Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторному проекту

По дисциплине «Модели и методы обработки больших объёмов данных»

По теме «Формирование ленты новостей по предпочтениям (тегам)»

Выполнил:

магистрант гр. 056241

Хлопцев А.А.

Проверил:

доцент кафедры информатики

Стержанов М.В.

Минск 2020

**Оглавление**

[Задача 3](#_Toc58528775)

[Основная идея 3](#_Toc58528776)

[Описание реализации проекта 3](#_Toc58528777)

[Получение данных 3](#_Toc58528778)

[Обработка данных 4](#_Toc58528779)

[Хранение данных 5](#_Toc58528780)

[Классификация статей 9](#_Toc58528781)

[Визуализация ленты новостей 10](#_Toc58528782)

[Вывод 12](#_Toc58528783)

# **Задача**

Создать проект, который будет агрегировать новости с интернет-ресурсов и с помощью алгоритмов машинного обучения устанавливать их принадлежность к различным темам (по тегам).

# **Основная идея**

Необходимо создать проект, состоящий из 3 частей.

1. Расширяемый модуль, в который можно добавить обработчики открытых API интернет-ресурсов. Данный модуль необходим для сбора информации.

2. Модуль, который будет рассортировывать и хранить собранную и обработанную информацию.

3. Модуль, реализующий некоторые алгоритмы машинного обучения, необходимые для классификации полученной информации и функции обработки полученной информации.

# **Описание реализации проекта**

## **Получение данных**

Для начала разработки был выбран один из самых популярных интернет-изданий TheGuardian. Данный ресурс был выбран по нескольким причинам.

1. Очень обширная база новостей на различные темы.

2. Имеется открытое API, для получения новостей (для использования надо запросить Api-key).

3. API запросы позволяют выбрать темы скачиваемых новостей, что сильно ускоряет реализацию проекта, т.к. позволяет пропустить этап ручной обработки (разметки) новостей.

Чтобы начать использовать API, необходимо пройти по ссылке https://open-platform.theguardian.com/access/ и запросить ключ доступа. Этим ключом мы будем помечать все наши запросы, чтобы нам была доступна не только тестовая коллекция новостей, но и остальные статьи.

После получения ключа и установки соединения с сервером необходимо реализовать несколько запросов для получения статей. В данном проекте статья включает в себя:

1. заголовок;

2. текст;

3. ссылка на статью;

4. ссылка на главную фотографию;

5. время размещения статьи.

В модуле TheGuardianParser было реализовано два основных вида команд:

1. Получение статей по тегу (это необходимо для получения размеченного набора статей, нужных для тренировки алгоритмов машинного обучения).

2. Получение последних новостей. Данный запрос не будет содержать тега, определяющего, какие именно новости скачивать, и будет производится каждые 5 минут.

Ограничения API состоит в том, что за один запрос можно получить максимум 50 статей, поэтому для запросов было реализовано постраничное чтение для получения необходимого кол-ва данных.

## **Обработка данных**

Для классификации статей был использован SVM алгоритм машинного обучения. Классификация будет производится по 7 темам:

* спорт;
* политика;
* наука;
* фильмы;
* музыка;
* экономика;
* все (включает все предыдущие темы + темы, не упомянутые в классификаторе).

Для того, чтобы лучше обучить ядро SVM алгоритма, данные необходимо очистить от лишнего "мусора" и "шума". Поэтому был разработан специальный модуль TextProcessor, который:

1. переводит текст в нижний регистр;

2. удаляет HTML тэги (если они есть в тексте);

3. заменяет URL на одно слово (“httpaddr”);

4. заменяет email-адреса на одно слово (“emailaddr”);

5. заменяет числа на одно слово (“number”);

6. заменяет знаки доллара ($) на слово “dollar”;

7. заменяет формы слов на исходное слово (такой подход называется stemming);

8. удаляет остальные символы и заменяет их пробелами;

9.  удаляет и заменяет "стоп-слова" (слова, не имеющие никакой информации) в тексте;

10. подсчитывает топ самых популярных слов для генерации словаря обучения.

Для реализации этих операций были использованы регулярные выражения и Python библиотека nltk с английским корпусом текстов.

После обработки группы статей, объединённых одной темой, на выходе мы получаем набор векторов, каждый из которых состоит только из информационных слов в их начальной форме и словарь самых популярных слов по данной теме.

После этого каждая статья кодируется в виде бинарного вектора X, длина которого равна длине словаря. 0 на i-м месте означает, что i-e слово в словаре не встречалось в данной статье, 1 – наоборот.

## **Хранение данных**

После обработки статей их необходимо сохранить для дальнейшей обработки и обучения SVM. Для этого была использована база данных MySQL, где каждому набору данных соответствует своя таблица, состоящая из:

1. Id - уникальный номер каждой обработанной статьи;

2. текст - строка, содержащая текст статьи (для повторного обучения в случае необходимости);

3. X - бинарный вектор обработки статьи на словаре этой темы;

4. y - бинарный вектор для обучения SVM состоящий из 0. Единственный элемент равен 1 - номер темы, к которой принадлежит статья.

База данных содержит следующие таблицы тренировочных данных (рис. 1):

* economy\_train\_data;
* films\_train\_data;
* musics\_train\_data;
* politics\_train\_data;
* science\_train\_data;
* sport\_train\_data.

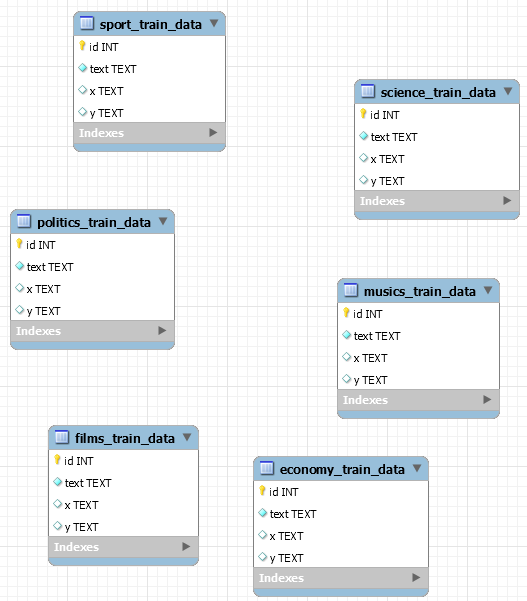


Рис. 1. ER диаграмма таблиц тренировочных данных

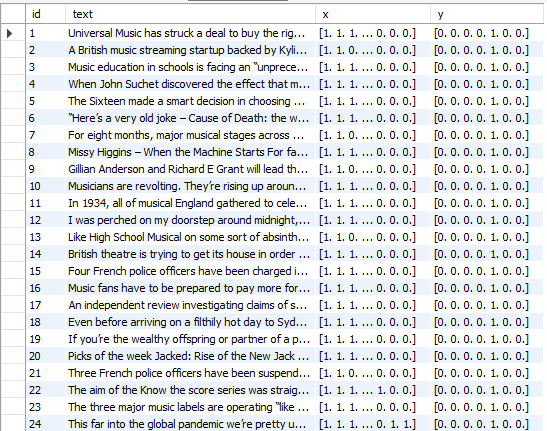


Рис. 2. Пример хранения тренировочных данных (таблица music\_train\_data)

Также в базе данных хранится одна общая таблица для статей, которые используются непосредственно для формирования ленты новостей (рис. 3), она состоит из:

1. Id - уникальный номер каждой обработанной статьи;

2. заголовок - строка содержащая заголовок статьи;

3. текст – строка, содержащая обрезанный текст статьи (около 100 символов);

4. ссылка на статью – ссылка, по которой можно перейти к оригиналу статьи;

5. ссылка на главную фотографию – ссылка, по которой будет грузится превью статья;

6. время размещения статьи - дата, когда данная статья была размещена;

7. тег - строка, содержащая тему, к которой принадлежит данная статья (рис. 4).

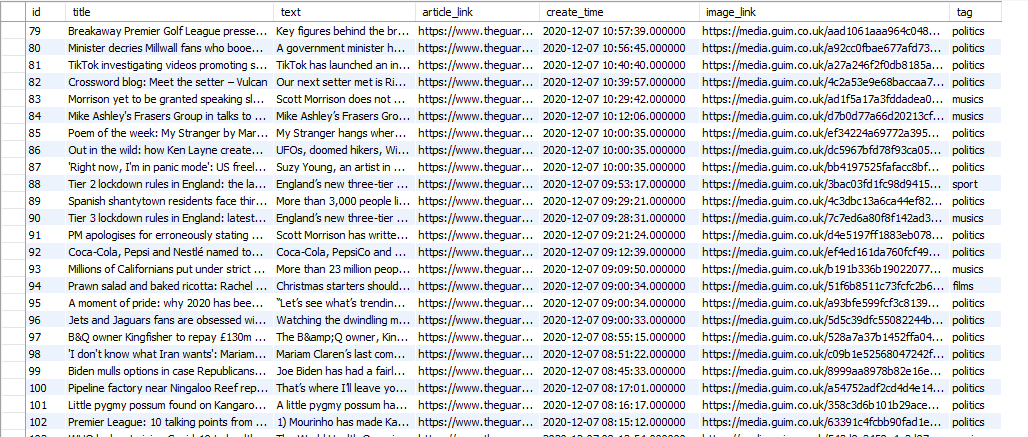


Рис. 3. Пример хранения классифицированных данных

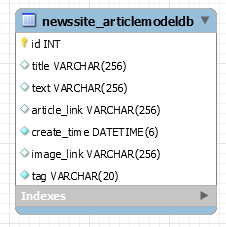


Рис. 4. ER диаграмма таблицы классифицированных данных

Также в папку "Cache models" были сохранены словари для каждой темы и коэффициенты SVM, чтобы не было необходимости обучать их заново при каждом запуске.

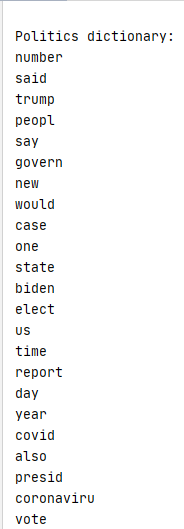


Рис. 5. Пример первых самых популярных слов в теме политика

## **Классификация статей**

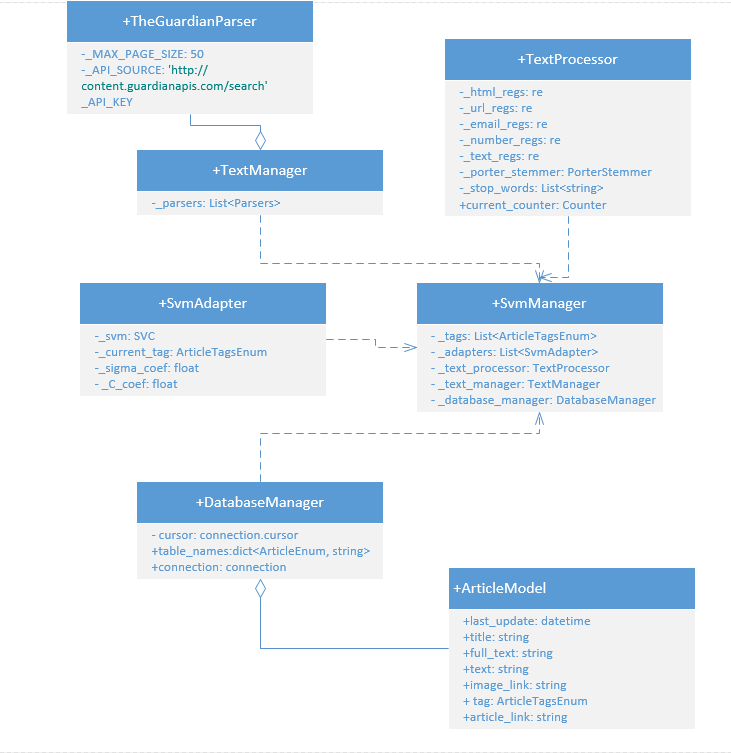
По итогу предыдущих пунктов созданы модули для сбора и обработки информации, необходимой для обучения алгоритмов машинного обучения, которые должны классифицировать поступившую статью по одной из 6 тем (или не классифицировать ни к одной, тогда статья будет помечена тегом all). Для этого было создано 6 ядер SVM с функцией Гауссового ядра. Для каждого ядра был подготовлен свой словарь самых популярных слов (1200 слов) в статьях данной темы, и набор обработанных статей, состоящий из 12000 векторов (по 2000 на каждую тему).

Однако, прежде чем обучать модели, необходимо было выбрать правильные коэффициенты C и sigma для тренировки модели. Для этого был создан небольшой набор данных 3000 статей (по 500 с каждой темы) в качестве тренировочного набора и 600 (по 100 с каждой темы) в качестве валидационного. После этого алгоритмом полного перебора (brute-force) были выявлены коэффициенты для каждого ядра, на котором наблюдается наименьшее число отклонений.

* Спорт: С=100, sigma=0.0209
* Политика: С=100, sigma=0.0023
* Наука: С=100, sigma=0.0082
* Фильмы: С=100, sigma=0.0167
* Музыка: С=100, sigma=0.0182
* Экономика: С=100, sigma=0.0137

После получения коэффициентов, ядра были обучены на выборке из 12000 статей. Далее точность обучения была проверена на новой тестовой выборке в 2400 (по 400 статей каждой темы). Полученная точность:

* Спорт: 89.83333333333333%;
* Политика: 96.95833333333333%;
* Наука: 99.08333333333333%;
* Фильмы: 92.08333333333333%;
* Музыка: 92.66666666666666%;
* Экономика: 98.95833333333334%.



Итого общая схема модулей приложения (рис. 6).

## **Визуализация ленты новостей**

Для визуализации ленты новостей был создан проект на Django 3.1. Данный проект содержит 1 страницу, на которую выводятся новости в зависимости от тега, выбранного сверху страницы. Перед запуском проекта с TheGuardian были скаченны и классифицированы 500 статей. Далее скрипт в фоновом потоке каждые 5 минут обращается к серверу за получением новых статей. После их классификации они загружаются в базу данных и отображаются после перегрузки страницы или смены фильтра.

Приведём примеры интерфейса приложения для некоторых тегов (рис. 7-9).

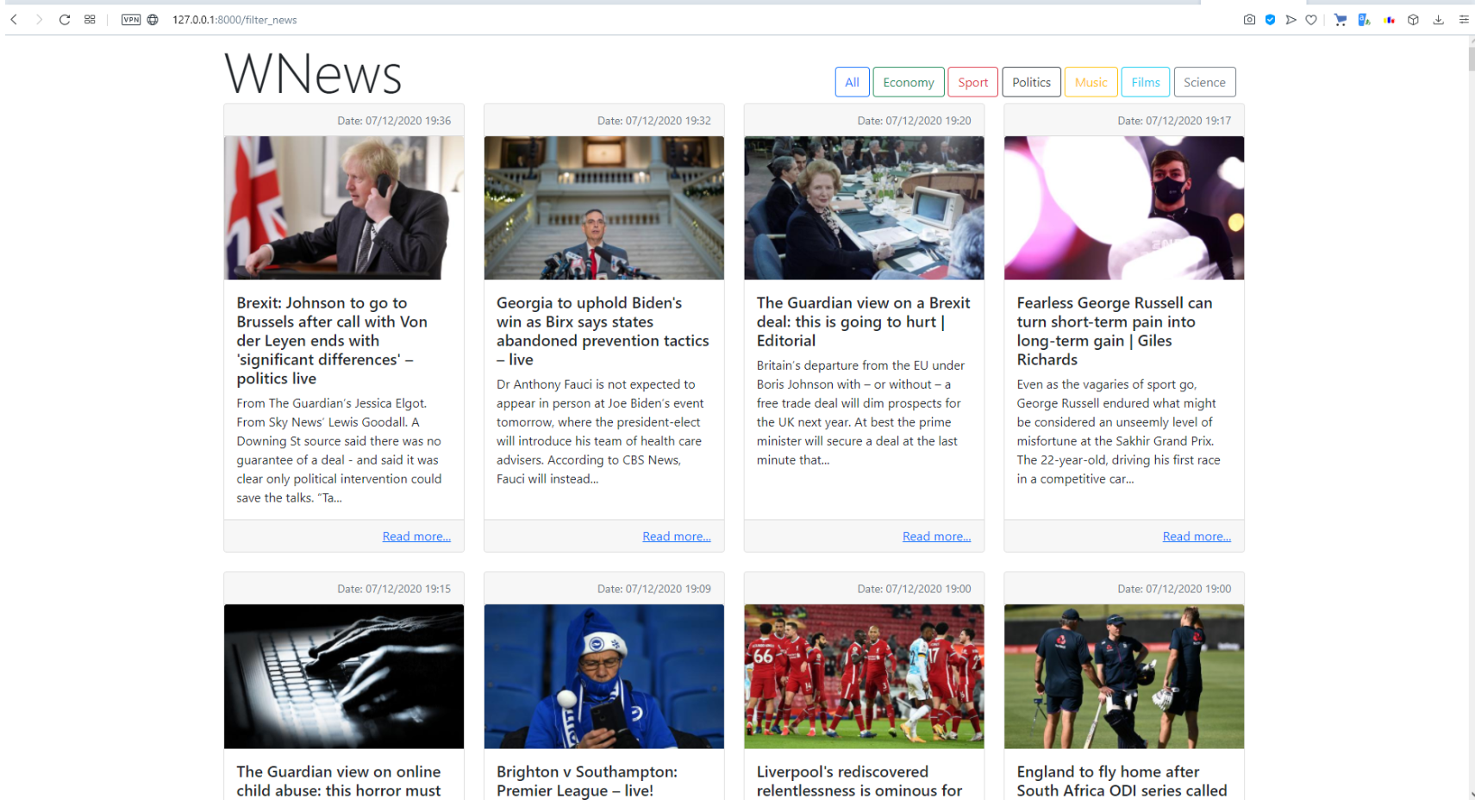


Рис. 7. Базовый интерфейс приложения (фильтр ”all”)

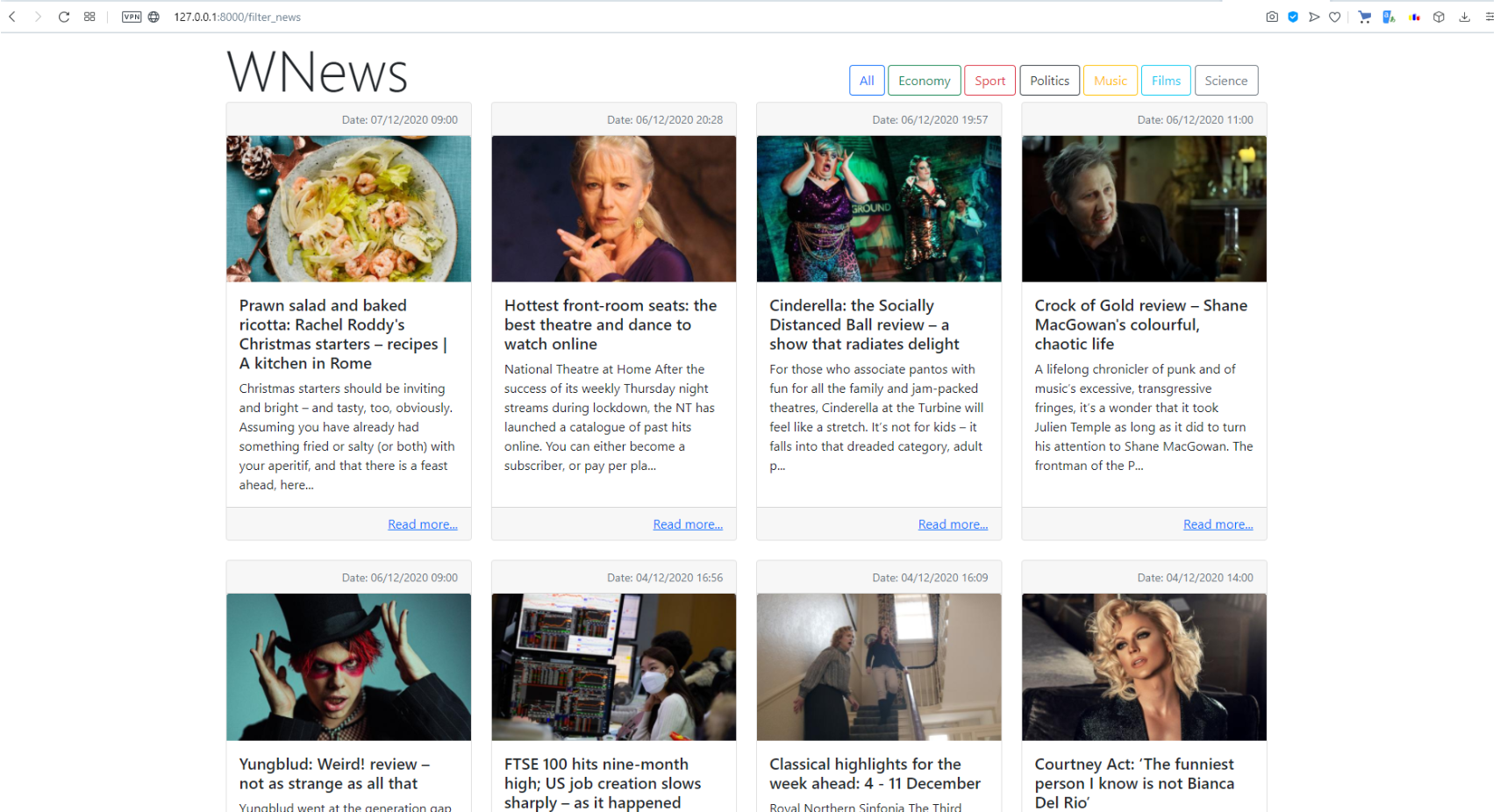


Рис. 8. Интерфейс приложения (фильтр “films”)

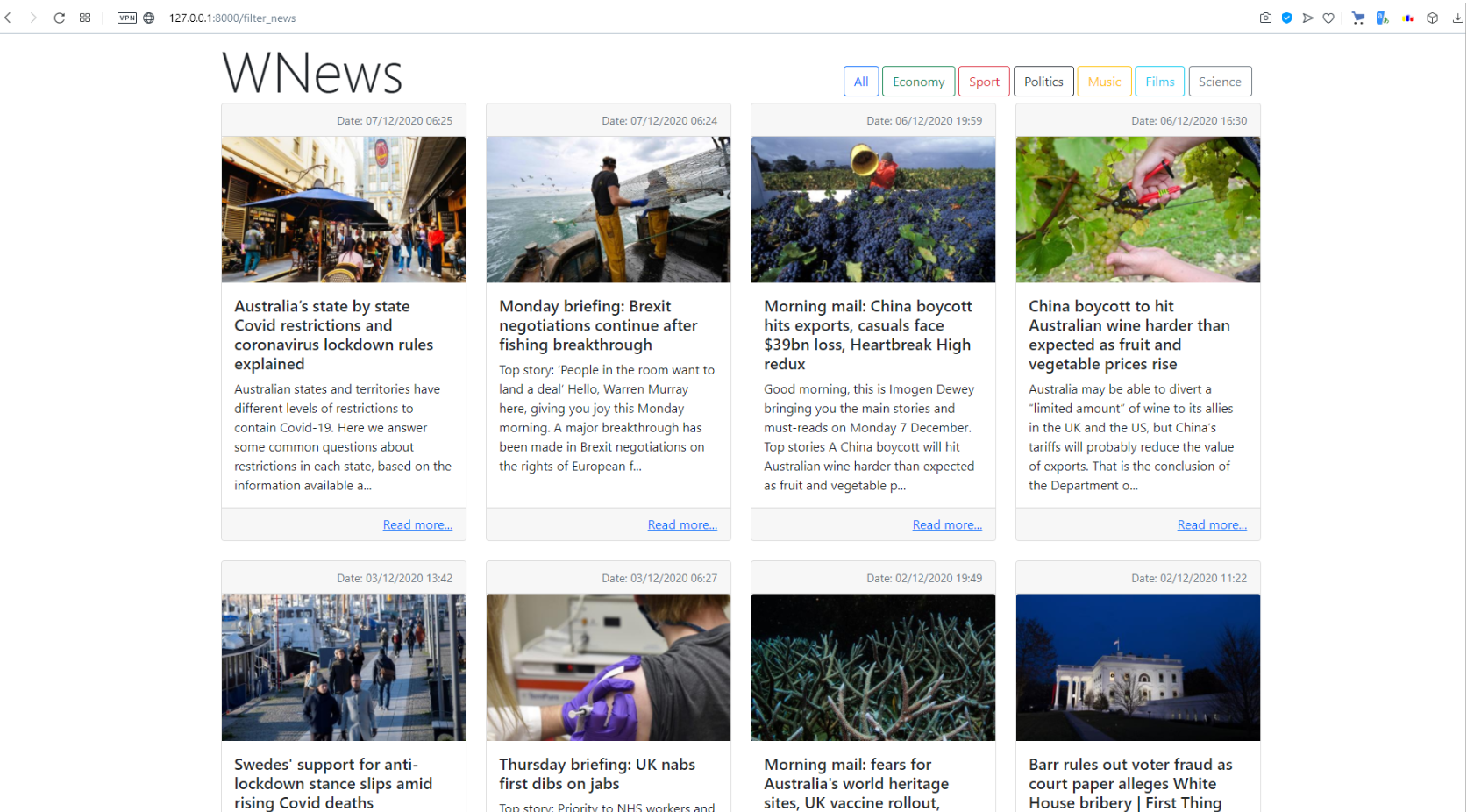


Рис. 9. Интерфейс приложения (фильтр “economy”)

# **Вывод**

В данном лабораторном проекте была разработана программа для сбора, обработки и хранения новостных статей. Данная программа состоит из 3 независимых модулей, каждый из которых предназначен для выполнения своих задач. Масштабируемый модуль Text Manager предназначенный для добавления открытых API для получения новых статей. Модуль TextProcessor – для обработки полученной информации. Модуль DatabaseStorage – для хранения обработанных данных и модуль SvmLib, содержащий 6 обученных ядер SVM, для классификации текста. Также был разработан сайт на Django 3.1 для отображения обработанной информации.

**Приложение 1 (исходный код программы)**

Модуль SvmManager:

class SvmManager:  
 \_tags = None  
 \_adapters = None  
 \_c\_vector = None  
 \_sigma\_vector = None  
 \_precision\_vector = None  
  
 \_text\_processor = TextProcessor()  
 \_text\_manager = TextManager()  
 \_database\_manager = DatabaseManager()  
  
 def \_\_init\_\_(self, \*args):  
 self.\_tags = args  
 self.\_adapters = [SvmAdapter(tag) for tag in args]  
  
 self.\_c\_vector = np.zeros(len(args))  
 self.\_sigma\_vector = np.zeros(len(args))  
 self.\_precision\_vector = np.zeros(len(args))  
  
 self.\_database\_manager.create\_connection(**"localhost"**, **"root"**, **"123qwe!"**, **"newsbase"**)  
 self.create\_tables()  
 print(**'SvmManager has been creating'**)  
  
 def get\_train\_data(self, articles\_count, features\_count, shift\_articles=0, save=False, dicts=None):  
 X = None  
 Y = None  
  
 for tag, adapter, dict in zip(self.\_tags, self.\_adapters, dicts):  
 print(tag, articles\_count)  
 \_, texts = self.\_text\_manager.get\_articles(tag, articles\_count)  
 new\_x, current\_dict = self.\_text\_processor.process\_articles(texts[shift\_articles:], features\_count, dict)  
 new\_y = adapter.get\_label\_matrix(new\_x, len(ArticleTagsEnum))  
 X = self.add\_rows(X, new\_x)  
 Y = self.add\_rows(Y, new\_y)  
  
 if save:  
 np\_texts\_array = np.array\_split(np.array(texts), 50)  
 np\_x\_array = np.array\_split(new\_x, 50)  
 np\_y\_array = np.array\_split(new\_y, 50)  
  
 *#adapter.save\_dictionary(current\_dict)* for nt, nx, ny in zip(np\_texts\_array, np\_x\_array, np\_y\_array):  
 self.\_database\_manager.add\_new\_articles(tag, nt, nx, ny)  
  
 return X, Y  
  
 def train\_adapters(self, X, Y, C\_coef, sigma\_arr):  
 for tag, adapter, sigma\_coef in zip(self.\_tags, self.\_adapters, sigma\_arr):  
 adapter.train\_svm(X, Y[:, tag.value], C\_coef, sigma\_coef)  
 print(**"Adapter {} has been fitting"**.format(tag))  
  
 def check\_train\_adapters(self, Xval, Yval):  
 for i, adapter in enumerate(self.\_adapters):  
 current\_precision = adapter.check\_validation\_svm(Xval, Yval[:, self.\_tags[i].value])  
 if self.\_precision\_vector[i] < current\_precision:  
 self.\_precision\_vector[i] = current\_precision  
 self.\_c\_vector[i] = adapter.C\_coef  
 self.\_sigma\_vector[i] = adapter.sigma\_coef  
  
 def predict\_article(self, text, dicts):  
 answer = ArticleTagsEnum.all  
 for tag, adapter, current\_dict in zip(self.\_tags, self.\_adapters, dicts):  
 x = self.\_text\_processor.process\_text(text, current\_dict)  
 if adapter.predict(x) == 1:  
 answer = tag  
 return answer  
  
 def save\_all\_states(self):  
 for adapter in self.\_adapters:  
 adapter.save\_svm\_state()  
  
 def load\_all\_svm\_states(self):  
 for adapter in self.\_adapters:  
 adapter.load\_svm\_state()  
  
 def load\_add\_svm\_dicts(self):  
 return [adapter.load\_dictionary() for adapter in self.\_adapters]  
  
 def create\_tables(self):  
 for tag in self.\_tags:  
 self.\_database\_manager.create\_table(tag)  
  
 @staticmethod  
 def add\_rows(matrix, new\_matrix):  
 return new\_matrix if matrix is None else np.vstack([matrix, new\_matrix])

Модуль SvmAdapter:

class SvmAdapter:  
 \_svm = None  
 \_current\_tag = ArticleTagsEnum.all  
  
 C\_coef = 0  
 sigma\_coef = 0  
  
 def \_\_init\_\_(self, tag):  
 self.\_current\_tag = tag  
  
 def train\_svm(self, x, y, C\_coef, sigma\_coef):  
 self.C\_coef = C\_coef  
 self.sigma\_coef = sigma\_coef  
 self.\_svm = svm.SVC(C=C\_coef, gamma=sigma\_coef)  
 self.\_svm.fit(x, y)  
 return self.\_svm  
  
 def check\_validation\_svm(self, x, y):  
 y\_predict = self.\_svm.predict(x)  
 good = 0  
 for i, predict in enumerate(y\_predict):  
 if predict == y[i]:  
 good += 1  
 print(**"{} SVM Predict: "**.format(self.\_current\_tag), good, **"Total: "**, len(y), **"{}%"**.format(good / len(y) \* 100.0))  
 return good / len(y) \* 100  
  
 def predict(self, x):  
 return self.\_svm.predict(x)  
  
 def get\_label\_matrix(self, matrix, labels\_count):  
 label\_matrix = np.zeros((len(matrix), labels\_count))  
 label\_matrix[:, self.\_current\_tag.value] = 1  
 return label\_matrix  
  
 def save\_svm\_state(self):  
 joblib.dump(self.\_svm, self.get\_cache\_path(self.\_current\_tag))  
  
 def save\_dictionary(self, dictionary):  
 joblib.dump(dictionary, self.get\_dictionary\_path(self.\_current\_tag))  
  
 def load\_svm\_state(self):  
 self.\_svm = joblib.load(self.get\_cache\_path(self.\_current\_tag))  
  
 def load\_dictionary(self):  
 return joblib.load(self.get\_dictionary\_path(self.\_current\_tag))  
  
 @staticmethod  
 def get\_dictionary\_path(tag):  
 return os.path.join(os.path.dirname(os.path.realpath(\_\_file\_\_)), **"CacheModels"**, **"Dictionaries"**, **"{}\_dict.plk"**.format(tag.name))  
  
 @staticmethod  
 def get\_cache\_path(tag):  
 return os.path.join(os.path.dirname(os.path.realpath(\_\_file\_\_)), **"CacheModels"**, **"{}.pkl"**.format(tag.name))

Модуль TextProcessor:

class TextProcessor:  
 \_html\_regs = re.compile(**'<.\*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});'**)  
 \_url\_regs = re.compile(  
 **'https?:\/\/(www\.)?[-a-zA-Z0-9@:%.\_\+~#=]{1,256}\.[a-zA-Z0-9()]{1,6}**\b**([-a-zA-Z0-9()@:%\_\+.~#?&//=]\*)'**)  
 \_email\_regs = re.compile(**'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'**)  
 \_number\_regs = re.compile(**'[0-9.]+'**)  
 \_text\_regs = re.compile(**'[#&%+:;\[\]/|><=`()@,**\'\"**!?\-}{\*\_\-®•“’”…‘–]'**)  
  
 \_porter\_stemmer = None  
 \_stop\_worlds = None  
  
 current\_counter = collections.Counter()  
  
 def \_\_init\_\_(self):  
 *#nltk.download('punkt')  
 #nltk.download('stopwords')* self.\_porter\_stemmer = PorterStemmer()  
 self.\_stop\_worlds = set(stopwords.words(**'english'**))  
  
 def process\_articles(self, articles, vector\_size, dictionary=None):  
 self.current\_counter.clear()  
 matrix = [self.\_convert\_article(article) for article in articles]  
 current\_dict = [t[0] for t in self.current\_counter.most\_common(vector\_size)] if dictionary is None else dictionary  
 return np.array([self.\_get\_features\_vector(row, current\_dict) for row in matrix]), current\_dict  
  
 def process\_text(self, text, dictionary):  
 return self.\_get\_features\_vector(self.\_convert\_article(text), dictionary).reshape(1, -1)  
  
 @staticmethod  
 def \_get\_features\_vector(words, dictionary):  
 vector = np.zeros(len(dictionary))  
 for i, key in enumerate(dictionary):  
 if key in words:  
 vector[i] = 1  
 return vector  
  
 def \_convert\_article(self, article):  
 article = re.sub(self.\_html\_regs, **''**, article.lower())  
 article = re.sub(self.\_url\_regs, **'httpaddr'**, article)  
 article = re.sub(self.\_email\_regs, **'emailaddr'**, article)  
 article = re.sub(self.\_number\_regs, **' number '**, article)  
 article = article.replace(**'$'**, **'dollar '**)  
 article = re.sub(self.\_text\_regs, **''**, article)  
  
 filtered\_worlds = [w for w in nltk.word\_tokenize(article) if w not in self.\_stop\_worlds]  
  
 for i, word in enumerate(filtered\_worlds):  
 filtered\_worlds[i] = self.\_porter\_stemmer.stem(word)  
 self.current\_counter[filtered\_worlds[i]] += 1  
  
 return filtered\_worlds