \max_{x \in X_l} |x_i - m_i| \quad i = 1, ..., n$$

Значение h_i в векторе H будет показывать максимальное отклонение по этой координате. После обучения классификатора сформируется вектор H. Рассмотрим процесс классификации нового документа. Напомню, что вектор, который характеризует этот документ, обозначается за $x \in X$.

$$|x_i - m_i| <= h_i, \quad i = 0, ..., n$$

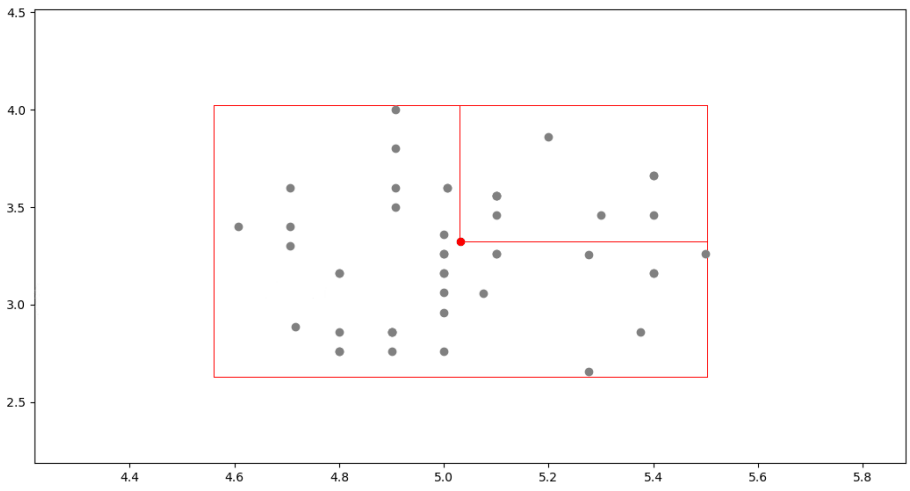

Рис. 7: Максимальные координаты

Как видно, такой подход делает область достаточно обширной и при этом не обобщает одно общее расстояние на все координаты, как было в случае с шаром.

Оценка асимптотической сложности и результатов

Обозначения остаются теми же. Подсчитаем сложность каждого шага алгоритма:

- Нахождение центра масс точек - $O(n * l)$
- Подсчет и максимизация отклонений, т.е. подсчет вектора H - $O(n * l)$

Итоговая временная сложность обучения классификатора будет равна $O(l * w)$. Для классификации нового документа потребуется $O(n)$ операций.

При использовании модели Bag of words:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 100$	0.97560	0.032	0.0008	0.06196
$\|X_l\| = 1000$	0.86471	0.4372	0.0684	0.58076
$\|X_l\| = 5000$	0.74794	0.7994	0.2694	0.77281

Эти результаты показывают, что появилось заметное улучшение. При размере обучающей выборки равной 5000: $tp = 3997$, а $tn = 3653$.

При использовании модели TF-IDF:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 100$	0.72727	0.0016	0.0006	0.00319
$\|X_l\| = 1000$	0.66	0.2112	0.1088	0.32
$\|X_l\| = 5000$	0.55810	0.6666	0.5278	0.60754

Плюсы:

- Скорость
- Простота реализации

Минусы:

- Низкая точность алгоритма

- Также нет четкой теоретической обоснованности

Основным недостатком этого подхода является то, что берется абсолютное максимальное отклонение по координате. Т.е. максимальное отклонение берется в обе стороны по оси от центра масс. Данную проблему хорошо отражает следующее изображение.

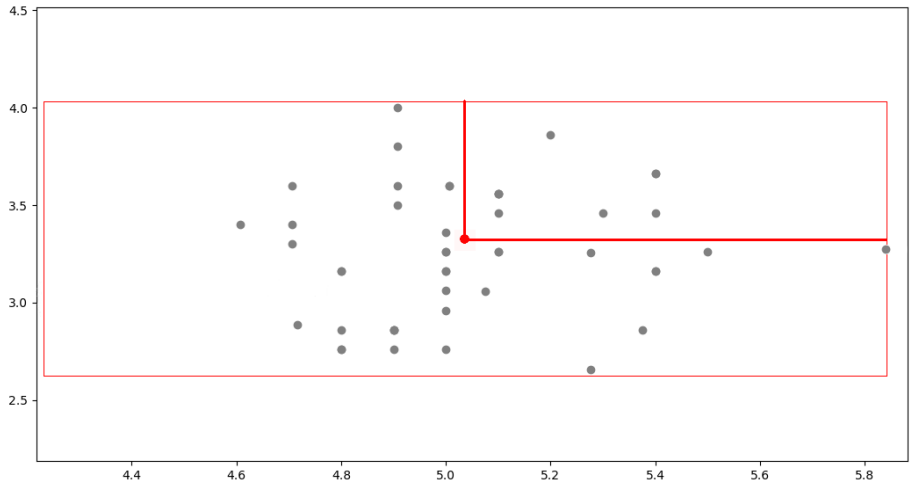

Рис. 8

Существует явный способ исправить эту проблему. Необходимо подсчитывать два возможных отклонения от центра масс в каждую сторону оси. Если $m_i - x_i < 0$, то обновлять вектор $H_l = \{h_1^l, ..., h_n^l\}$, иначе следует обновлять вектор $H_r = \{h_1^r, ..., h_n^r\}$.

$$h_i^l, \ if \quad m_i - x_i < 0$$
$$h_i^r, \ if \quad m_i - x_i >= 0$$
$$h_i^l = h_i^r = \max_{x \in X_l} |m_i - x_i| \quad i = 1, ..., n$$

2.3. Эвристики

Во всех предыдущих шагах было показано, что существование одной "аномальной" точки могло очень сильно испортить оболочку множества. Чтобы исправить эту ситуацию, можно взять p-ый перцентиль от всех расстояний в предыдущих методах. Это не совсем вписывается в формализацию задачи, т.к. теперь, после обучения, алгоритм будет не с абсолютной точностью классифицировать обучающую выборку. Также, стоит учесть, что обучение происходит только на криминальных документах. Можно учитывать слова взятые из некриминальных статей, уменьшая их важность. При расчете центра масс, можно учитывать вес точек, который бы формировался в зависимости от расстояний до других точек, т.е. центр масс будет смещаться к тем точкам, который находятся близко к друг другу. В результате всех этих изменений точность и полнота при классификации новых документов может повыситься.

Итоговый лучший результат после применения всего вышесказанного был достигнут при размерности обучающей выборки в 1000, методом максимального отклонения. Характеристики получились следующими:

$$Precision = 0,80035$$

$$TPR = 0,8202$$

$$FPR = 0,2046$$

$$F - measure = 0,81044$$

2.4. Метод k-ближайших соседей

Метод k-ближайших соседей(англ. k-nearest neighbors algorithm) - метрический алгоритм, предназначенный для классификации.

Метод ближайшего соседа

Для определения класса нового объекта необходимо найти ближайшую к нему точку. Результатом работы алгоритма будет класс этой точки.

Метод k-ближайших соседей

Для повышения точности классификации обычно увеличивают число соседей, по которым определяют класс. Новый документ следует отнести к тому классу, к которому принадлежит большинство из его k соседей. Если рассматривать задачу бинарной классификации, то число соседей берут нечетным, чтобы не возникало неоднозначности, когда одинаковое число соседей принадлежит разным классам.

Метод взвешенных ближайших соседей

В задачах с числом классов более двух, нечётность уже не помогает и всё равно может возникнуть ситуация неоднозначности. Для решения данной проблемы $i-$ соседу приписывается вес w_i. Объект следует отнести к классу, который набирает больший суммарный вес среди k-ближайших соседей.

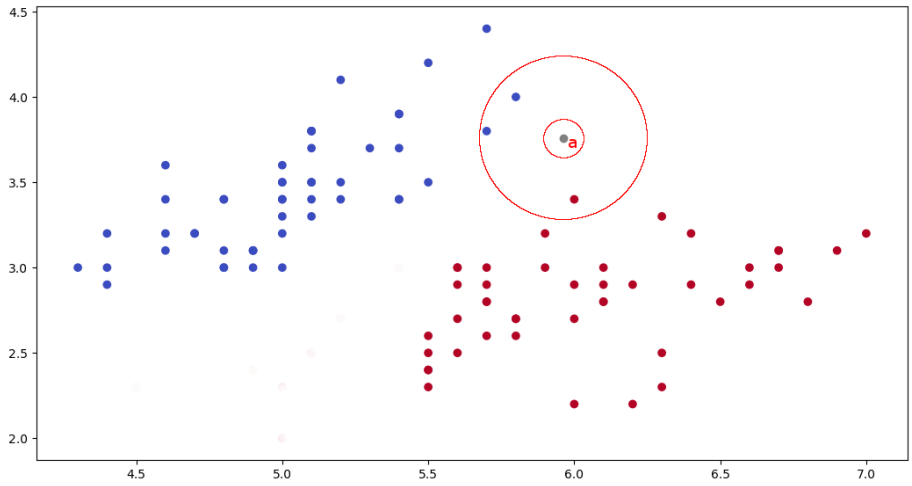

Рис. 9: Описание работы k-ближайших соседей

На рисунке показан алгоритм классификации нового документа a. Сначала отобразим этот документ на плоскости, далее начнем постепенно увеличивать радиус шара, центром которого является a. Как только в шаре будет находится k точек, перестаем расширять его. Теперь документ a нужно отнести к тому классу, к которому принадлежит большинство его соседей. При $k = 3$ точка a будет отнесена к синему классу.

Метод взвешенных ближайших соседей разбираться в данной работе не будет, т.к. данная работа нацелена на бинарную классификацию, поэтому, как говорилось выше, достаточно брать нечетное число соседей.

Оценка асимптотической сложности и результатов

Оценим асимптотическую сложность каждого шага алгоритма:

- Подсчет расстояний от документа до всех точек - $O(n * l)$
- Выбор k-ближайших соседей из расстояний подсчитаных на предыдущем шаге - $O(n*log(n))$ с использованием сортировки слиянием (можно быстрее, за $O(n)$).
- Выбор класса - $O(k)$

Итоговая сложность равна $O(n * l)$. Далее приведены результаты работы алгоритма. Число документов для классификации также равно 10000. Число соседей k равно 5. Половина обучающей выборки состоит из криминальных, другая половина - из некриминальных.

При использовании Евклидова расстояния и модели Bag of words:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.88219	0.7234	0.0966	0.79494
$\|X_l\| = 2000$	0.76571	0.972	0.2974	0.85661
$\|X_l\| = 10000$	0.78732	0.9788	0.2644	0.87268

При использовании расстояния Махаланобиса (при p = 50) и модели Bag of words:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.80360	0.748	0.1828	0.77480
$\|X_l\| = 2000$	0.85066	0.8886	0.156	0.86921
$\|X_l\| = 10000$	0.87390	0.9204	0.1328	0.89655

Значимым минусом этого алгоритма является то, что проблематично выбрать число соседей k.

3. Метод опорных векторов

3.1. Описание

Метод опорных векторов (англ. Support Vector Machine, SVM) - линейный классификатор, основан на разделении множества векторов из n-мерного пространства гиперплоскостью. Этот метод требует обучения перед классификацией.

Определение. Два множества называются линейно разделимыми (или линейно сепарабельными), если существует гиперплоскость, которая отделяет их друг от друга.

Рассмотрим задачу бинарной классификации. Допустим, обучающая выборка является линейно разделимой. Тогда нужно найти такую гиперплоскость, которая бы их разделяла. В общем случае, существует более одной такой гиперплоскости. Возникает вопрос: какую выбрать?

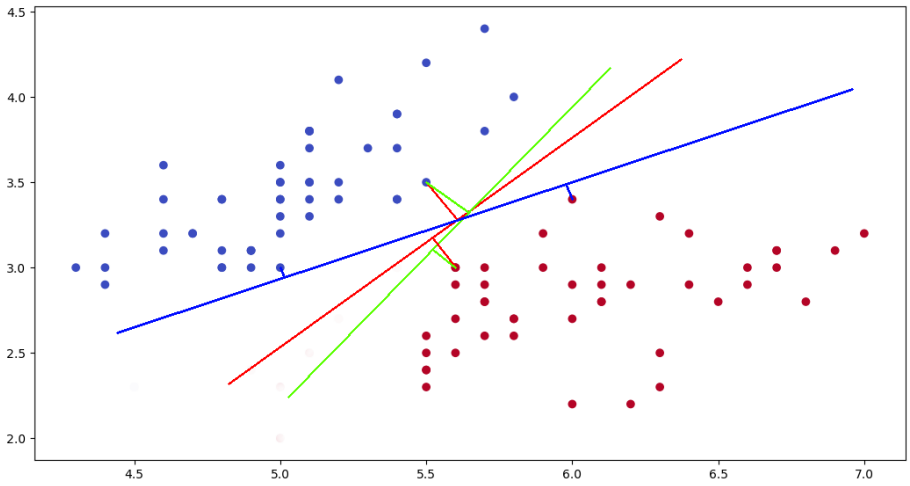

Рис. 10: Линейно разделимое множество

Логично предположить, что следует выбрать ту гиперплоскость, которая максимально удалена от обоих классов. Такую гиперплоскость и будем называть **оптимальной**.

Допущение о линейной разделимости обучающей выборки было довольно сильным. На практике это скорее исключение, чем правило. В

дальнейшем будут рассмотрены методы, которые успешно справляются и в случае линейной неразделимости.

3.2. Формализация

Обозначим множество документов за $X \subset R^n$, его элементы соответсвенно x_i. Будем рассматривать задачу бинарной классификации, поэтому, не уменьшая общности, обозначим $Y = \{-1, 1\}$. Множество $S = \{(x_i, y_i)|x_i \in X, y_i \in Y\}$, i = 1, ..., k.

Определение. *Опорными векторами* будем называть те вектора, расстояние от которых до разделяющей гиперплоскости минимально.

Из аналитической геометрии известно, что гиперплоскость задается следующим уравнением:

$$(c, x) - d = 0$$

Тогда очевидно, что документы одного класса должны удовлетворять $(c, x_i) >= d$, а другого $(c, x_j) <= d$. Зафиксируем гиперплоскость, тогда параллельные гиперплоскости, содержащие опорные вектора, будут выглядеть таким образом:

$$(c, x) = d + \varepsilon$$
$$(c, x) = d - \varepsilon$$

Отклонение ε одно и тоже в силу того, что расстояние до опорных векторов будет одинаковым, иначе такая гиперплоскость точно не будет оптимальной (очевидный факт, но может быть следует пояснить)

Сделаем нормировку, а именно поделим вектор c и число d на ε.

$$(c, x) - d = 1$$
$$(c, x) - d = -1$$

Для двумерного случая это продемонстрировано на рисунке 11:

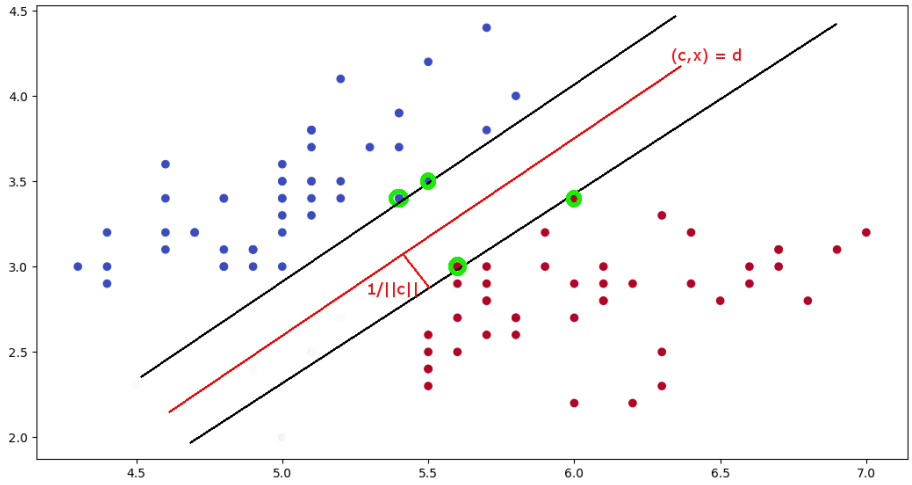

Рис. 11: Опорные вектора и параллельные гиперплоскости

Введем Евклидову метрику в наше пространство. Тогда расстояние между двумя параллельными гиперплоскостями можно вычислить следующим образом:

$$dist = \frac{|d_1 - d_2|}{||c||},$$

Здесь d_1 - свободный член первой гиперплоскости;
d_2 - свободный член второй гиперплоскости;
c - одинаков, вследствие параллельности гиперплоскостей.

Тогда несложно убедиться, что расстояние от параллельных гиперплоскостей до оптимальной гиперплоскости равно $\frac{1}{||c||}$. Теперь, чтобы найти оптимальную гиперплоскость, нужно решить следующую задачу оптимизации:

$$||c|| \to min$$
$$(c, x_i) - d >= 1, y_i = 1$$
$$(c, x_i) - d <= -1, y_i = -1$$

или в более удобной записи

$$||c|| \to min$$
$$y_i * ((c, x_i) - d) >= 1$$

Эта задача сводится к двойственной задаче поиска седловой точки функции Лагранжа. В данной работе решение этой задачи описано не будет, т.к. является частью теории методов оптимизации. Все приведенные выше рассуждения были основаны на линейной разделимости обучающей выборки. Что делать в том случае, если это не так? Позволим алгоритму допускать ошибки, но будем искать такую гиперплоскость, которая бы минимизировала их.

Тем самым рассуждения выше сохраняются, за исключением самой задачи оптимизации, которая изменится следующим образом:

$$||c|| + \sum_{i=0} n\delta_i \to min$$
$$y_i * ((c, x_i) - d) >= 1 - \delta_i$$
$$\delta_i >= 0, i = 0, ..., n$$

Здесь, δ_i равно нулю в том случае, если соотвествующее неравенство выполняется и без него. Т.е. δ_i является минимально возможным значением, при котором неравенство будет выполнено. Решая эти задачи, можно получить вектор с, который и нужен для построения гиперплоскости.

3.3. Ядра

Если обучающая выборка почти линейно разделима, т.е. существует достаточно мало "аномальных" точек, то предыдущий метод с введением δ_i вполне хорошо себя показывает. Остается вопрос - как быть в ситуации, когда классы не разделяются линейно?

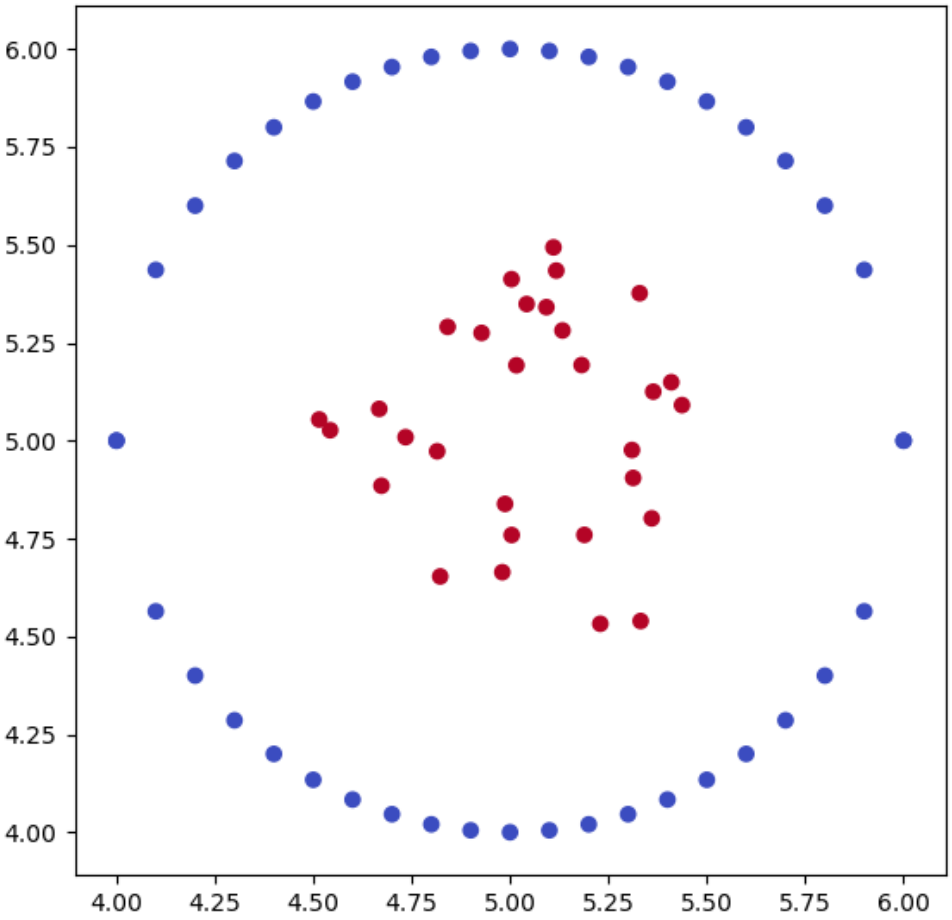

Рис. 12: Один класс в окружении другого

Теорема (Ковера)**.** *Нелинейное преобразование сложной задачи классификации образов в пространство более высокой размерности повышает вероятность линейной разделимости образов.*

В соответствии с теоремой Ковера, при повышении размерности пространства повышается и вероятность того, что эти два множества

будут линейно разделимыми. Когда выборку нужно разделить нелинейно, применяется так называемый kernel trick - переход от скалярного произведения к произвольным ядрам. По сути, это переход от исходного пространства признаков к X к новому пространству L с помощью некоторого отображения $h : X \to L$. Теперь, если допустить, что признаковым описанием является $h(x_i)$, то построение гиперплоскости производится точно также, за исключением того, что скалярное произведение (x_i, x_j) заменятся на $(h(x_i), h(x_j))$.

Определение. Функция $K : X * X \to R$ называется *ядром*, если она представима в виде $K(x_1, x_2) = (h(x_1), h(x_2))$, при некотором отображении $h : X \to L$, где L — пространство со скалярным произведением.

Если посмотреть на постановку задачи то можно заметить, что вся теория основана лишь на скалярном произдведении векторов и не зависит напрямую от самих признаков. Из определения очевидно, что любое скалярное произведение является ядром. Таким образом, можно заменить скалярное произведение на ядро. В общем случае ядро нелинейно.

Возникает следующий вопрос: какая функция может быть ядром?

Теорема. *Для того чтобы функция $K(x_1, x_2)$, являлась ядром, необходимо и достаточно, чтобы она была неотрицательно определена и симметрична.*

Рассмотрим несколько распространенных ядер:

- **Линейное:** $K(x_1, x_2) = (x_1, x_2)$
- **Полиномиальное:** $K(x_1, x_2) = ((x_1, x_2) + const)^d$
- **Сигмоида:** $K(x_1, x_2) = \tanh(k*(x_1, x_2)+const)$, где $const < 0, k > 0$
- **Радиальная базисная функция (RBF):** $\exp(-\gamma * ||x_1 - x_2||^2)$, где $\gamma > 0$

Сама идея ядра достаточно красиво и продуктивно вписывается в SVM. Понятно, что после перехода в новое прострнство и нахождения

там гиперплоскости, приведет к тому, что в изначальном пространстве получившаяся кривая, в общем случае, не будет линейной. Если рассмотреть полиномиальное ядро, то можно увидеть, что оно реализует идею полиномиальных разделяющих поверхностей. Концепция ядра достаточно универсальна, поэтому SVM может покрыть большой спектр задач по классификации.

При использовании полиномиального ядра с $d = 3$, точки на рисунке 12 можно классифицировать так:

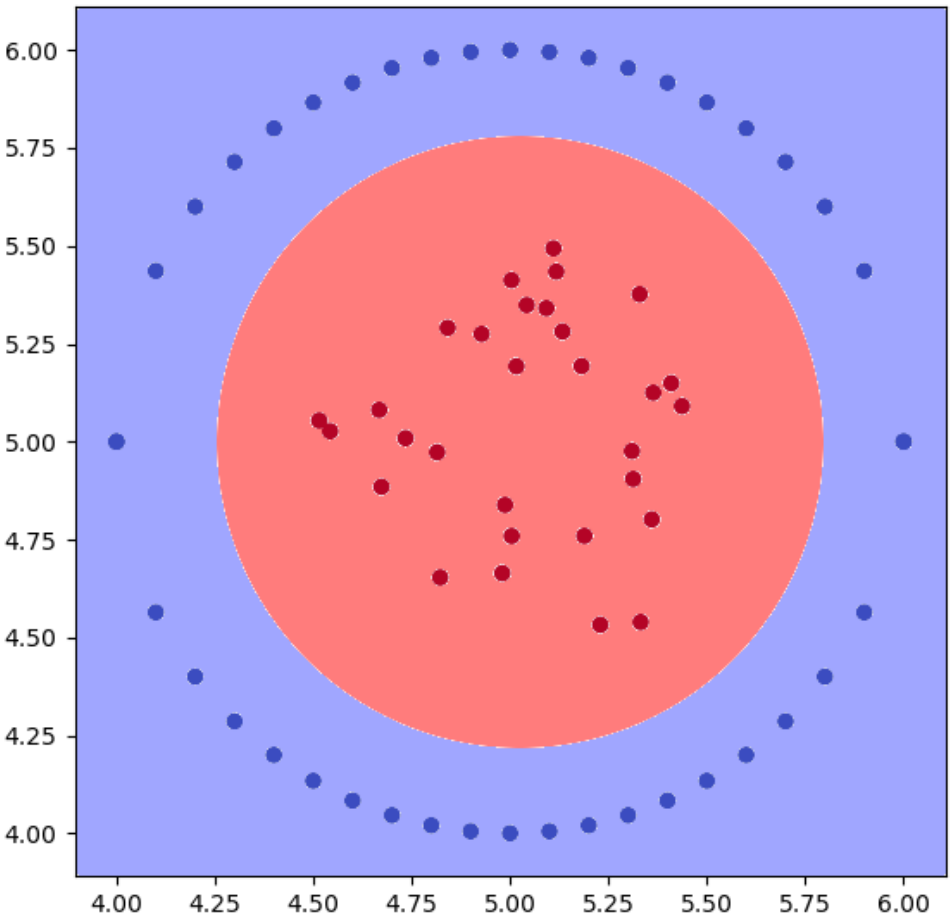

Рис. 13: Разделение в новом пространстве

3.4. Оценка результатов

Алгоритм был реализован на языке Python. Далее приведены результаты работы этого алгоритма с использованием разных ядер и размеров обучающей выборки. Количество документов для классификации - 10000. Обучающая выборка состоит наполовину из криминальных статей.

При использовании линейного ядра:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.95618	0.9472	0.0434	0.95167
$\|X_l\| = 2000$	0.97606	0.9542	0.0234	0.965008
$\|X_l\| = 10000$	0.97363	0.9676	0.0262	0.970608

При использовании RBF ядра:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.94972	0.9218	0.0488	0.93555
$\|X_l\| = 2000$	0.97737	0.9416	0.0218	0.95915
$\|X_l\| = 10000$	0.98104	0.9728	0.0188	0.97690

При использовании сигмоидального ядра:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.93195	0.9368	0.0684	0.93437
$\|X_l\| = 2000$	0.97767	0.937	0.0214	0.95690
$\|X_l\| = 10000$	0.96835	0.9608	0.0314	0.96456

4. Наивный байесовский классификатор

4.1. Описание

Наивный байесовский классификатор - вероятностный классификатор, основанный на теореме Байеса. Он называется наивным, потому как считает, что наличие одного из признаков никак не зависит от наличия другого. С первого взгляда, такое допущение может показаться слишком сильным, но на практике этот алгоритм показывает достаточно хорошие результаты.

Теорема Байеса говорит о том, как рассчитать апостериорную вероятность:

$$P(c_i|x) = \frac{P(x|c_i) * P(c_i)}{P(x)}$$

Здесь $P(c_i|x)$ - вероятность того, что вектор $x \in X$ относится к классу $c_i \in C$;
$P(x|c_i)$ - вероятность того, что x примет данное значение, при условии, что выбран класс c_i;
$P(c_i)$ - априорная вероятность данного класса;
$P(x)$ - априорная вероятность данного вектора.

4.2. Формализация

Для того, чтобы понять, какой у вектора x класс, найдем такой класс, который бы максимизировал $P(c_i|x)$.

$$y_i = \arg\max_{c_i \in C} P(c_i|x) = \arg\max_{c_i \in C} \frac{P(x|c_i) * P(c_i)}{P(x)}$$

Можно заметить, что для решения задачи нахождения аргумента, при котором достигается максимум, не нужно учитывать знаменатель, т.к. он не зависит от c_i.

$$y_i = \arg\max_{c_i \in C} P(x|c_i) * P(c_i)$$

Далее, самым инересным в этой формуле является $P(x|c_i)$, распишем его:

$$P(x|c_i) = P(x_1, x_2, ..., x_n|c_i) = P(x_1|c_i) * P(x_2, x_3, ..., x_n|c_i, x_1) = \\ = P(x_1|c_i) * P(x_2|c_i, x_1) * P(x_3, ...x_n|c_i, x_1, x_2)$$

Продолжая эту цепочку равенств, можно прийти к следующему

$$P(x|c_i) = P(x_1|c_i) * P(x_2|c_i, x_1) * ... * P(x_n|c_i, x_1, x_2, ..., x_{n-1})$$

Эту вероятность достаточно сложно посчитать, т.к. в большинстве случаев непонятно, как именно появление одного признака зависит от появления другого, т.е. $P(x_n|c_i, x_1, x_2, ..., x_n - 1)$.

Далее, воспользуемся предположением о независимости признаков. Именно в этот момент алгоритм становится "наивным".

$$P(x_2|c_i, x_1) = P(x_2|c_i)$$

$$...$$

$$P(x_n|c_i, x_1, ..., x_{n-1}) = P(x_n|c_i)$$

Тогда, если учесть данное предположение, задача приводится к следующему виду

$$y_i = \arg\max_{c_i \in C} P(c_i) * P(x_1|c_i) * P(x_2|c_i) * ... * P(x_n|c_i)$$

Зная $P(c_i)$ и условные вероятности $P(x_j|c_i)$, можно найти вероятность принадлежности документа x к каждому классу c_i. Далее, тот класс, на котором достигается максимальная вероятность и будет считаться результатом работы алгоритма.

4.3. Применение к задаче

Рассмотрим применение этого алгоритма к нашей задаче. В нашей задаче существует всего два класса документов $C = \{-1, 1\}$. Множество документов все также обозначим за $X \subset R^n$.

Как говорилось выше, для решения задачи нужно понять, чему равно $P(c_i)$. Если считать, что обучающая выборка сформирована достаточно точно, т.е. описывает реальное распределение документов по классам, то $P(c_i)$ можно посчитать так

$$P(c_i) = \frac{D_{c_i}}{D}, \quad i = 1, 2$$

D_{c_i} обозначает количество документов класса c_i во всем множестве документов, D - общее количество документов. Оценка принадлежности документа к классу $P(x_j|c_i)$ может вычисляться несколькими способами, рассмотрим последовательно каждый из них.

Мультиномиальное (полиномиальное) распределение

Используется в случае дискретных признаков, например, если документы были переведены в слова с помощью Bag of words. Далее все формулы написаны в предположении о том, что документы были переведены в n-мерное пространство с помощью Bag of words. Также зафиксируем класс, это позволит уменьшить количество индексов в дальнейшем и при этом никак не уменьшит общности рассуждений.

$$P(x|c) = (\sum_{i=1}^{n} x_i)! \prod_{i=1}^{n} \frac{P(w_i|c)^{x_i}}{x_i!}$$

w_i - то слово, которое соответствует x_i.

Для обучения такого классификатора потребуется знание $P(w_i|c)$.

$$P(w_i, c) = \frac{\sum_{x \in X_1} x_i}{\sum_{j=1}^{n} \sum_{x \in X_1} x_j}$$

$X_1 \subset X$ - множество тех документов, которые принадлежат классу c. В числителе находится число вхождений слова w_i в документы, которые относятся к классу c. В знаменателе же описано количество всех слов в

документах, относящихся к классу c. Теперь допустим, что после обучения сформировались значения $P(w_i, c)$. Если в момент классификации встретится слово, которое не учавствовало в обучающей выборке, то $P(w_new, c)$ будет равна 0. В итоге, весь документ нельзя будет классифицировать, т.к. P(x|c) для любого класса с будет равна нулю. Для решения этой проблемы часто применятся *сглаживание Лапласа*. Будем прибавлять 1 к частоте каждого слова.

$$P(w_i, c) = \frac{1 + \sum_{x \in X_1} x_i}{n + \sum_{j=1}^{n} \sum_{x \in X_1} x_j}$$

Распределение Бернулли

Используется в случае дискретных признаков, которые могут принимать значения 0 или 1. Опять же, модель Bag of words может соответствовать этому предположению.

$$P(x|c) = \prod_{i=1}^{n} P(w_i|c)^{x_i} * (1 - P(w_i|c))^{1-x_i}$$

Для нахождения $P(w_i, c)$ можно использовать формулы, приведенные выше, за исключением того, что количественный признак сменяется на бинарный, т.е. наличие или отсутвие слова в документе.

4.4. Оценка результатов

Алгоритм был реализован на языке Python. Далее приведены результаты работы этого алгоритма с использованием разных вариантов распределения и размеров обучающей выборки. Количество документов для классификации - 10000.

При гипотезе о мультиномиальном распределении:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.95077	0.9734	0.0504	0.96195
$\|X_l\| = 2000$	0.97437	0.9658	0.0254	0.97006
$\|X_l\| = 10000$	0.97670	0.9728	0.0232	0.97474

При гипотезе о распределении Бернулли:

	Precision	TPR	FPR	F-мера
$\|X_l\| = 200$	0.83677	0.9556	0.1864	0.89225
$\|X_l\| = 2000$	0.92024	0.9484	0.0822	0.93410
$\|X_l\| = 10000$	0.91436	0.9396	0.088	0.92681

Заключение

Как видно из результатов тестирования, ни один из наивных алгоритмов не показал результатов, которые были бы близки к SVM или наивному байесу. В итоге, наилучшим образом себя показали следующие алгоритмы:

- Наивный байесовский классификатор с мультиномиальным распределением.
- Метод опорных векторов с RBF ядром.

В связи с тем, что в приоритете скорость классификации, то был выбран наивный байесовский классификатор, который удовлетворяет всем требованиям заказчика.

Список литературы

[1] C. Brandford Barber David P. Dobkin Hannu Huhdanpaa. The Quickhull Algorithm for Convex Hulls.

[2] Kozma László. k Nearest Neighbors algorithm (kNN).

[3] The Porter stemming algorithm. — 2017. — URL: `http://snowball.tartarus.org/algorithms/porter/stemmer.html`.

[4] Shimodaira Hiroshi. Text Classification using Naive Bayes.

[5] Воронцов К. В. Лекции по методу опорных векторов.

[6] Воронцов К. В. Лекции по статистическим алгоритмам классификации.

[7] Задачи классификации // Профессиональный информационно-аналитический ресурс, посвященный машинному обучению, распознаванию образов и интеллектуальному анализу данных. — 2017. — URL: `http://www.machinelearning.ru`.