НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Факультет компьютерных наук
Образовательная программа
09.03.04 Программная инженерия
Курсовая работа

# Программа прогнозирования интереса к тексту по анализу его содержания

Выполнил студент группы БПИ-198
Мелехин Денис Антонович
Научный руководитель:
Старший преподаватель департамента программной инженерии факультета компьютерных наук Пантюхин Д.В.

# ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

## Цель работы

Написать программу для прогнозирования интереса к тексту по анализу его содержания.

## Задачи работы

1. Найти датасет (набор данных), который будет содержать информативные (интересные) и не информативные (не интересные) тексты
2. Преобразовать данные датасета в более удобный вид, удалить “мусор”.
3. Обучить модели на полученных данных
4. Протестировать модели
5. Написать интерфейс для демонстрации работы моделей

# Функциональные требования программы демонстрации работы моделей

**Список требований**

1. Загрузка входных данных
2. Оценка информативности текста
3. Отображение оценки
4. Отображение справки о программе

# Описание датасета

Выбранный датасет представляет собой набор из 33000 писем. Каждое письмо принадлежит одной из двух категорий: спам (не информативный текст) или не спам (информативный текст). В датасете знаки препинания были выделены пробелами. Датасет доступен по ссылке: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html
Примеры текста из датасета:

3323.2005-06-30.SA_and_HP.spam.txt

```
Subject: save your money by getting an oem software !
need in software for your pc ? just visit our site , we might have what you need . . .
best regards ,
fallon
```

0183.2000-01-11.kaminski.ham.txt

```
Subject: congratulations !
congratulations on your promotion to managing director ! it ' s great that your
hard work and dedication to enron have been recognized .
sherry
```

0378.2004-08-28.BG.spam.txt

```
Subject: for gallegos ' s shop customers !
dear customer !
we updated our programs list , and now we offer you more new software items
visit our full catalog and check new software titles here :
with best regards ,
product manager
emory arnold
```

# Описание моделей

## Наивный Байесовский классификатор:

Это простой вероятностный классификатор, основанный на применении теоремы Байеса. Несмотря на наивный вид и, несомненно, очень упрощенные условия, наивные Байесовские классификаторы часто дают прекрасные результаты для самых различных по сложности задач. Достоинством наивного Байесовского классификатора является малое количество данных для обучения.

## LSTM нейросеть:

Долгая краткосрочная память(Long short-term memory, LSTM) - разновидность архитектуры рекуррентных нейронных сетей, предложенная в 1997 году Зеппом Хохрайтером и Юрген Шмидхубером. В отличие от традиционных нейронных сетей, LSTM сеть хорошо приспособлена к обучению на задачах классификации.

# Описание наивного Байесовского классификатора

Наивный Байесовский классификатор работает по следующему алгоритму алгоритму:

1) С помощью метода os.walk(path) происходит обход по всем файлам. Каждый файл представляется в виде массива слов (массив элементов типа string). Далее датасет разбивается на 2 части - тренировочную и тестовую. В тренировочную часть вошло 85% датасета, а в тестовую - 15%
2) Создается 2 словаря. В обоих из них ключ - это слово, а значение, в первом - количество упоминаний этого слова в интересных текстах, а во втором - количество упоминаний этого слова в не интересных текстах.
3) Далее, используя формулу Байеса (Рисунок 1)

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}.$$

Рисунок 1

Программа считает вероятность того, что данный текст информативный.
B = набор слов bi, 1 <= i <= n
P(A|B) = вероятность, что данный текст информативный;
P(bi|A) = вероятность, что текст содержащий слова bi информативный
P(B|A) = P(b1|A) * P(b2|A) * … * P(bn|A)
P(A) = вероятность что текст информативный (вне зависимости от его текста). Т.е. кол-во информативных текстов / кол-во всех текстов.
Вероятность, что данный текст не информативный считается аналогично.

# Описание наивного Байесовского классификатора (продолжение)

Если вероятность того, что данный текст информативный, больше, то программа определяет данный текст информативным, иначе не информативным.
Значение P(B|A), при наличии большого количества слов в тексте, становится настолько близким к нулю, что при вычислениях теряется точность. А так как для определения является текст информативным или не информативным программа только сравнивает вероятности, то можно P(B|A) считать по такой формуле:

$$P(B|A) = \ln(P(b1|A)) + \ln(P(b2|A)) + \ldots + \ln(P(bn|A))$$

Где ln – натуральный логарифм. Так как для логарифма выполняется свойство

$$\ln(a*b) = \ln(a) + \ln(b)$$

и, так как логарифм является непрерывной функцией, то для сравнения мы можем использовать формулу с логарифмом.
Далее программа для каждого текста из тестового набора данных определяет, является ли текст информативным и подсчитывает статистику правильных ответов. Затем выводит статистику в консоль и в файл result.txt.
При обучении и тестировании на оговоренном раннее датасете наивный байесовский классификатор достиг результата в 88.34% правильных ответов на тестовой части датасета. Более детальный результат:
Общий результат:  85.74%
Верно определено информативных текстов:  71.69%
Верно определено не информативных текстов:  99.71%

# Устройство LSTM сети

1) Предобработка входных данных
2) Embedding слой
3) LSTM слой
4) Слой линейного преобразования
5) Сигмоидальное преобразование

# Описание алгоритма LSTM сети

1) Предобработка входных данных

1.1) С помощью метода os.walk(path) происходит обход по всем файлам. Каждый файл представляется в виде массива слов (массив элементов типа string). Далее датасет разбивается на 2 части - тренировочную и тестовую. В тренировочную часть вошло 85% датасета, а в тестовую - 15%

1.2) Входные данные разбиваются на группы (батчи) по 80 текстов в каждой группе. Текста, которые остались (так как количество файлов не делится нацело на 80) не берутся в расчет и удаляются.

1.3) Составляется словарь всех слов из всего датасета, в котором ключ - слово, а значение - количество его упоминаний в датасете. Если в тестовом датасете появляется слово, не встречавшееся в тренировочной части датасета, то количество его упоминаний считается равным 1. После чего словарь сохраняется в файл в формата json. Это необходимо для ускорения работы программы при последующих запусках.

1.4) Каждое слово заменяется на соответствующее ему значение в словаре. В каждом тексте оставляются только первые 25 слов, так как нейросеть получает на вход тензор (массив с неизменяемой размерностью). Если в тексте не было 25 слов, то оно дополняется словами, так называемыми “padding”, их значение в словаре считается равным 0.

1.5) После преобразований всех текстов, входные данные сохраняются в файлы train_x.pt - файл с тренировочными данными, train_y.pt - файл с ответами тренировочной части, test_x.pt - файл с тестовыми данными, test_y.pt  - файл с ответами на тестовую часть. Это необходимо для ускорения работы программы при последующих запусках.

# Описание алгоритма LSTM сети (продолжение)

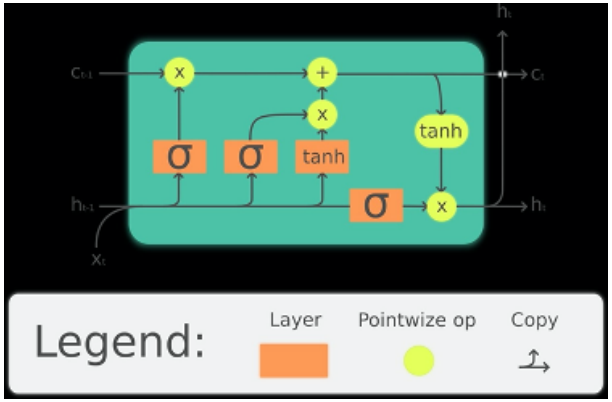
2) Embedding слой

Данный слой необходим для преобразования каждого слова в вектор. Он принимает на вход вектор, размером с словарь, где на месте значения слова в словаре стоит 1, а на остальных местах нули. Выход этого слоя состоит из 100 нейронов (embedding_dim), в каждом из которых будет некоторое значение. Вектор из этих выходных значений – это новое векторное представление слова в нейросети. Данный вектор получается для каждого слова и затем каждое слово заменяется этим вектором. Данный слой реализован в модуле torch.nn.Embedding.
Более подробное описание: https://pytorch.org/docs/stable/nn.html#torch.nn.Embedding

# Описание алгоритма LSTM сети (продолжение)

3) LSTM слой

Данный слой является основным в LSTM сети. Он реализован в модуле nn.lstm библиотеки “pytorch”. В программе слой представляет собой блок нейронов с рекуррентными связями. Структура нейрона в LSTM слое представлена на схеме:




Всего таких нейронов 25 в блоке (hidden_size).

На вход lstm слой получает выход из Embedding слоя размерностью <batch_size, min_words_in_sentence, embedding_dim>. В нашем случае это трехмерный вектор размером <80, 25, 100>.

Выход из lstm слоя имеет размерность <batch_size, min_words_in_sentence, hidden_size>. В нашем случае это трехмерный вектор размером <80, 25, 25>.

Более подробное описание принципов работы lstm слоя: https://pytorch.org/docs/stable/nn.html#torch.nn.Embedding

# Описание алгоритма LSTM сети (продолжение)

4) Слой линейного преобразования

Данный слой необходим для преобразования выхода lstm слоя. Он принимает выход последнего нейрона lstm слоя размерностью min_words_in_sentence, после чего преобразует данный вектор в скаляр.
Более подробное описание линейного слоя: https://pytorch.org/docs/stable/nn.html#linear

5) Сигмоидальное преобразование

Данный слой необходим для представления выходного скаляра из предыдущего слоя в вероятность принадлежности информативному или не информативному тексту. Функция, применяемая к каждому итоговому значению каждого батча:

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Более подробное описание в: https://pytorch.org/docs/stable/nn.html#sigmoid

# Обучение LSTM сети

Обучение производилось оптимизатором из модуля torch.optim.Adam. Обучение на входных данных в программе происходило 4 раза (количество эпох). Для подсчета функции ошибки использовалась бинарная кросс-энтропия (модуль nn.BCELoss). Для регулировки скорости использовался планировщик torch.optim.lr_scheduler.ReduceLROnPlateau.

Адам – это современный алгоритм, используемый в области глубокого обучения, который быстро достигает отличных результатов.
Подробное описание работы оптимайзера: https://pytorch.org/docs/stable/optim.html#torch.optim.Adam

Функция бинарной кросс-энтропии используется для подсчета ошибки в задачах, в которых необходимо распределить входные данные на 2 категории. Данная функция принимает на вход два числа из отрезка [0;1] и возвращает число из этого же отрезка.
Подробное описание функции ошибки: https://pytorch.org/docs/stable/nn.html#bceloss

Планировщик в нейронной сети необходим для установки более подходящей скорости обучения для каждой эпохи. Он позволяет нейросети обучаться медленнее в конце обучения.
Подробное описание torch.optim.lr_scheduler.ReduceLROnPlateau: https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau

Результаты тестирования на тестовом датасете LSTM нейросети:
На тестовом датасете распознано верно 97.81%
На тестовом датасете информативных текстов верно распознано 97.74%
На тестовом датасете не информативных текстов верно распознано 97.87%

# Описание программы демонстрации работы нейросети

Для демонстрации работы было создано отдельное desktop приложение. Данная программа была реализована на языке C#. Интерфейс программы реализован на windows forms. Модуль анализа данных размещен на сервере.

Алгоритм функционирования программы

1) Получение входных данных от пользователя

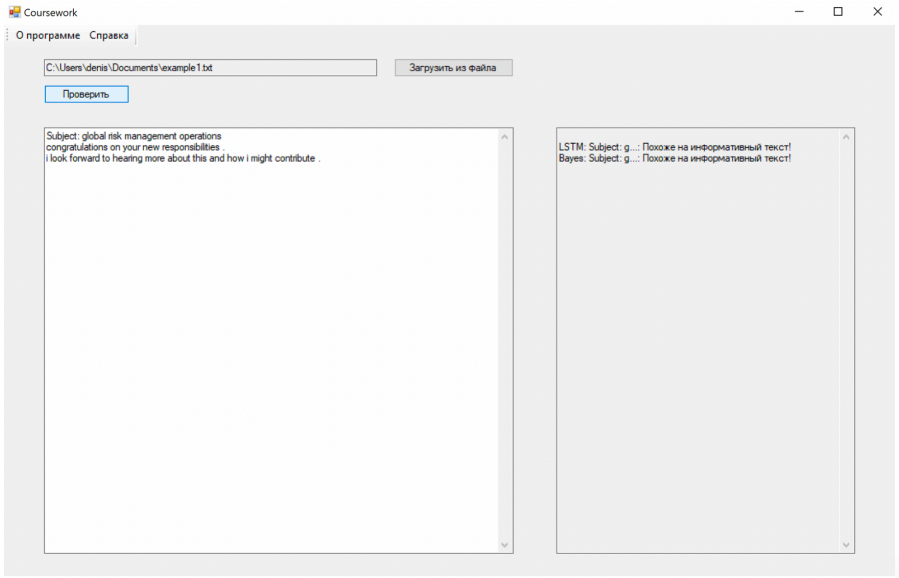
Программа позволяет пользователю ввести входные данные двумя способами: в текстовом поле или выбрать файл с разрешением .txt. Текст следует вводить на английском языке, так как модели были обучены на тренировочных данных, тексты которых на английском языке.

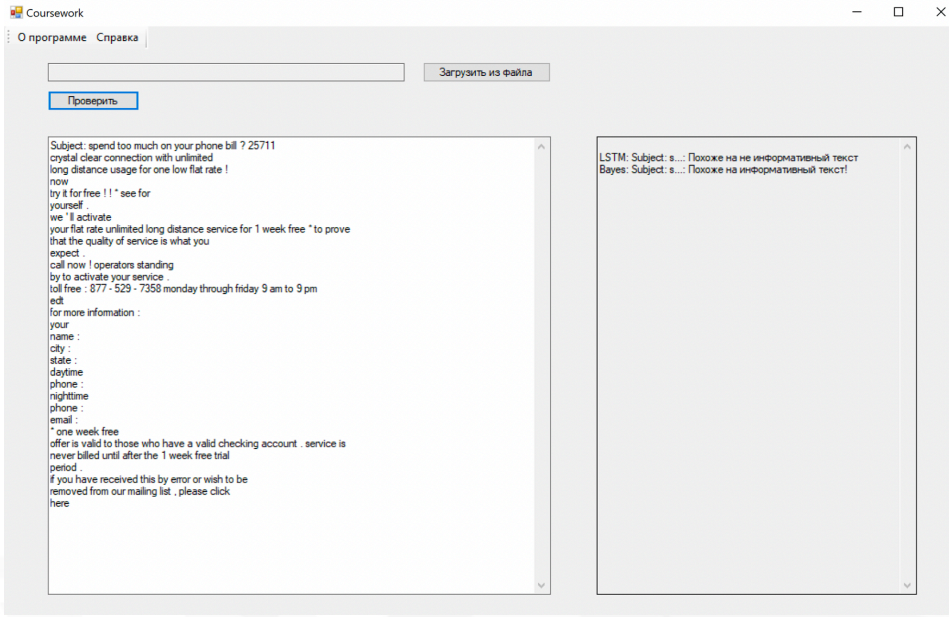
2) После получения входных данных программа отправляет данные на сервер. Модуль, принимающий POST запрос, находится по ссылке: http://denis.hiweb.ru/cgi-bin/SpamClassifier.py

Реализация POST запроса производилась с помощью библиотеки WebRequest, а получение ответ с помощью библиотеки WebResponse.

3) После получения ответа от сервера в поле для выходных данных выводится ответ от сервера или сообщение об ошибке, если такая произошла при обращении к серверу.

# Демонстрация работы программы



# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1.Модуль Pytorch [Электронный ресурс]// URL: https://pytorch.org (Дата обращения: 09.05.2020, режим доступа: свободный).
2.Word embeddings tutorial [Электронный ресурс]// URL: https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html (Дата обращения: 09.05.2020, режим доступа: свободный).
3.Spam-Ham Classification [Электронный ресурс]// URL: https://medium.com/analytics-vidhya/spam-ham-classification-using-lstm-in-pytorch-950daec94a7c (Дата обращения: 09.05.2020, режим доступа: свободный).
4.Документация языка C# [Электронный ресурс]// URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (Дата обращения: 09.05.2020, режим доступа: свободный).
5. Модуль json [Электронный ресурс]// URL: https://docs.python.org/3/library/json.html (Дата обращения: 09.05.2020, режим доступа: свободный).
6.Python [Электронный ресурс]// URL: https://www.python.org (Дата обращения: 09.05.2020, режим доступа: свободный).
7.Модель Embeddings [Электронный ресурс]// URL: https://pytorch.org/docs/stable/nn.html#torch.nn.Embedding (Дата обращения: 09.05.2020, режим доступа: свободный).
8.Модель LSTM [Электронный ресурс]// URL: https://habr.com/ru/company/wunderfund/blog/331310/ (Дата обращения: 09.05.2020, режим доступа: свободный).
9.Описание линейного слоя [Электронный ресурс]// URL: https://pytorch.org/docs/stable/nn.html#linear (Дата обращения: 09.05.2020, режим доступа: свободный).
10.Описание сигмоидального слоя [Электронный ресурс]// URL: https://pytorch.org/docs/stable/nn.html#sigmoid (Дата обращения: 09.05.2020, режим доступа: свободный).
11.Описание оптимизатора [Электронный ресурс]// URL: https://pytorch.org/docs/stable/optim.html#torch.optim.Adam (Дата обращения: 09.05.2020, режим доступа: свободный).
12.Описание функции ошибки BCELoss [Электронный ресурс]// URL: https://pytorch.org/docs/stable/nn.html#bceloss (Дата обращения: 09.05.2020, режим доступа: свободный).
13.Описание sheduler [Электронный ресурс]// URL: https://pytorch.org/docs/stable/optim.html#torch.optim.lr_scheduler.ReduceLROnPlateau (Дата обращения: 09.05.2020, режим доступа: свободный).
14.Требования .Net Framework [Электронный ресурс]// URL: https://docs.microsoft.com/ru-ru/dotnet/framework/get-started/system-requirements (Дата обращения: 09.05.2020, режим доступа: свободный).

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

# Спасибо за внимание!

Мелехин Денис Антонович
damelekhin@edu.hse.ru

Москва - 2020