Дополнительная обработка. Кластеризация

Сегодня

1. Дополнительная обработка слов
2. Кластеризация

Предобработка данных

Выполнили предобработку текстов.

```
import regex as re

#удаление HTML-тегов, пунктуации, декодирование (чистка)
def clean_content(content):
    #приведение к нижнему регистру
    content = content.apply(lambda x: x.lower())
    #удаление HTML-тегов
    content = content.apply(lambda x: re.sub(r'\<[^<>]*\>','',x))
    #удаление всех символов кроме букв, цифр и подчеркивания
    content = content.apply(lambda x: re.sub(r'^\W+|\W+$',' ',x))
    #удаление пробелов, переводов строк и табов
    content = content.apply(lambda x: re.sub(r'\s',' ',x))
    #удаление знаков препинания
    content = content.apply(lambda x: re.sub(r'[^a-zA-Z0-9]',' ',x))
    return(content)
```

Предобработка данных

Удалили стоп-слова и провели токенизацию.

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
#from nltk.stem import WordNetLemmatizer

stops = set(stopwords.words("english"))
```

```
def clean_stopwords_tokenize(content):
    #Токенизация
    content = content.apply(lambda x: word_tokenize(x))
    #удаление стоп-слов
    content = content.apply(lambda x: [i for i in x if i not in stops])
    return(content)
```

Стемминг

Замена слов на основу слова.

```
import nltk.stem

s = nltk.stem.SnowballStemmer('english')
print([s.stem(w) for w in ['imagine', 'image', 'imagination']])
```

```
['imagin', 'imag', 'imagin']
```

Лемматизация

Приведение слов к начальной форме.

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('worse', pos=wordnet.ADJ))
print(lemmatizer.lemmatize('better', pos=wordnet.ADJ))
```

```
bad
good
```

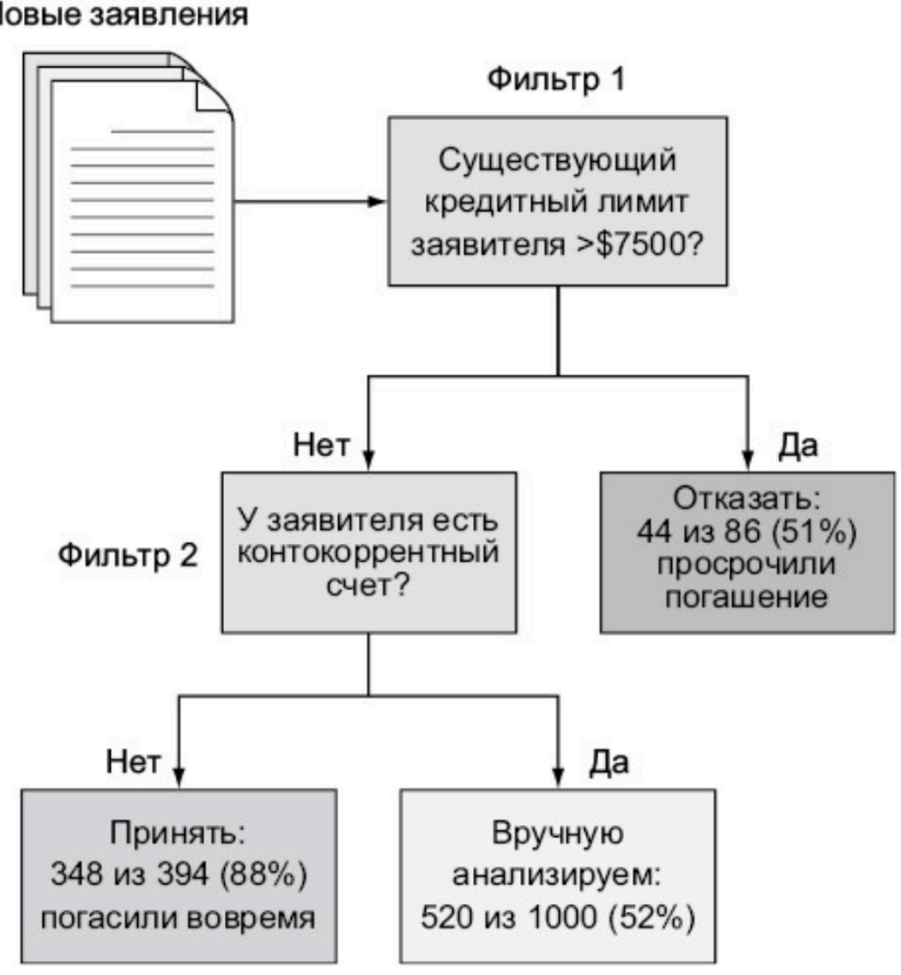
Что такое машинное обучение

Машинное обучение (МО) - наиболее распространенный и мощный метод анализа данных.
Это **процесс**, в ходе которого система обрабатывает **большое число примеров**, выявляет **закономерности** и использует их, чтобы **прогнозировать** характеристики **новых данных**.

Для МО необходимо:

- большой объем данных для обучения;
- возможность представления данных в виде набора признаков.

Распространенный пример задачи МО:
одобрение кредита клиенту банка.



Постановка задачи МО

Цель машинного обучения — обнаружение закономерностей и взаимосвязей в данных и практическое применение полученной информации.

Общая постановка задачи:

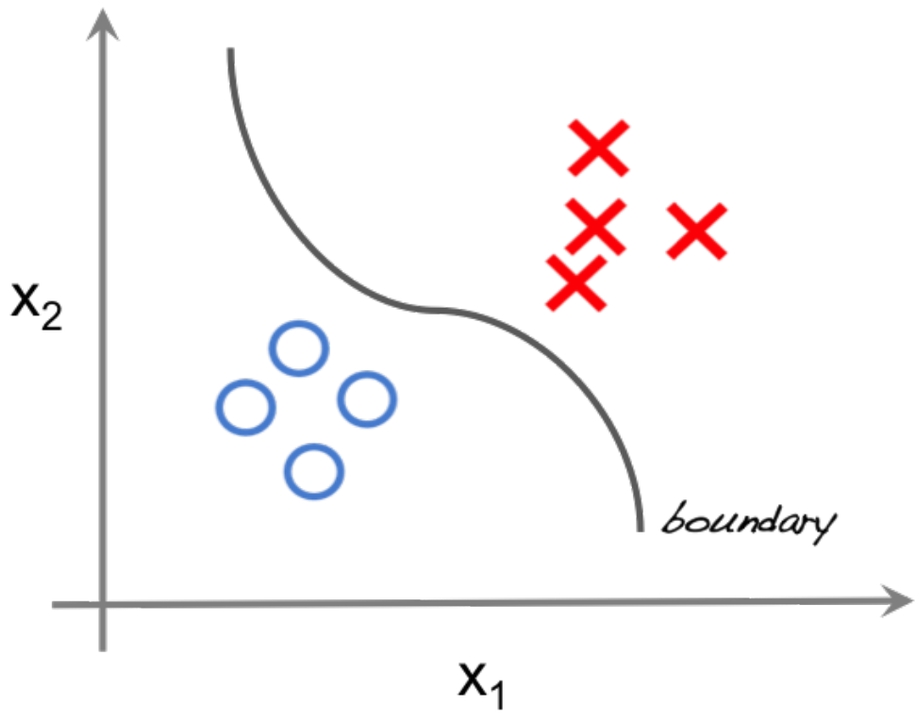
Дано: конечное множество ***прецедентов*** (объектов, ситуаций), по каждому из которых собраны (измерены) некоторые данные. Данные о прецеденте называют также его описанием. Совокупность всех имеющихся описаний прецедентов называется ***обучающей выборкой***.

Требуется: по этим частным данным выявить ***общие зависимости, закономерности, взаимосвязи***, присущие не только этой конкретной выборке, но вообще всем прецедентам, в том числе тем, которые ещё не наблюдались.

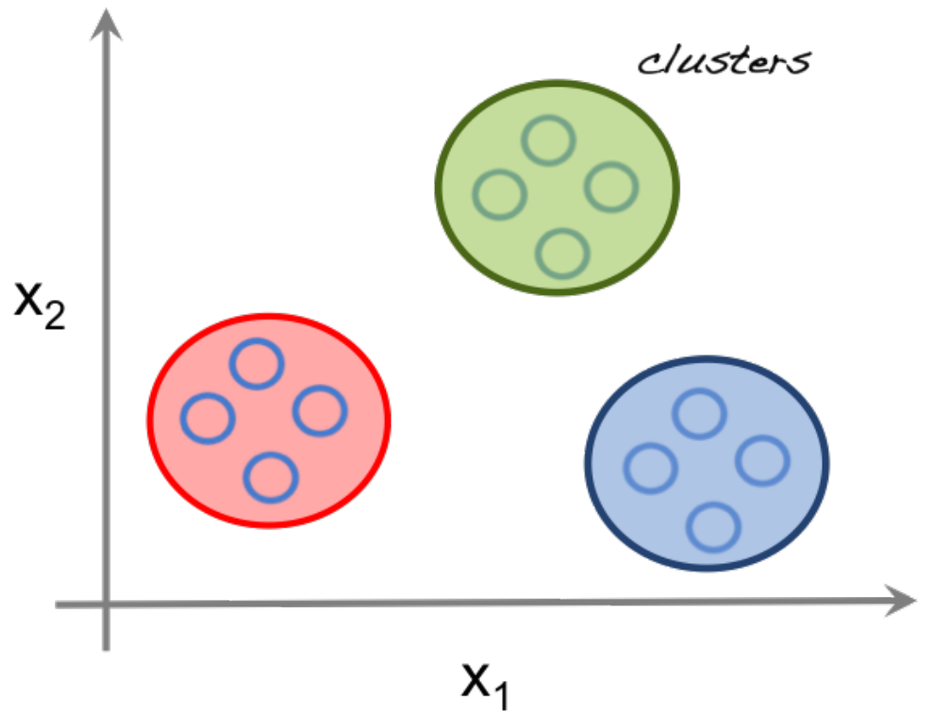
Типы задач МО

Задачи с машинным обучением делятся на два типа — **обучение с учителем (supervised learning)** и **обучение без учителя (unsupervised learning)**.

Supervised learning



Unsupervised learning



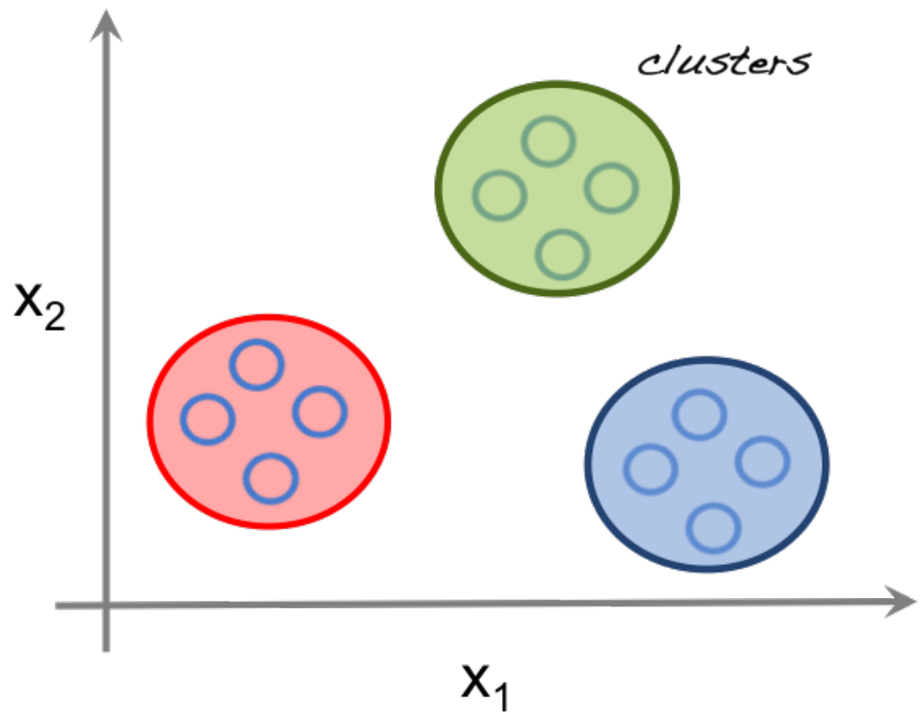
Обучение без учителя

Обучение без учителя (unsupervised learning). Каждый прецедент представляет собой «объект» без «ответа». Требуется искать зависимости между объектами.

Это методы решения задачи, где **правильный ответ не определен**.

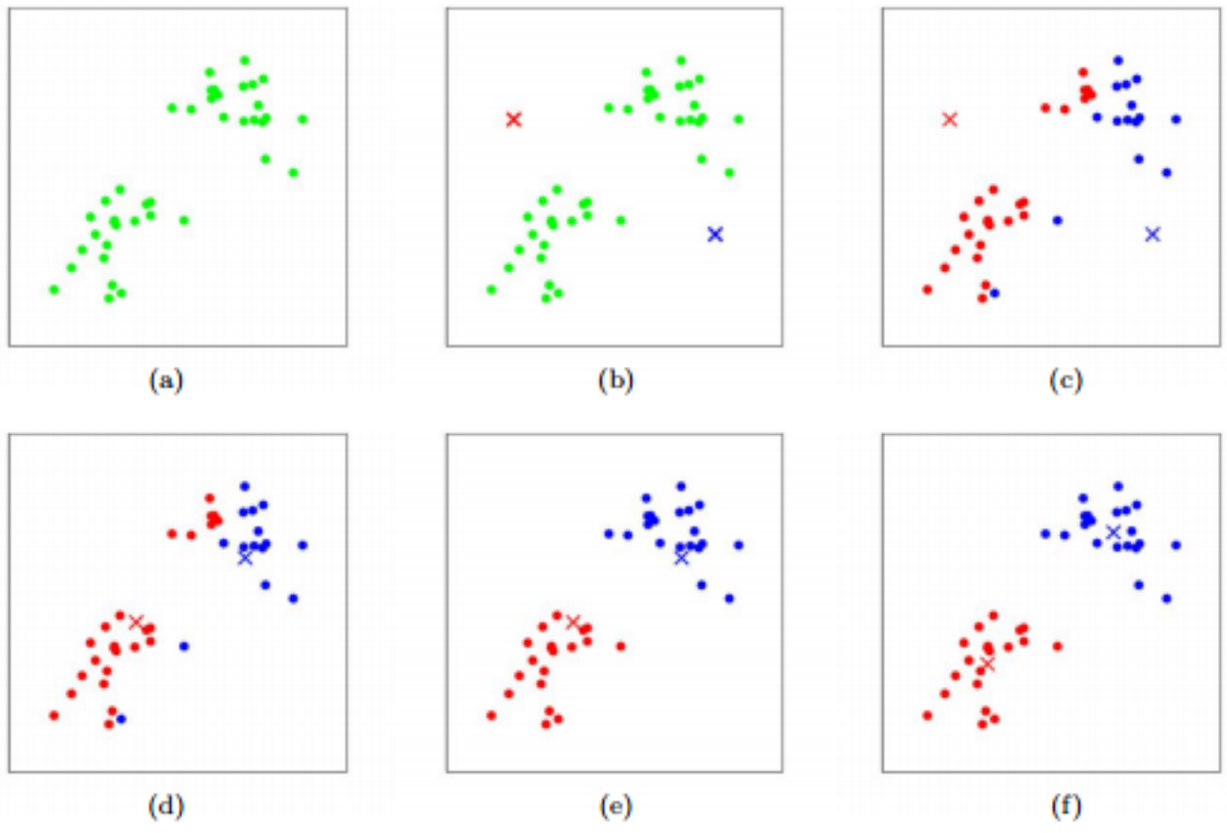
Основные решаемые задачи: кластеризация, фильтрация выбросов.

Задача **кластеризации** (clustering) заключается в том, чтобы **сгруппировать объекты** в кластеры, используя данные о попарном сходстве объектов.



Кластеризация

Метод кластеризации **KMeans.**



Векторизация

Построим векторы для текстов.

```
vectorizer_c = CountVectorizer(min_df=1)
X_train_c = vectorizer_c.fit_transform(list(flat_list))
#число сообщений, число слов
count_samples_c, count_features_c = X_train_c.shape
#print('titles_count=%d, words_count=%d' % (count_samples_c, count_features_c))

#сообщение-вопрос
request_2 = 'safely travel tour Amazon jungle'
request_vect_c = vectorizer_c.transform([request_2])
print(request_vect_c.toarray())
```

Кластеризация

Выполним кластеризацию методов K-средних и выведем топ-10 слов для центра каждого кластера.

```
#кластеризация
from sklearn.cluster import KMeans
clusters_count = 3
model = KMeans(n_clusters=clusters_count, init='k-means++', max_iter=100, n_init=1)
model.fit(X_train_c)
```

```
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = Tfidf_vectorizer.get_feature_names()
for i in range(clusters_count):
    print ("Cluster %d:" % i,)
    for ind in order_centroids[i, :10]:
        print (' %s' % terms[ind],)
```

Кластеризация

Выполним кластеризацию методов K-средних и выведем топ-10 слов для центра каждого кластера.

```
Top terms per cluster:
Cluster 0:
 around
 know
 tickets
 flights
 good
 atlitlan
 north
 new
 country
 n
Cluster 1:
 service
 best
 hong
 dallas
 airline
 buy
 kong
 ticket
 tickets
 work
Cluster 2:
 traps
 good
 country
 western
 best
 find
 visas
 vietnam
 wall
 get
```

Определение редких слов

Есть способ при векторизация повысить вес редких слов.

TF-IDF - характеристика, позволяющая оценить важность слова во всем корпусе документов.

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

Определение редких слов

Построим векторизатор с использованием TF-IDF.

```
#Самые частые и редкие слова
from sklearn.feature_extraction.text import TfidfVectorizer

def identity_tokenizer(text):
  return text
Tfidf_vectorizer = TfidfVectorizer(tokenizer=identity_tokenizer,lowercase=False)
vectorized = Tfidf_vectorizer.fit_transform(final_data.values)
print(vectorized.shape)
```

```
(100, 418)
```

Определение редких слов

Выведем слова с наибольшим и наименьшим весом.

```
indices = np.argsort(Tfidf_vectorizer.idf_)[::-1]
features = Tfidf_vectorizer.get_feature_names()
top_n = 50
top_freq_features = [features[i] for i in indices[-top_n:]]
print(top_freq_features)
```

```
['information', 'money', 'etc', 'anyone', 'traveling', 'another', 'area', 'able', 'safe', 'country', 'without', 'trav
elling', 'even', 'first', 'things', 'possible', 'many', 'us', 'also', 'planning', 'people', 'could', 'really', 'wonde
ring', 'visiting', 'see', 'getting', 'find', 'want', 'going', 'go', 'use', 'much', 'good', 'looking', 'take', 'year',
'around', 'countries', 'way', 'visit', 'know', 'best', 'travel', 'get', 'like', 'time', 'one', 'trip', 'would']
```

```
top_rare_features = [features[i] for i in indices[:top_n]]
print(top_rare_features)
```

```
['limitations', 'hollywood', 'hawaii', 'helena', 'helped', 'hemisphere', 'hikes', 'hiring', 'historical', 'honest', '
happens', 'hooray', 'hope', 'hoped', 'hostel', 'hostile', 'hot', 'hotels', 'haul', 'happened', 'hundreds', 'guests',
'great', 'groggy', 'ground', 'group', 'guangzhou', 'guatemala', 'guess', 'guided', 'hanoi', 'guys', 'habit', 'hackett
', 'handful', 'handled', 'handling', 'hands', 'houses', 'hungary', 'israel', 'interest', 'instability', 'instigate',
'intend', 'intended', 'intending', 'intense', 'inter', 'interests']
```

Кластеризация

Кластеризуем тексты.

```
#кластеризация
from sklearn.cluster import KMeans
clusters_count = 3
model = KMeans(n_clusters=clusters_count, init='k-means++', max_iter=100, n_init=1)
model.fit(vectorized)

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = Tfidf_vectorizer.get_feature_names()
for i in range(clusters_count):
    print ("Cluster %d:" % i,)
    for ind in order_centroids[i, :10]:
        print (' %s' % terms[ind],)
```

Кластеризация

Кластеризуем тексты.

```
Top terms per cluster:
Cluster 0:
 best
 good
 visa
 usa
 around
 country
 visit
 russia
 travelling
 cheapest

Cluster 1:
 travel
 without
 agency
 us
 possible
 fund
 work
 seasonal
 flying
 towns

Cluster 2:
 safe
 place
 ways
 find
 travel
 good
 orlando
 florida
 take
 get
```

Задание 4

1. Сформулируйте предположение о том, на какие кластеры могут быть разделены ваши данные.
2. Постройте векторизатор ваших текстов с использованием характеристики TF-IDF.
3. Выведите топ-50 слов с наибольшим весом и с наименьшим. Как вы предполагаете, почему именно такие слова имеют наибольший и наименьший вес?
4. Выполните обучение кластеризатора:
    1. с использованием векторизация из прошлых заданий;
    2. с использованием характеристики TF-IDF;
    3. ДО предобработки и ПОСЛЕ предобработки.