Robust PDF Table Locator

Nick Pether and Todd Macdonald

December 17, 2016

1 Introduction

Data scientists rely on an abundance of tabular data stored in easy-to-machine-read formats like .csv files. Unfortunately, most government records and scientific papers are published in Adobe PDF format. PDF documents are often formatted poorly, or only contain an embedded image of a scanned document[1]. While this isn't a problem for human readers, it makes automated data extraction close to impossible, meaning those who want to use this data must endure the tedious horror of re-entering it by hand. This makes solving the problem of extracting data from PDFs using only computer vision one of immense value to governments, businesses and researchers. Towards this end, we built a computer vision classifier to label any tabular data in a PDF.

The input to our algorithm is a JPEG image of a single page from a PDF file. We preprocess the image to show only the outline of distinct "blocks" of text and split it into partial images based on where characters are clustered via k-means. We then take the Histogram of Gradients (HoG) feature descriptors of the resulting images and classify them as partial tables or not using an SVM binary classifier. Finally we take a bounding rectangle around all overlapping 'partial tables' as the coordinates of a table.

2 Dataset and Preprocessing

Since we could not find any pre-existing dataset of images of tables vs. not tables, we compiled, preprocessed and labeled our own dataset via the following process:

2.1 Scraping Financial documents

The PDFs we used as our inputs were 100 Comprehensive Annual Financial Statements (CACS) for various local governments scraped from the California Common Sense Open Records Initiative[2]. We split each PDF into one JPEG file per page.

2.2 Finding potential partial tables as classifier inputs

Because it would be too computationally expensive to classify every possible crop of a given page as a table or not, we narrowed down our search space to include clusters of text. First, we approximated the location of every character on the page by searching for any distinct area of solid color within a certain size using the OpenCV library[3]. We then took 5500 image crops of 'clusters' of characters (see section 4.1), hand labelled them and saved their pixel coordinates on the original page image.

Figure 1: Original image

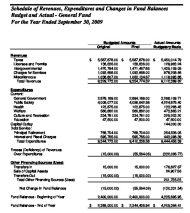


Figure 2: Characters located

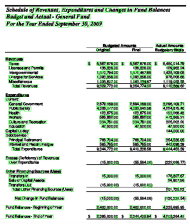
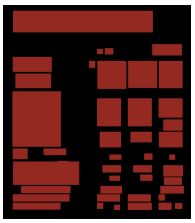


Figure 3: Redacted

Figure 4: Filtered

Figure 5: Final text block image



2.3 Preprocessing images by blocking out text

We realised that visually, the distinguishing feature of tables as compared to paragraph text was that it was arranged in an array-like structure. To further simplify the inputs for our classifier, we "redacted" our page images. Using openCV we took the character boxes(fig. 1), filled them in (fig 2), applied a color filter to obtain only the outline of the redacted text (fig 3) and applied the same methods used to find the character boxes to simplify our approximation of the shape of text blocks (fig 4). We then took the coordinates of the potential 'partial tables' and used them to extract cropped images from our processed page. Finally, we converted to grayscale and resized all the redacted partial images to size 64x64 and took feature descriptors from those images as our SVM inputs.

2.4 Extracting HoG features

Our actual input to the SVM classifier was the Histogram of Oriented Gradients (HoG) feature descriptor of each 64x64 redacted partial image[4]. A HoG descriptor is a way of representing a 2-dimensional image as a single feature vector, such that the same object (a table) produces near identical feature vectors. At a high level, HoG descriptors convey information about the location of edges and corners, which we though was adequate to represent the basic shapes of text blocks.

HoG feature extraction works as follows: Each 64x64 image is divided into overlapping 8x8 pixel cells. For each pixel we take its gradient vector, a 2-d vector of the differences between the pixel values of its neighboring pixels on the x and y plane. This vector is perpendicular to actual edges in the image, meaning that these vectors are together adequate information to tell whether a cell contains any edges or corners. These gradients are collected in a histogram, where a gradient's column is decided by which discrete bin its angle falls into, and its contribution to that column's value is its magnitude. Thus the contents of a 64 pixel cell are represented by only a 9-valued vector. Since there are 64 cells in a 64x64 image, we end up with a 576 valued vector as our final feature vector and input for our SVMs.

3 Methodology

As mentioned above, our approach to locating tables consisted of two distinct machine learning steps. They were:

1. Run k-means on the locations of the characters to generate bounding boxes that might contain a full or partial table.
2.Identify the bounding boxes that contain tables using an SVM classifier on HoG descriptors of the image crops from step 1.

3.1 Identifying Candidate Bounding Boxes Using K-Means

The purpose of this section was to find good 'candidate' crops of page images that might contain a table or partial table, so as to avoid exhaustively classifying every possible crop of a page. To find our 'candidates' we clustered the characters on a page using k-means then took a bounding box around all characters within each cluster. An identical bounding box was cropped from the preprocessed page image (fig 5) and HoG features were extracted from the resulting image.

The criteria for success here, qualitatively, is that all tables should be covered in their entirety by some combination of overlapping cluster bounding boxes, in order that all tables would be contained by some subset of 'candidate' tables.

As we have discussed in lecture[5], the k-means approach is as follows, for clustering points of dimension n:

1. Generate k centroids, where each centroid u_i initialized randomly in $\mathbb{R}^n$.
2. Assign each point p_i to the closest centroid u_j by euclidean distance
3. Set centroid u_j's location to be the mean location of every point p_i assigned to it.
4. Repeat until convergence.

Since we did not want to assume how many tables might be on a given page, and anticipated that a table might be split across multiple clusters, we could not use k-means with a fixed k. We solved this problem by considering all values of k between 1 and 12 and using the clusters from the values of k that produced the smallest silhouette score.

The silhouette score is defined as follows: where a is the mean intra-cluster distance and b is the mean distance to the second nearest cluster:

$$\frac{(b-a)}{max(a,\, b)}$$

As for the input points to the k-means algorithm, we ran the algorithm using both the x,y coordinates of each character and just the y coordinate to find which produced better 'coverage' of the tables visually.

3.2 Image Classification With SVM's

The classifier we used on the partial images to identify which ones contained full or partial tables was a support vector machine (SVM[6]) using various different kernels provided by the

scikitlearn[7]library. As explained earlier, we used arrays representing the HoG descriptors of our images as our inputs.

For our loss function, we use the $L(x^{(i)}, y^{(i)}) = max(0, 1 - y^{(i)} * (\sum_{i=1}^{m} \alpha_i \times K(x^{(i)}, x)))$ described in class. λ is the regularization rate, and $(x^{(i)}, y^{(i)})$ is the ith training example. For the kernel $K(x, z)$, we use two different representations, a polynomial kernel and a linear kernel[8]. For the polynomial kernel, we define $K(x, z) = (x^T z + 1)^3$. For the linear kernel, also known as the Gram matrix, we define $K(x, z)_{ij} = x_i^T z_j$. Since regularization factor λ corrects against heavily weighted features, we use it in our cross validation step to minimize overfitting.

4 Results and Analysis

4.1 K-Clustering using only Y-values produces better table coverage

We found that clustering characters in one dimension based on their y-coordinates alone produced much better 'coverage' of tables visually than using 2-d clustering using their (x,y) coordinates. In the latter case, visual examination showed that tables tended to be split across multiple non-overlapping boxes (see fig), meaning they could not be useful for reconstructing the original tables. Based on these qualitative results we chose the bounding boxes created by clustering along the y-axis to create the images for our classifier.

Figure 6: (x,y) clustering - tables tend to be split over non-overlapping clusters

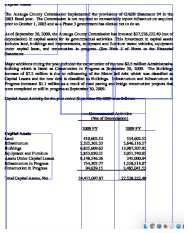
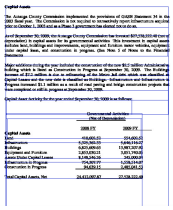


Figure 7: (y) clustering produces better coverage

4.2 Cross Validation and Hyperparameter Tuning

For our SVM, we employed a more quantitative approach to optimization, since the SVM is a supervised learning algorithm. First, we divided our labeled images into a training and testing set, where the testing set composed 30 percent of the overall images. To tune the hyperparameters involved in an SVM, such as the regularization constant and kernel type, we used 10-fold cross validation and averaged the precision, recall, and F1 scores over all 10 iterations of the 10-fold algorithm. We define the F1 score as follows: $2 \times (\frac{1}{recall} + \frac{1}{precision})^{-1}$. From our data, we found the polynomial kernel to produce the best F1 score.



4.3 Table Classification

Through a combination of optimization and computer vision techniques in pre-processing the images, we were able to achieve a recall of 0.66, which is significantly higher than the baseline 0.55 recall for table identification acheived by Tabula[9], an open source PDF table extraction tool that uses parses correctly formatted PDFs rather than using computer

vision to find tables. Tabula's 0.99 precision, however, is much better than what our algorithm achieved. Even though we chose our regularization constant and kernel to maximize the F1 score in the cross-validation set, we see that our SVM classifies the training set much better than the testing set, which is normally indicative of overfitting. We concluded that using better descriptors with less features might solve this problem.

	Precision	Recall	F1
Train	0.98	0.88	0.93
Cross Validation	0.78	0.77	0.73
Test	0.80	0.66	0.73

Figure 8: Precision, Recall, and F1 values for the training, testing, and cross-validation sets of preprocessed images. The classifier is an SVM with a polynomial kernel. The values for cross validation are the averages for 10-fold cross validation.

4.4 Raw vs. Preprocessed Images

Independent of the kernel or regularization constant, we see that our pre-processing of the images significantly increased the recall and precision of our algorithm.

	Precision	Recall	F1
polynomial kernel	0.74	0.39	0.51
polynomial kernel with pre-processing	0.80	0.66	0.73
linear kernel	0.64	0.45	0.53
linear kernel with pre-processing	0.61	0.78	0.68

Figure 9: Precision, Recall, and F1 values on binary classification of table images from the test set.

5 Conclusions

Given the significant improvements made to our classifier's results using descriptors of preprocessed images showing the general shape of blocks of text vs those of the raw text images, we can conclude that simpler images are vastly easier to classify, and that similar preprocessing methods to ours could be of use to future object detection applications. We were able to get decent precision and recall better than our baseline, and since we used computer vision to do so it is likely these results would be replicable on scanned images where the baseline (Tabula) would be useless. Based on our results, further work using more sophisticated image descriptors and the polynomial kernel on our preprocessed images looks to be the most promising avenue for creating a more successful table detector. The logical next step in reading in tabular data from PDFs would be to apply optical character recognition technology to tables located by this application.

References

[1] Explanation of PDF format and its drawbacks,
https://www.propublica.org/nerds/item/
heart-of-nerd-darkness-why-dollars-for-docs-was-so-difficult

[2] Open Records Initiative Database
http://ori.cacs.org/

[3] OpenCV (Open source computer vision) Documentation
http://opencv.org/

[4] Histogram of Oriented Gradients
http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/

[5] k-means Clustering Algorithm
http://cs229.stanford.edu/notes/cs229-notes7a.pdf

[6] Support Vector Machine Algorithm
http://cs229.stanford.edu/notes/cs229-notes3.pdf

[7] SciKitLearn SVM documentation
http://scikit-learn.org/stable/modules/svm.html

[8] Kernel Explanation
https://www.cs.cmu.edu/ ggordon/SVMs/new-svms-and-kernels.pdf

[9] Tabula Website
http://tabula.technology/