# **1.1.1 Парсинг данных**

Для начало мы импортируем библиотеки для удобной работы

import pandas as pd

import requests

from bs4 import BeautifulSoup as bs

import re

import pymorphy2

# **1.1.2 Парсинг данных**

Потом мы импортировали каждый файл для того чтоб работать с ними, всего получилось их три вместо пяти (нам не удалось прочитать два файла Инотех и skillbox, из того что он был испорчен или поврежден ):

-Gett.json,

-Здоровье.ру,

-Проект по использованию технологий компьютерного зрения на базе искусственного интеллекта для анализа медицинских изображений.json

**1.1.3 Дальше мы парсим информацию из сайта**

Подключаем и проверяем соединение с сайтом, и ищем нужные нам статьи.

Так же мы создаём переменную в которой будет названия каждой компании,

(company **=** ['avito', 'vtb', 'vk', 'alfa', 'kaspersky', 'rosatom', 'rostelecom', 'spbifmo', 'sbermarket', 'agima', 'mkb', 'tensor'])

это сделано для того чтоб в ручную не писать один тот же код для всех компаний

Дальше мы разделяем одну колонку(refs) на несколько других(day, timе, rate, views).

Дальше мы обрабатываем текст, приводим весь текст к одному регистру(Нижний), так же удаляем все спецсимволы и лишние пробелы, для того чтоб в будущем было более точная информация для обучения.

# **2.1.1 Поиск ключевых слов/n-грамм. Векторизация текстов**

Для начало мы импортируем библеотеки которые будем использовать в этой работе

import pandas as pd

import pymorphy2

from sklearn.feature\_extraction.text import CountVectorizer

так же мы удаляем повторный индекс для того чтоб нам эти значения не как не повлияли на работу,

**2.1.2 всего я буду использовать 3 модели для векторизации:**

1.TfidfVectorizer - Преобразование набора необработанных документов в матрицу функций TF-IDF. я буду использовать её так как она хорошо работает с большим количество данных.

2.HashingVectorizer - она имеет ряд преимуществ к примеру это маленькая память она масштабирует.

3.CountVectorizer - Преобразуйте коллекцию текстовых документов в матрицу количества токенов.

После всех проверок я буду использовать модель CountVectorizer так как он выводит целые число.

# **3.1.1Разбиение выборки**

Для начало мы импортируем нужные нам библеотеки:

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression

from sklearn.datasets import load\_iris

from sklearn.feature\_selection import SelectKBest, chi2

from sklearn.decomposition import PCA

### **3.1.2 разбиение полученной выборки на обучающую и тестовую**

Для начало мы выполните разбиение полученной выборки на обучающую и тестовую

для этого мы будем использовать библеотеку train\_test\_split.

после разделения можно переходить для обучения моделей.

### **3.1.3 обучение моделей.**

### **Для обучения моделей я буду использовать три модели:**

#### **1 RandomForestClassifier**

Алгоритм также основанный на деревьях решений, но вместо последовательного обучения обучает множество деревьев на разных частях датасета и усредняет значения. Подходит для достаточно больших датасетов, однако HistGradientBoostingClassifier зачастую справляется быстрей и с большей точностью

#### **2 HistGradientBoostingClassifier**

Алгоритм основаный на деревьях решений, и работающий так, что деревья последовательно обучаются учитывая ошибки предидущих деревьев. Данная модель зачастую показывает одну из самых лучших точностей предсказаний, но может работать слишком долго на больших данных. Если данных слишком много для GradientBoostingClassifier можно использовать HistGradientBoostingClassifier, который также может справляться с неизвестными значениями

#### **3 Logistic regression**

В отличие от трёх предыдущих моделей, является линейной моделью и имеет в себе совершенно иной принцип работы. Классифицирует только двоичные значения, однако также может классифицорать и множество классов, путём обучения модели на каждый класс. Модель работает очень быстро и подходит для очень больших датасетов, особенно при булевой классификации или классификации с малым количеством классов

### **3.1.3 оценку моделей**

Для оценки я буду использовать метрику:

#### **f1\_score -**Вычислите оценку F1, также известную как сбалансированная F-оценка или F-мера. он прост в использование а так же F1 пытается достигнуть своего наилучшего значения

после каждой проверенной модели, можно посмотреть какая модель показала наилучший показатель.

После проверок я выберу RandomForestClassifier так как он показал хорошие результат

# **3.2 Оптимизация модели.**

### **3.2.1 выбор значимых признаков.**

Для этой части я буду использовать две библеотеки.

##### ***SelectKBest / chi2***

Одномерный отбор признаков работает путем выбора лучших признаков на основе одномерных статистических тестов. Это можно рассматривать как шаг предварительной обработки к оценщику.

я её выбрал так как она удаляет всё, кроме самых результативные функции.

### **3.2.2 понижение размерности**

для понижения размерности я буду использовать

##### ***PCA -*** PCA используется для разложения многомерного набора данных на набор последовательных ортогональных компонентов, которые объясняют максимальную величину дисперсии. это то что нам нужно для наших данных.

### **3.2.3 настраивая гиперпараметры**

Для начало я выбрал два параметра в модели **RandomForestClassifier**

'n\_estimators' со значениями [60, 80, 100],

'max\_death', со значениями [None, 1, 10]

**4.1 Предсказание целевой переменной.**

я начну со второго задания для того чтоб было легче выполнить первое

# **4.2 Разработка прикладного решения.**

Для начало я импортирую библеотеку которые буду использовать для работы

from tkinter import \*

Дальше мы делаем api которое будет прогнозировать номинации конкурсанта, и так же добавил окно и немного текста

Для создание окна я решил использовать tkinter, так как он прост в использование.