Задача двухклассовой классификации изображений

Гущин А. Е.

Аннотация

В работе решается задача двухклассовой классификации изображений. К первому классу изображений относятся те изображения, которые содержат запрещенную к размещению на них информацию (номера телефонов, адреса сайтов). Задача разбивается на два этапа - локализации текста на изображении и определении, содержит ли текст запрещенную информацию. Для нахождения символов на изображении производится локализация текста с помощью алгоритма Stroke Width Transform. Для распознавания символов используется OCR (Optical Recognition System) Tesseract. Качество работы алгоритма оценивается с помощью Area Under Curve.

Ключевые слова: computer vision, Stroke Width Transform, text localizaton, Optical Charaster Recognition, Tesseract, text recognition

Репозиторий с кодом: https://github.com/aguschin/2014ImageClassificationTraining

1 Постановка задачи

Дан набор изображений с реальными фотографиями, на которых может присутствовать текст. Некоторый текст было запрещено размещать на изображениях, вошедших в этот набор (номера телефонов, адреса сайтов, e-mail). Изначально задача ставится следующим образом: необходимо определить, на каких изображениях присутствует запрещенный к размещению текст.

Предложим один из подходов к решению задачи. Для этого переформулируем задачу в терминах компьютерного зрения и машинного обучения. Вначале нужно выявить все положения текста на этих фотографиях, распознать текст и определить, содержит ли распознанный текст запрещенную информацию. На основе полученных в процессе работы с фотографией признаков (которыми являются, например, количество локализованных областей текста, распознанный текст, наличие запрещенной информации в нём) построить двухклассовый классификатор, определяющий, содержит ли изображение запрещенный текст.

Пусть у нас есть цветное изображение. Рассмотрим изображение I как трехмерную матрицу, в которой первые два индекса соответствуют координатам пикселя на изображении, а третий индекс — цвету в RGB-системе (цвет для каждой компоненты (Red,Green,Blue) кодируется числом от 0 до 255, где 255 соответствует присутствию соответствующего цвета, 0 — его отсутствию, а остальные числа – градациям этого цвета).

1.1 Предобработка

Чтобы улучшить результаты алгоритма, осуществляется следующая предобработка:

- логотип 'Авито', присутствующий в правом нижнем углу каждой фотографии из имеющегося набора, заменяется прямоугольником. Для этого можно выполнить в терминале команду

  mogrify -rotate 180 -draw 'rectangle 19,5 101,18' -rotate 180 -fill gray -quality 92 *.*

- каждая фотография увеличивается в два раза. Это связано с тем, что фотографии в наборе имеют небольшое разрешение, в основном 640x480, а Stroke Width Transform, описанный ниже, обрабатывает каждый пиксель в отдельности, и таким образом увеличение изображения и небольшое размытие улучшают качество его работы. Для этого можно выполнить в терминале команду

  mogrify -resize 200% -quality 92 *.*

1.2 Локализация текста

Первым этапом алгоритма является применение Stroke Width Transform [1,2] для локализации текста. В основе этого метода лежит то, что блок текста на изображении выделяет то, что он имеет практически одинаковую ширину линии (stroke width). Основной идеей является способ посчитать ширину линии для каждого пикселя на изображении и сгруппировать области с похожими значениями ширины линий. Рассмотрим этот способ подробнее.

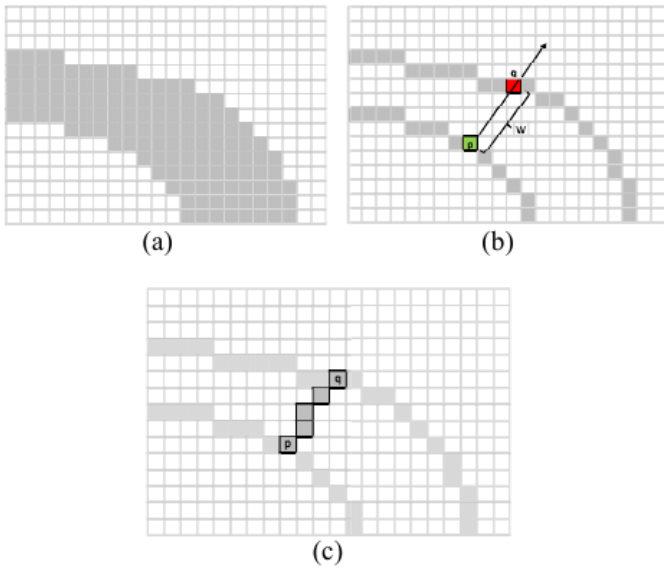

Figure 3: Implementation of the SWT. (a) A typical stroke. The pixels of the stroke in this example are darker than the background pixels. (b) p is a pixel on the boundary of the stroke. Searching in the direction of the gradient at p, leads to finding q, the corresponding pixel on the other side of the stroke. (c) Each pixel along the ray is assigned by the minimum of its current value and the found width of the stroke.

SWT является локальным оператором, который высчитывает наиболее вероятную ширину линии для каждого пикселя. Результатом работы этого оператора является изображение такого же размера, как изначальное, каждому пикселю которого присвоено найденное значение ширины линии. Линия определяется как непрерывная часть изображения, которая имеет форму полоски с приблизительной одинаковой шириной.

Для начала применим фильтр Canny [3], чтобы выделить границы объектов на изображениях. В результате мы получаем черно-белое изображение. Присвоим каждому пикселю результирующего изображения SWT значение ∞. Затем найдем градиент d_p для каждой граничной точки p и будем двигаться по лучу $r = p + d_p * n, n > 0$ до тех пор, пока не найдем другую граничную точку q. Если градиент d_q в точке q приблизительно противоположен d_p ($d_q = d_p \pm pi/6$), то каждому элементу s на отрезке $[p, q]$ присвоим значение $\|p - q\|$, если только s ещё не было присвоено меньшее значение. Если q не найдено, мы забываем про этот луч.

На следующем этапе мы группируем точки с приблизительно равными значениями. Для этого используется модификация алгоритма поиска компонент связности с модификацией условия объединения точек.

Чтобы отсеять области, которые могут быть ошибочно приняты за текст, применяется несколько эвристик:

- Для каждой связной компоненты вычисляется разброс значений ширины линии и если он достаточно велик, компонента отбрасывается.
- Если отношение высоты к ширине связной компоненты менее 0.1 или более 10, компонента отбрасывается.

Следующим шагом является объединение соседних найденных компонент связности в линии, в предположении, что они являются буквами в одном слове. Этот шаг позволяет улучшить алгоритм, так как отдельные буквы появляются на изображениях гораздо реже, чем целые слова. Для объединения необходимо, чтобы соседние компоненты имели близкие по значению ширины линий и отличались по высоте не более чем в два раза (это связано с наличием строчных и прописных букв). После обнаружения всех таких пар происходит их объединение в цепочки. Две цепочки объединяются в одну, если они имеют одинаковый конец и похожее направление.

В конце концов алгоритм выполняется и для луча $r = p - d_p * n, n > 0$ (это связано с тем, что для белых букв на черном фоне градиент будет направлен в обратную сторону, нежели для для черных букв на белом фоне).

1.3 Распознавание текста

Следующим этапом будет являться распознавание текста в областях, найденных на предыдущем этапе. Для этого воспользуемся OCR Tesseract [4].

Для всех фотографий разобъем распознанные тексты на всевозможные сочетания из трех букв , идующих подряд ($trigram$). Для каждого такого сочетания посчитаем:

1. сколько раз оно было распознано на всех фотографиях обучающей выборки
2. сколько раз оно было распознано на всех фотографиях обучающей выборки, содержащих запрещенную к размещению информацию
3. отношение значения 2 к значению 1: $probability$

После этого построим новую фичу $Feature$ для классикатора, используя идею статистической меры TF-IDF : для каждой фотографии вычислим сумму всех $probability$ для каждого $trigram$ в распознанном тексте и разделим на количество $trigram$. Эвристические соображения, приводящие к такой фиче: $probability$ - это вычисленная на обучающей выборке 'вероятность' того, что данный триграм принадлежит фотографии, содержащей запрещенную к размещению информацию. Соответственно, $Feature$ имеет смысл средней по всем $trigram$ "вероятности".

2 Описание признаков классификатора

Для классификатора использованы следующие признаки:

- **Области**, N: Количество областей, обнаруженных SWT.

- **Площадь областей**: S_{sum}, суммарная площадь областей, обнаруженных SWT.
- **Фича, полученная после распознавания текста**, $Feature$: средняя по всем $trigram$ "вероятность"того, что $trigram$ из этой фотографии относится к "запрещенной"фотографии.

Алгоритмом для классификатора выбран SVM с линейным ядром (LinearSVC).

3 Результаты

Результаты работы классификатора для кросс-валидации с nfold=5 :

LinearSVC на двух первых признаках : AUC: 0.759 (+/- 0.028)
LinearSVC на трех признаках : AUC: 0.809 (+/- 0.028)

Список литературы

[1] E. O. Epshtein Boris and Y. Wexler, “Detecting text in natural scenes with stroke width transform,” 2010.

[2] K. Saurav and A. Perrault, “Text detection on nokia n900 using stroke width transform,” 2010.

[3] J. Canny, “A computational approach to edge detection,” 1986.

[4] R. Smith, “An overview of the tesseract ocr engine,” 2007.