Построение и отбор признаков

Известный принцип garbage in – garbage out на 100% применим к любой задаче машинного обучения; любой опытный аналитик может вспомнить примеры из практики, когда простая модель, обученная на качественно подготовленных данных, показала себя лучше хитроумного ансамбля, построенного на недостаточно чистых данных.

Извлечение признаков (Feature Extraction)

## 1. Тексты

1.1 Самой простой моделью является Bag of Words: создаем вектор длиной в словарь, для каждого слова считаем количество вхождений в текст и подставляем это число на соответствующую позицию в векторе.

Используя алгоритмы вроде Вag of Words, мы теряем порядок слов в тексте, а значит, тексты "i have no cows" и "no, i have cows" будут идентичными после векторизации, хотя и противоположными семантически. Чтобы избежать этой проблемы, можно сделать шаг назад и изменить подход к токенизации: например, использовать N-граммы (комбинации из N последовательных терминов).

In : from sklearn.feature\_extraction.text import CountVectorizer

In : vect = CountVectorizer(ngram\_range=(1,1))

In : vect.fit\_transform(['no i have cows', 'i have no cows']).toarray()

Out: array([[1, 1, 1], [1, 1, 1]], dtype=int64)

In : vect.vocabulary\_

Out: {'cows': 0, 'have': 1, 'no': 2}

In : vect = CountVectorizer(ngram\_range=(1,2))

In : vect.fit\_transform(['no i have cows', 'i have no cows']).toarray()

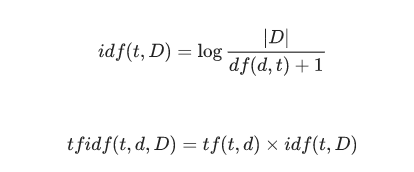
Out: array([[1, 1, 1, 0, 1, 0, 1], [1, 1, 0, 1, 1, 1, 0]], dtype=int64)

In : vect.vocabulary\_

Out: {'cows': 0, 'have': 1, 'have cows': 2, 'have no': 3, 'no': 4, 'no cows': 5, 'no have': 6}

1.2 TF-IDF

Развитие идеи Bag of Words: слова, которые редко встречаются в корпусе (во всех рассматриваемых документах этого набора данных), но присутствуют в этом конкретном документе, могут оказаться более важными. Тогда имеет смысл повысить вес более узкотематическим словам, чтобы отделить их от общетематических. Этот подход называется TF-IDF, его уже не напишешь в десять строк, потому желающие могут ознакомиться с деталями во внешних источниках вроде [wiki](https://en.wikipedia.org/wiki/Tf%E2%80%93idf). Вариант по умолчанию выглядит так:



Выбор признаков (Feature selection)

## 1. Статистические подходы

Отсеиваем те, у которых дисперсия ниже определенной границы, то есть они слабо меняются.

from sklearn.feature\_selection import VarianceThreshold

from sklearn.datasets import make\_classification

x\_data\_generated, y\_data\_generated = make\_classification()

In : x\_data\_generated.shape

Out: (100, 20)

In : VarianceThreshold(.7).fit\_transform(x\_data\_generated).shape

Out: (100, 19)

## 2. Отбор с использованием моделей

Другой подход: использовать какую-то baseline модель для оценки признаков, при этом модель должна явно показывать важность использованных признаков. Обычно используются два типа моделей: какая-нибудь "деревянная" композиция (например, Random Forest) или линейная модель с Lasso регуляризацией, склонной обнулять веса слабых признаков. Логика интутивно понятна: если признаки явно бесполезны в простой модели, то не надо тянуть их и в более сложную.

## 3. Перебор

Наконец, самый надежный, но и самый вычислительно сложный способ основан на банальном переборе: обучаем модель на подмножестве "фичей", запоминаем результат, повторяем для разных подмножеств, сравниваем качество моделей. Такой подход называется [Exhaustive Feature Selection](http://rasbt.github.io/mlxtend/user_guide/feature_selection/ExhaustiveFeatureSelector/).

Перебирать все комбинации – обычно слишком долго, так что можно пробовать уменьшить пространство перебора. Фиксируем небольшое число N, перебираем все комбинации по N признаков, выбираем лучшую комбинацию, потом перебираем комбинации из N+1 признаков так, что предыдущая лучшая комбинация признаков зафиксирована, а перебирается только новый признак. Таким образом можно перебирать, пока не упремся в максимально допустимое число признаков или пока качество модели не перестанет значимо расти. Этот алгоритм называется [Sequential Feature Selection](http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/).

Этот же алгоритм можно развернуть: начинать с полного пространства признаков и выкидывать признаки по одному, пока это не портит качество модели или пока не достигнуто желаемое число признаков.