Алгоритм для конкурса Avito.ru-2014

Евгений Нижибицкий
nizhibitsky@ya.ru

Аннотация

Данный документ описыват подход к решению задачи поиска контактной информации на изображениях. Главную роль в итоговом алгоритме играет логистический классификатор, построенный на признаках, выделенных из нескольких последних fully-connected слоев глубинной сети, аналогичной AlexNet. Небольшой вклад также дает поиск паттернов в текстах, найденных на основе движка OCR Tesseract. Большую часть времени работы алгоритма при этом занимает именно предварительная обработка изображений и выделение признаков (около 2-х часов на обработку и около 2-х часов для каждой из групп признаков).

Предварительная обработка данных

Во избежание ложно-положительных срабатываний tesseract'а на домене-логотипе, на соответствующем участке изображения предварительно производится обработка медианным фильтром.

Генерация и отбор признаков

В качестве данных для обучения выступают тексты, полученные на основе распознавания с помощью tesseract, и предварительно с'max-poll'енные фичи из 6-го и 7-го fully-connected слоев предобученной глубинной сети bvlc_reference_caffenet в составе библиотеки Caffe.

На основе текстовой информации производится поиск групп патернов:

1. номера телефонов: (XX, XX), -XX, XXX + ['8(', '8 (', '(8', '8-8', '+79'];
2. части слов из корпусов с 5000-ю словами русского и английского языков;
3. паттерны, придуманные вручную: urls = ['.ru', 'vk.', 'http', ':/'], emails = ['@ma', 'e-ma'], words = [u'монта', u'тел.', u'звонит', u'кред', u'автом', u'адрес', u'конт', u'окон', u'слуг', u'недв'].

На основе fully-connected слоев сети размерности 4096x10 каждый после макспулинга по второй размерности получаем один вектор длины 8192.

Процесс обучения

Формально процесс обучения заключается в настройке логистической регрессии на 8192 caffenet-признаках. Затем подбираются различные линейные комбинации полученного ответа и вектора частот обнаруженных паттернов с предварительным эвристическим преобразованием последнего.

Описание программы алгоритма

Итоговый проект для конкурса состоит из 3-х ipython-ноутбуков:

1. pillow.ipynb — отвечает за предварительное замазывание медианным фильтром логотипа Компании, для работы достаточно первых трех Cell'ов. Содержит также старые куски прошлых экспериментов, которые более не актуальны.

2. ocr.ipynb — отвечает за поиск паттернов в текстах, найденных с помощью tesseract. В начале, после стандартных импортов и загрузки данных производится сравнительная демонстрация работы tesseract и cuneiform, затем производится выделение признаков в выборках A и B за 80 и 40 минут соответственно (по некоторым причинам второй запуск производился на 2-х ядрах из 4-х). После распознавания текста производится поиск «дискриминативных» паттернов — для каждой подгруппы при поиске задается порог частоты встречаемости, и условие, что с данным паттерном все объявления оказались заблокированными. Далее в ноутбуке производятся различные выкладки по тому, насколько вообще эффективно эти самые паттерны искать и краткий итог этого — одних паттернов нам не достаточно. В последнем cell'е производится сохранение паттернов.

3. caffe.ipynb — содержит собственно саму суть алгоритма. После загрузки преднастроенной глубинной сети производится выделение признаков из последних двух fully-connected слоев после пулинга и rectified linear'a. На основе выделенных признаков строится логистическая регрессия, для которой подбирается гиперпараметр на основе простого наблюдения изменений качества на обучении и двух половинах валидации. Лучший алгоритм использует параметр **C=0.000035**. Затем на основе предсказанных регрессией вероятностей производится смешение с вектором частот срабатываний паттерн-матчинга на tesseract-паттернах. Наилучшее качество при этом показывает преобразование вида

$$y_{pred} = \alpha \cdot y_{caffe} + (1 - \alpha) \cdot \tanh(0.25 \cdot freq_{tesseract}), \alpha = 0.5,$$

которое добавляет еще пару-тройку тысячных к AUC-ам:

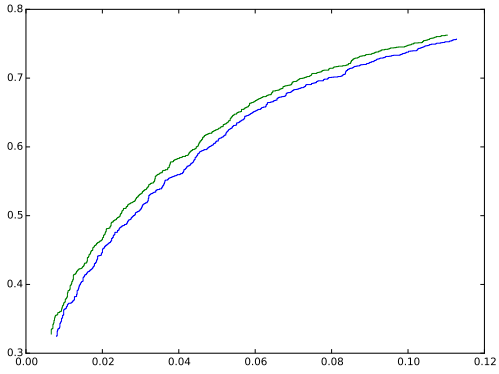

Рис. 1: Участок ROC-кривой (зеленому соответствуют обогащенные ответы).

Описание рабочей среды

Для проведения экспериментов использовался ноубук ASUS N550JV с 4-х-ядерным процессором i7 4700HQ и 8 ГБ памяти с ОС Ubuntu 14.04.1. «Холодный прогон» алгоритма на двух выданных выборках занимает в среднем чуть больше 6-ти часов при использовании многопоточности и GPU.

Зависимости

- Как минимум 8 ГБ памяти — при выделении признаков на всех объектах обучения использование памяти доходит до 7 ГБ.
- (Для Ubuntu 14.04.1) Пакеты tesseract-ocr, tesseract-eng, tesseract-rus. Cuda SDK 6.5, установленный по инструкции с официального сайта NVIDIA — установка производится в итоге через пакетный менеджер и не мусорит. Теоретически можно обойтись без CUDA — в caffe.ipynb тогда следует заменить net.set_mode_gpu() на net.set_mode_cpu() — на моем ноутбуке это увеличивает длительность выделения признаков в 3–5 раз.
- Библиотеки Python (использовался дистрибутив Anaconda для версии 2.7): ipython, numpy, pandas, matplotlib, seaborn, sklearn, Pillow, pyocr (из git), caffe (из git), и зависимости, из них вытекающие.

Инструкция по воспроизведению

Для воспроизведения достаточно пошагово выполнять клетки каждого из ноутбуков, раскомменчивая/закомменчивая код, который относится к сериализации/десериализации — к примеру, после прогона tesseract'а по всей выборке и сериализации стоит закомментить выделение текста и сериализацию, рестартнуть ядро ноутбука, и в следующий раз начать с десериализации. То же относится к другим емким по памяти процедурам во избежание утечек. При этом, естественно, стоит изменить пути к данным, а также указать путь до корня форка Caffe в caffe.ipynb.

При возникновении трудностей на каком-то этапе, можно воспользваться готовыми pickle-дампами, которые временно выложены на Яндекс.Диск (ссылка ниже), а также связаться с автором.

Ссылки

- Pickle-дампы
- Anaconda Scientific Python Distribution
- Caffe classification/layer visualization notebook
- Caffe GitHub
- PyOCR GitHub
- NVIDIA CUDA Getting Started Guide for Linux