# **Работа с текстом**

Начнем с того, что посмотрим на наши данные:

import matplotlib

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# Настройки для визуализации

# Если используется темная тема - лучше текст сделать белым

TEXT\_COLOR = 'black'

matplotlib.rcParams['figure.figsize'] = (15, 10)

matplotlib.rcParams['text.color'] = 'black'

matplotlib.rcParams['font.size'] = 14

matplotlib.rcParams['axes.labelcolor'] = TEXT\_COLOR

matplotlib.rcParams['xtick.color'] = TEXT\_COLOR

matplotlib.rcParams['ytick.color'] = TEXT\_COLOR

# Зафиксируем состояние случайных чисел

RANDOM\_STATE = 42

np.random.seed(RANDOM\_STATE)

from sklearn.datasets import fetch\_20newsgroups

# В функции загрузки уже есть разделение на обучение/тест

# воспользуемся этим на момент подготовки модели

# Для анализа лучше посмотреть на все данные

newsgroups\_data = fetch\_20newsgroups(subset='all', random\_state=RANDOM\_STATE)

print(newsgroups\_data.keys())

# Посмотрим, какой у данных тип

data = newsgroups\_data['data']

targets = newsgroups\_data['target']

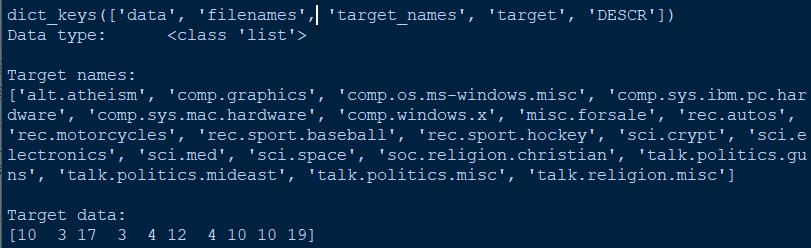
target\_names = newsgroups\_data['target\_names']

print(f"Data type:\t{type(data)}\n")

print(f"Target names:\n{target\_names}\n")

print(f"Target data:\n{targets[:10]}")

Результат



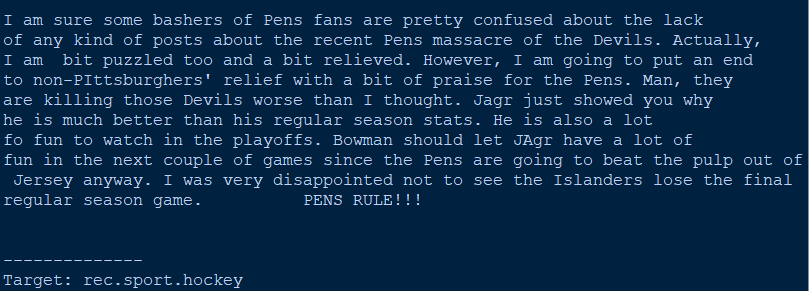
Таакс, данные - это какой-то список и мы скоро узнаем какой именно, целевые переменные - индексы категорий. Посмотрим на пример данных:

print(data[0])

print('--------------')

print(f'Target: {target\_names[targets[0]]}')

Результат



# **BOW**

# Здесь специально сделана некоторая предобработка,

# которая обычно делается в рамках подготовки

texts\_dataset = [

"пирожок это лишь пирожок",

"пирог не кушать пирожок можно",

"сегодня ходил кино поел пирог"

]

corpus = set()

# Для начала составим словарь

for text in texts\_dataset:

tokens = text.split(' ')

corpus.update(tokens)

corpus = list(corpus)

print(f'Corpus: {corpus}')

# После составления корпуса мы можем составить матрицу попаданий

samples\_count = len(texts\_dataset)

corpus\_len = len(corpus)

X\_data = np.zeros((samples\_count, corpus\_len), dtype=int)

for i\_sample, text in enumerate(texts\_dataset):

tokens = text.split(' ')

for token in tokens:

token\_index = corpus.index(token)

X\_data[i\_sample, token\_index] += 1

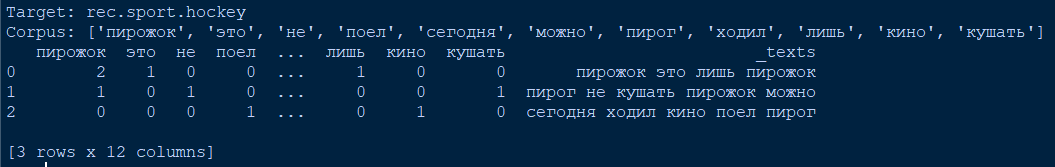
# Для лучшего представления составим DataFrame

X\_df = pd.DataFrame(X\_data, columns=corpus)

X\_df['\_texts'] = texts\_dataset

print(X\_df)

Результат



Как мы можем отнормировать векторы? Да просто поделить на количество слов в каждом тексте, то есть для каждой строки находим сумму элементов в строке и всю строку делим на нее:

row\_sums = X\_data.sum(axis=1)

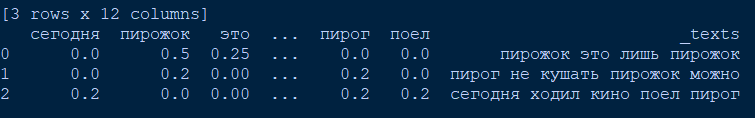
X\_data\_norm = X\_data/row\_sums[:,None]

X\_df = pd.DataFrame(X\_data\_norm, columns=corpus)

X\_df['\_texts'] = texts\_dataset

print(X\_df)

Результат



# **TF-IDF**

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(

# Ограничение максимального кол-ва признаков (размера выходного вектора)

# None -> не ограничено, вычисляется из данных

max\_features=None,

)

X\_data = [

"Пирожок - это лишь пирожок!",

"Пирог не кушать, пирожок - можно.",

"Я сегодня ходил в кино и поел пирог!"

]

X\_data\_vec = vectorizer.fit\_transform(X\_data)

# Отобразим векторизированное представление (кол-во данных, кол-во фич)

print(X\_data\_vec.shape)

# Мы можем проверить корпус, который сформировался при генерации

corpus = vectorizer.get\_feature\_names()

print(corpus)

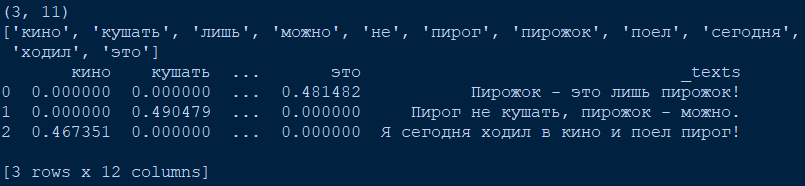
dict(zip(corpus, vectorizer.idf\_))

df = pd.DataFrame(X\_data\_vec.todense(), columns = corpus)

df['\_texts'] = X\_data

print(df)

Результат



# **Байесовский классификатор**

Для примера возьмем следующие данные:

df = pd.DataFrame({

'x1': [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2],

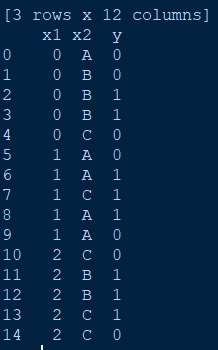
'x2': ['A', 'B', 'B', 'B', 'C', 'A', 'A', 'C', 'A', 'A', 'C', 'B', 'B', 'C', 'C'],

'y': [0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0]

})

print(df)

Результат



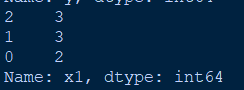
Начнем с первых двух, это априорные вероятности, то есть, нам просто нужно определить, насколько вероятно на основе данных получить запись с классом IMG_256 или IMG_257. Как это сделать? Поделить количество записей класса на общее количество записей!

df['y'].value\_counts()

Давайте посмотрим, какие показатели нам нужны для вероятностей:

print(df.loc[df['y'] == 1, 'x1'].value\_counts())

Результат



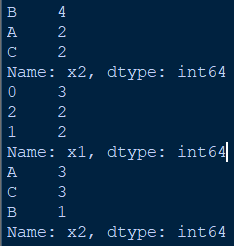
По аналогии считаем остальные показатели:

print(df.loc[df['y'] == 1, 'x2'].value\_counts())

print(df.loc[df['y'] == 0, 'x1'].value\_counts())

print(df.loc[df['y'] == 0, 'x2'].value\_counts())

Результат



Интерфейс работы с ничем не отличается от того, что мы привыкли, вот класс [MultinomialNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html):

# Подгрузим данные

newsgroups\_train = fetch\_20newsgroups(

subset='train', random\_state=RANDOM\_STATE)

newsgroups\_test = fetch\_20newsgroups(

subset='test', random\_state=RANDOM\_STATE)

X\_train = newsgroups\_train['data']

y\_train = newsgroups\_train['target']

X\_test = newsgroups\_test['data']

y\_test = newsgroups\_test['target']

from sklearn.naive\_bayes import MultinomialNB

# Без всяких предобработок кидаем, что есть в трансформацию

vectorizer = TfidfVectorizer()

X\_train\_vec = vectorizer.fit\_transform(X\_train)

X\_test\_vec = vectorizer.transform(X\_test)

nb\_clf = MultinomialNB()

nb\_clf.fit(X\_train\_vec, y\_train)

print(f'Train accuracy: {nb\_clf.score(X\_train\_vec, y\_train)}')

print(f'Test accuracy: {nb\_clf.score(X\_test\_vec, y\_test)}')

Результат



# **Предобработка данных**

import nltk

# Скачиваем необходимые модули фреймворка nltk

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')

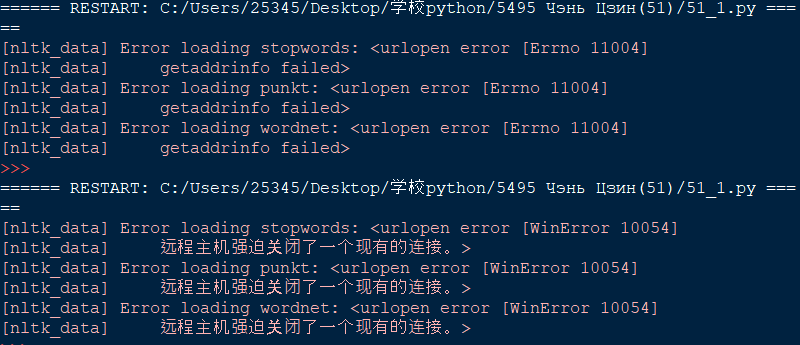
from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import re

from string import punctuation

Результат



# **Задачи**

* Проведите анализ текстовых данных набора данных 20newsgroups;
* Постройте модель MultinominalNB с параметрами по-умолчанию и без предобработки текста, только векторизация (baseline);
* Добавьте предобработку текста, оцените работу модели;
* Отобразите CM и сделайте выводы, какие разделы путаются;
* Изучите влияние α байесовского классификатора (7 значений);
* Изучите влияние максимального количества признаков TfidfVectorizer (10 значений max\_features);
* Оцените работу модели при различных значения ngram\_range (варьируя верхний предел - от 1 до 3 включительно);
* Определите лучшую модель путем настройки гиперпараметров (по показателю f1\_macro), отобразите CM и отчет по классификации;
* Постройте модель случайного леса, определите лучшие гиперпараметры и сравните с моделью байесовского классификатора;
* Проанализируйте ошибки классификаторов и сделайте выводы;
* Удалите дополнительную информацию (используйте аргумент remove в функции fetch\_20newsgroups() с указанием в качестве значения кортеж ('headers', 'footers', 'quotes')) -> оцените работу лучшей модели (с переобучением на очищенном наборе данных);
* Определите новую лучшую модель (с учетом удаления доп.информации из текстов) путем настройки гиперпараметров (по показателю f1\_macro);
* Отобразите CM и сделайте выводы, какие разделы путаются;

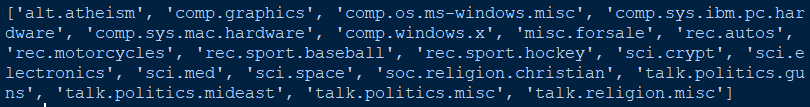
from sklearn.datasets import fetch\_20newsgroups

from pprint import pprint

newsgroups\_train = fetch\_20newsgroups(subset='train')

print(list(newsgroups\_train.target\_names))

Результат



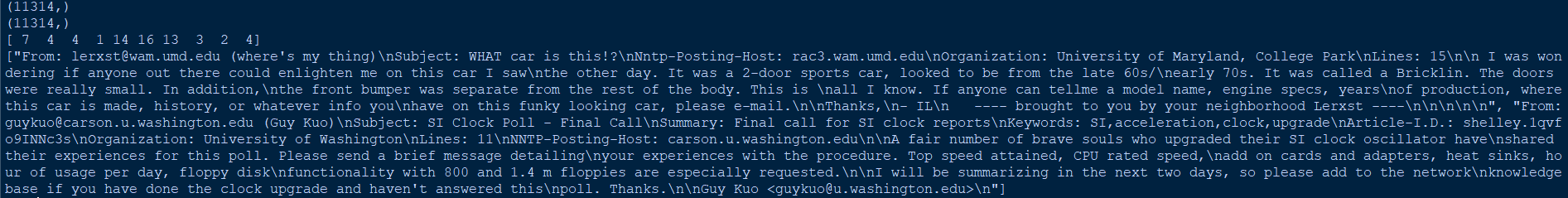
print(newsgroups\_train.filenames.shape)

print(newsgroups\_train.target.shape)

print(newsgroups\_train.target[:10])

print(newsgroups\_train['data'][:2])

Результат



fetch\_20newsgroups(data\_home=None,

subset='train',

categories=None,

shuffle=True,

random\_state=42,

remove=(),

download\_if\_missing=True

)

# **Преобразовать текст в (TF-IDF)вектор**

from sklearn.feature\_extraction.text import TfidfVectorizer

categories = ['alt.atheism', 'talk.religion.misc','comp.graphics', 'sci.space']

newsgroups\_train = fetch\_20newsgroups(subset='train',categories=categories)

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit\_transform(newsgroups\_train.data)

print(vectors.shape)

print(vectors.nnz / float(vectors.shape[0]))

Результат



**Классификация с использованием байесовского метода**

from sklearn.feature\_extraction.text import TfidfVectorizer

categories = ['alt.atheism', 'talk.religion.misc','comp.graphics', 'sci.space']

newsgroups\_train = fetch\_20newsgroups(subset='train',categories=categories)

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit\_transform(newsgroups\_train.data)

print(vectors.shape)

print(vectors.nnz / float(vectors.shape[0]))

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score,f1\_score

newsgroups\_test=fetch\_20newsgroups(subset='test',categories=categories)

vectors\_test=vectorizer.transform(newsgroups\_test.data)

clf=MultinomialNB(alpha=0.1)

clf.fit(vectors,newsgroups\_train.target)

pred=clf.predict(vectors\_test)

print(f1\_score(newsgroups\_test.target,pred,average='macro'))

print(accuracy\_score(newsgroups\_test.target,pred))

Результат

