

Машинное обучение: работа с текстовыми данными



Виды задач

Какие виды задач с текстами можно решать с помощью машинного обучения?

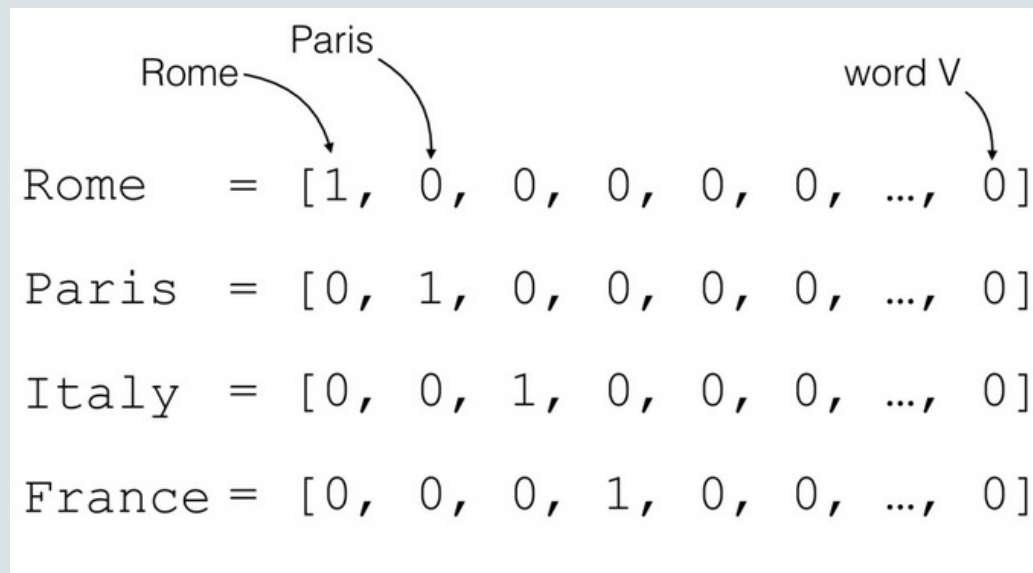
- классификация текстов (sentiment analysis...)
- классификация токенов (PoS-tagging, NER...)
- кластеризация текстов (topic modelling...)
- генерация текстов (классическое МО – цепи Маркова)

Где у текстов признаки?

- Можем считать, что признаки в текстовых данных – это сами слова.
- Но нужно превратить их в числа
- **Самый простой способ это сделать?**

Где у текстов признаки?

- Можем считать, что признаки в текстовых данных – это сами слова. Или буквы.
- Но нужно превратить их в числа
- Самый простой способ это сделать – **One Hot Encoding**



	Rome	Paris						word V	
Rome	=	[1,	0,	0,	0,	0,	0,	...,	0]
Paris	=	[0,	1,	0,	0,	0,	0,	...,	0]
Italy	=	[0,	0,	1,	0,	0,	0,	...,	0]
France	=	[0,	0,	0,	1,	0,	0,	...,	0]

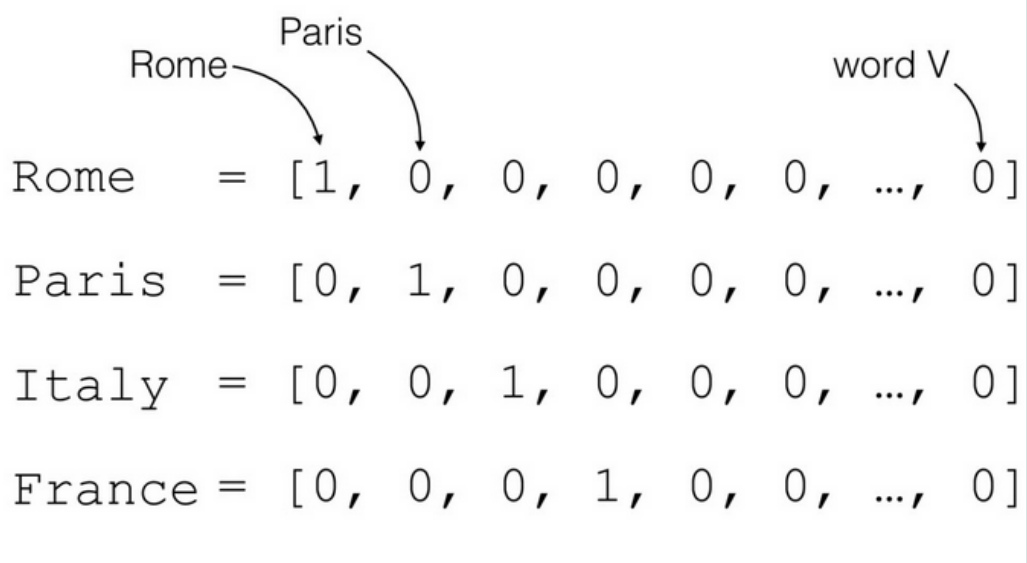
Где у текстов признаки?

- Можем считать, что признаки в текстовых данных – это сами слова. Или буквы.
- Но нужно превратить их в числа
- **Самый простой способ это сделать – One Hot Encoding**
- Какие недостатки?

	Rome	Paris						word V	
Rome	=	[1,	0,	0,	0,	0,	0,	...,	0]
Paris	=	[0,	1,	0,	0,	0,	0,	...,	0]
Italy	=	[0,	0,	1,	0,	0,	0,	...,	0]
France	=	[0,	0,	0,	1,	0,	0,	...,	0]

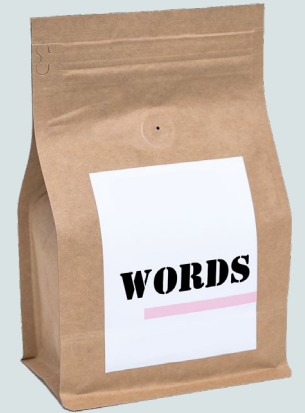
Где у текстов признаки?

- Можем считать, что признаки в текстовых данных – это сами слова. Или буквы.
- Но нужно превратить их в числа
- Самый простой способ это сделать – **One Hot Encoding**
- Это очень неэффективно
и не интерпретируемо
- С другой стороны, это единственный способ передать в числах именно сами слова
- А как текст передать в виде вектора?



Rome	=	[1, 0, 0, 0, 0, 0, ..., 0]
Paris	=	[0, 1, 0, 0, 0, 0, ..., 0]
Italy	=	[0, 0, 1, 0, 0, 0, ..., 0]
France	=	[0, 0, 0, 1, 0, 0, ..., 0]

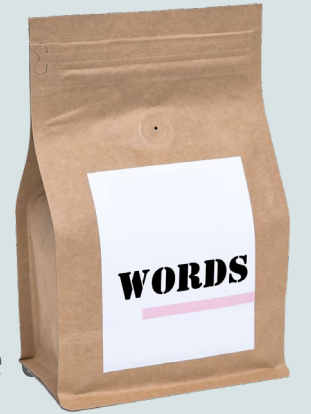
Bag of Words



- Составляем словарь для всех наших текстов, сортируем и нумеруем слова
- Каждый текст – это вектор такой же длины, какой у нас словарь
- Вписываем частоту слова под его порядковым номером
- Получаем эмбединги предложений

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Bag of Words



- Это уже неплохая идея: мы получаем представление как минимум о частоте
поэтому если в нашем тексте 10 раз встретилось «купи», вероятно, это спам.
- В `sk-learn` эта идея реализована в инструменте для предобработки признаков
`CountVectorizer()`.
- Можно эту идею улучшить!
- Почему бы вместо слов не использовать n -граммы?
- n -граммы позволят нам учитывать еще и контекст
- но если n слишком большое, у нас будут опять разреженные вектора

TF-IDF

- Давайте еще усовершенствуем нашу модель.
- Как вам частоты словечек типа «и», «или», «в»?

TF-IDF

- Давайте еще усовершенствуем нашу модель.
- Наверное, хочется как-то учитывать еще и «важность» слов: чисто интуитивно слово «котик» важнее, чем слово «и».
- Как это сделать?

TF-IDF

- Давайте еще усовершенствуем нашу модель.
- Наверное, хочется как-то учитывать еще и «важность» слов: чисто интуитивно слово «котик» важнее, чем слово «и».
- Давайте вместо частоты будем записывать term frequency-inverse document frequency.
- Term frequency (частота слова) – отношение частоты слова к общему числу всех слов
- IDF (обратная частота документа) – инверсия частоты, с которой слово встречается в разных документах

TF-IDF

- TF: $\frac{\text{частота слова}}{\text{все слова в корпусе}}$
- IDF: $\log \frac{\text{количество документов в корпусе}}{\text{количество документов, в которых встречается наше слово}}$
- Пример:

у нас в корпусе 3 документа, в которых в сумме 100 слов.

слово «котик» встречается 30 раз, но в двух документах.

$$\text{его TF-IDF} = \frac{30}{100} \cdot \log \frac{3}{2} = 0.12$$

а слово «карбюратор» встречается 15 раз, но только в одном документе.

$$\text{его TF-IDF} = \frac{15}{100} \cdot \log \frac{3}{1} = 0.16$$

TF-IDF

- Есть, однако, одна засада...
- Посчитайте TF-IDF для слова «и», которое в нашем корпусе из 3 документов и 100 слов встретилось 40 раз и во всех трех документах.

TF-IDF

- Получается, что TF-IDF для такого слова будет равно 0
- Но и для слова, которого в наших документах вообще нет, тоже 0
- Поэтому обычно используют *сглаживание* и таким словам приписывают какое-нибудь очень маленькое значение.
- TF-IDF реализован в `sk-learn` как `TfidfVectorizer()`.

Предварительные ИТОГИ:

- ONE слишком плох (но дает эмбединги слов)
- BoW и TF-IDF дают нам эмбединги предложений, а не слов
- А если нам все-таки нужны более качественные эмбединги самих слов?

Эмбединги по контексту

- Если считать эмбединг слова по его самым частотным контекстам? Например, такой алгоритм:
 - 1) Берем большой набор текстов, составляем словарь из уникальных слов длиной V
 - 2) Выбираем ширину контекстного окна
 - 3) Проходимся по всему корпусу и вписываем в вектор длины V частоты слов, попавших в контекстное окно для нашего слова
- Даже это уже работает!

Эмбединги по контексту

- Вот так выглядят самые похожие слова на эмбедингах, обученных на «Войне и мире»
- «Похожие» слова – слова, у которых маленькое косинусное расстояние между их векторами

Give a word to look up or enter "stop"
сказать
говорить
видеть
думать
спрашивать
любить
подумать
просить
продолжать
прибавлять
бояться
Give a word to look up or enter "stop"
утро
день
вечер
еще
уже
час
ночь
пьер
быть
ростов
дом

Give a word to look up or enter "stop"
петербург
москва
дом
уже
быть
приезжать
еще
человек
принимать
казаться
пьер
Give a word to look up or enter "stop"
мать
отец
наташа
брат
борис
сын
соня
теперь
все
быть
анатоль

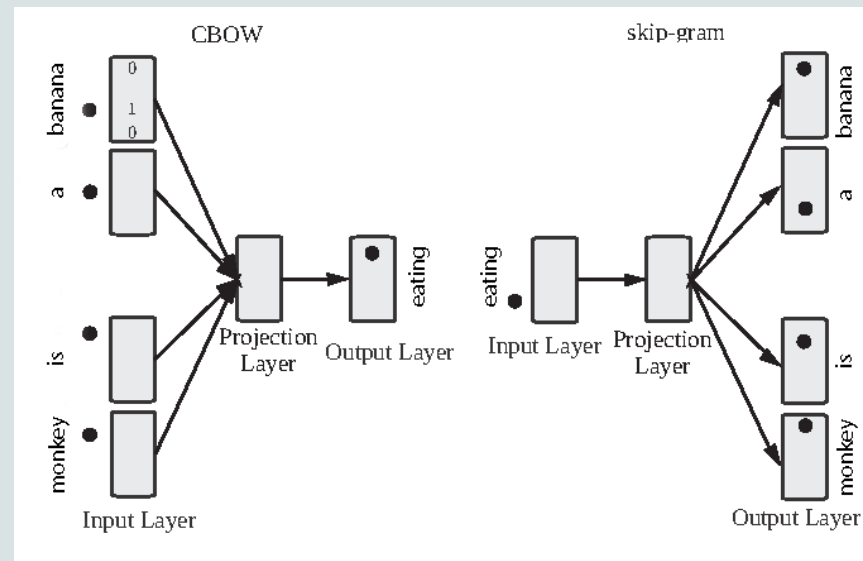
Эмбединги по контексту

- Видим, что хотя «корпус» был очень мал (всего одна, хотя большая книга), даже на этом корпусе можно установить, что слова «утро», «день» и «вечер» похожи, причем чуть более похожи, чем «ночь»
- Следовательно, мы можем судить о семантике таких слов по **косинусной близости их эмбедингов**
- Очень похожая идея (но более сложная) лежит в основе статических эмбедингов

```
Give a word to look up or enter "stop"
утро
день
вечер
еще
уже
час
ночь
пьер
быть
ростов
дом
```

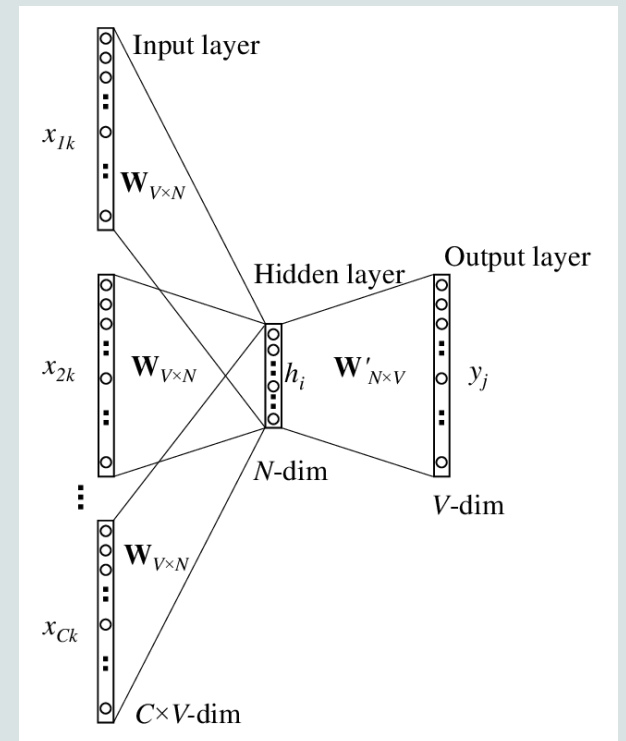
Word2Vec

- Эмбеддинги по контексту на максималках: собираем эмбеддинги таким образом, чтобы максимально хорошо решалась одна из двух задач:
 1. Предсказание контекста по слову (skip-gram)
 2. Предсказание слова по контексту (continuous bag of words)



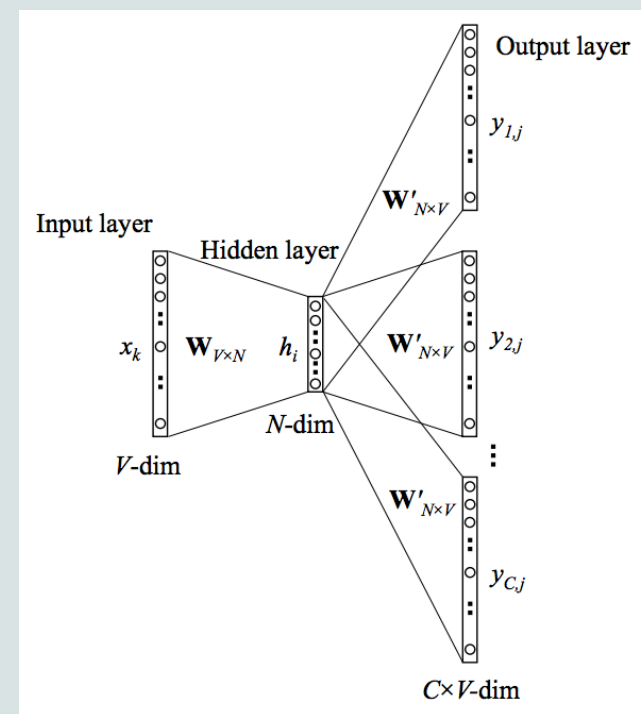
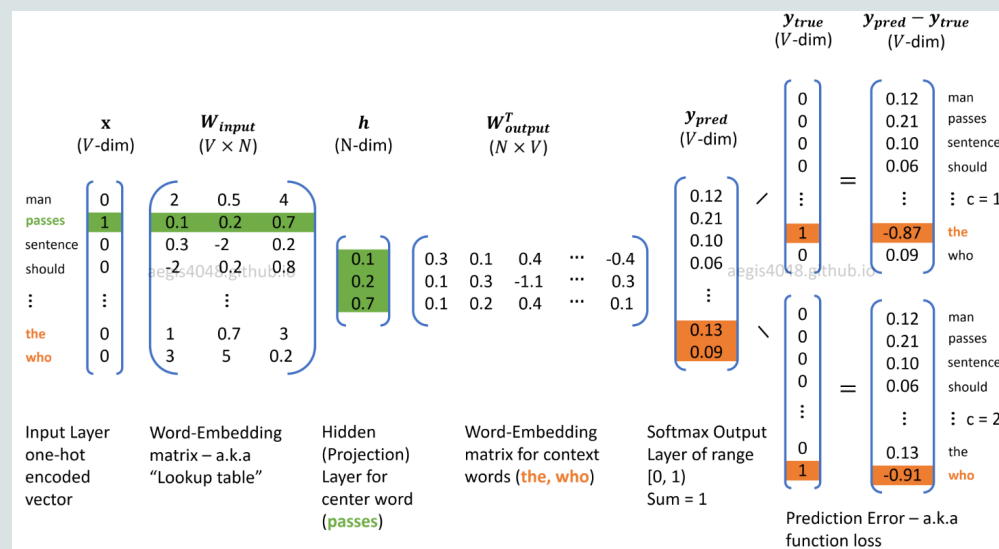
Word2Vec. CBoW

- Модель пытается предсказать следующее слово по предыдущим.
- Мы выбираем окно контекста: наш контекст будет подаваться сетке на вход (можно слова закодировать просто ONE)
- Сетка должна предсказать слово, которое подходит в контексте
- Здесь уже работают **нейронные сети**



Word2Vec. Skip-gram

- Делает ровно наоборот: на входе у нас одно слово (в виде вектора), а на выходе N слов, которые лучше всего подходят для контекста
- Подробнее поговорим в курсе по нейронкам
- Еще пугающих картинок
- [Статья Миколова](#)
- Еще [статья](#) про skip-gram



Word2Vec, GloVe, fasttext

- Все вышеперечисленное – разновидности статических эмбеддингов
- Они уже предобученные: можно взять готовые вектора и подключить их к своей модели в качестве признаков
- Даже контекстуальные эмбеддинги BERT можно подключить к банальной логистической регрессии!



Подводя итог:

- В sklearn реализованы только ONE, BoW, TF-IDF
- Word2Vec – статические эмбединги на нейронках, имплементация есть в библиотеке gensim