Digital Camera Specification Extraction

Ethan Lozano
edlozano@stanford.edu

I: ABSTRACT

In this paper, I present a method for training a classifier to use for an Information Extraction (IE) system that extracts digital camera specifications from a document.

The scope of the methods I present are limited. First, my system has only been tested with digital camera documentation where the majority of the text is simply camera specifications. Second, my system was trained and tested on documentation where the majority of the documents describe a single camera. Finally, my methods are limited to extraction of numbers; more specifically, my system will not extract each number's unit.

II: ANNOTATED DATA

I produced my own annotated data. I used the numerical values in an existing database to automatically annotate each number in my corpus; this procedure is known in the literature as, 'Distant Supervision'.

I produced my own camera database by recreating the camera database of dpreview.com by scraping their website. The camera database has information on approximately 1700 cameras, with 70 unique attributes to describe each camera. Though, I only use 31 of the attributes that have a significant quantity of numerical value matches.

I gathered the unannotated camera documentation from company support sites. Specifically, I found the support/user manual for each camera, then I only collected the pages of the manual with the camera specifications listed. I managed to collect a little more than 550 documents; Specifically, I was able to collect about 150 documents from each of Canon, Olympus, Samsung, and Sony.

III: TRAINING DATA

Text processing:

I performed a few operations on the text before I trained my LR classifier. First, I ***tokenized*** the text. Tokenization was a two-step process: I split numbers from words since several attribute values had units concatenated to them (3x, 8.3MP, 5"), then I ran the Stanford English Tokenizer on the resulting text. The Stanford Tokenizer does some basic ***normalization***, such as converting parenthesis pairs into '-lrb-' and '-rrb-'. Second, I ***lower-cased*** the text. Finally, I ***stemmed*** the text with the Porter2 English Stemmer.

In order to train my classifier, I had to convert token features into numbers. In order to do this, I chose a ***closed vocabulary***; the vocabulary consists of the 2000 tokens that appeared the most frequently in the text. I chose to include both punctuation and word tokens in the vocabulary, but I excluded any words with numbers from the vocabulary. Punctuation was necessary to include since many camera attributes use punctuation to describe the values; some units are expressed with punctuation (3" and 8'), and several attributes are delimited by punctuation (4:3 and 1/400). I converted any token that existed in my vocabulary to its index within the vocabulary.

I took a similar approach to turn attribute names into numbers. Specifically, I alphabetically sorted the list of 70 attribute names. Then, I converted the attribute name to its index within this sorted list.

Features:

I only managed to use two types of features for my classifier: a feature for the tokens that

surrounded the number, and a feature for the attributes that preceded the number.

My first feature simply considers the 10 tokens surrounding each number. After looking at my data, I noticed that most attributes had the attribute name followed by a list of values, where each value usually had a unit. So, I include the 7 tokens to the left with the hope of capturing the attribute name, and I include 3 tokens to the right to capture the number's unit.

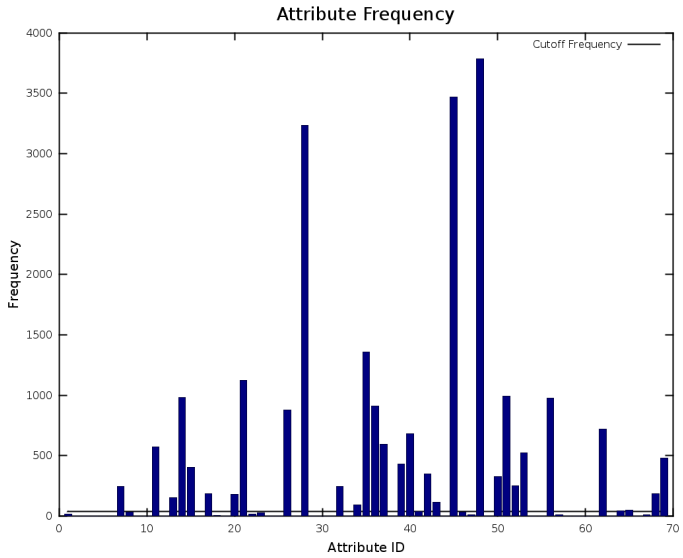
My second feature included the attribute value of the numbers preceding the number in question. I limited the window to the 7 tokens preceding the number; many attribute values were listed together, so the knowledge that the preceding number was classified as a certain attribute would make it more likely that the number in question is apart of the same attribute. Furthermore, many attributes were listed in the same groups, so the classifier could learn attribute dependencies.

My resulting feature vector had a dimension of 2071. This is due to my vocabulary size of 2000 tokens, an attribute size of 71 attributes (with the addition of the 'spurious' attribute). The feature vector was a binary vector, so it did not include frequency, only presence.

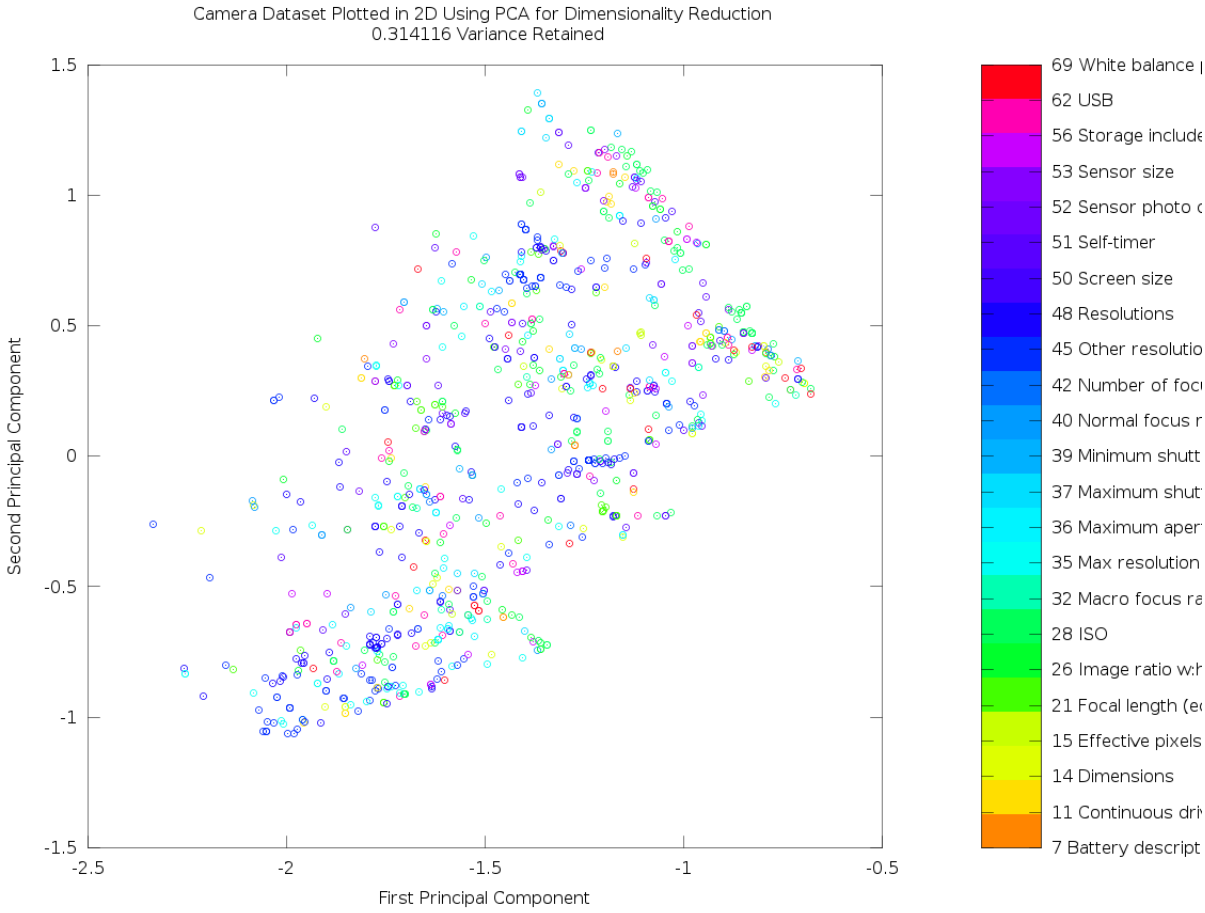
Attribute/Class Selection:

As I mentioned earlier, my database has 70 unique attributes, but I chose to use only 31 of the attributes/classes. This decision was motivated by the fact the several attributes were not described by numbers ('videography notes', 'body type', 'image stabilization', etc.). Furthermore, I could not collect enough examples from my data for some attributes. So, I chose to exclude attributes with fewer than 40 examples from my model.

Below is a graph of the frequency of each attribute (by attribute ID) within my corpus. The 3 attributes that have a significant number of examples are 'ISO', 'Other Resolutions', and 'Resolutions'. All three of these attributes list several distinct values for each camera.



Below is a graph that visualizes the data in 2D. The graph was produced by using Principal Component Analysis (PCA), which reduced the 2071 dimension feature vectors to 2 dimensions. The graph retains 31% of the data's variance. The legend shows the 23 most frequent attributes; each attribute is preceding by its attribute ID.



IV: TRAINING

My classifier is a one-vs-all Logistic Regression (LR) classifier. I used a classification threshold of 0 for each attribute/class. Furthermore, my classifier uses basic Gaussian prior smoothing; I used 0.5 as the value for my smoothing parameter (lambda).

Below are two word clouds for the tokens that are the learned top predictors for the attributes, ‘Effective Pixels’ and ‘Screen Size’ respectively.

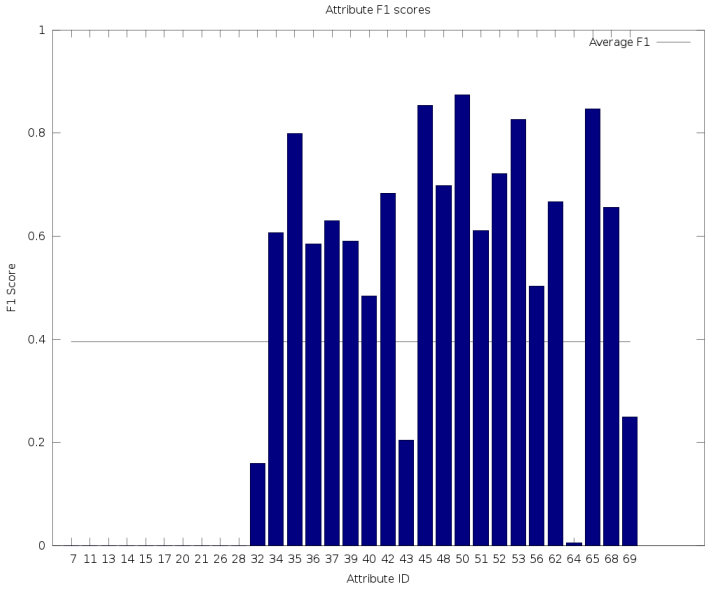
V: CLASSIFIER EVALUATION

I split my annotated data into three segments for evaluating my classifier: training, validation, and testing. I used the split 60%, 20%, 20% respectively. I made use of the training data for training my model. Then I used the validation and testing data for evaluation my model.

I used the validation data to produce learning curves. Learning curves were an invaluable tool for

early evaluation.

I used the testing data to evaluate the performance of my classifier for each of the attributes/ classes. Since one-vs-all classification results in skewed classes, I used F1 as my evaluation metric. My average F1 score for 31 attributes was 40%.



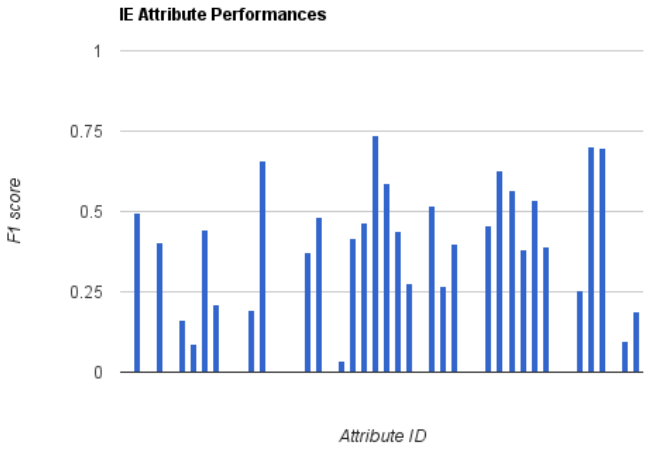
VI: INFORMATION EXTRACTION / SUMMARIZATION

I built an Information Extraction (IE) model by extending the classifier presented above. Specifically, the IE model performs the same steps as the classifier to produce a feature vector for each number; then, it uses the classifier to classify each vector. Finally, it keeps track of positive classifications.

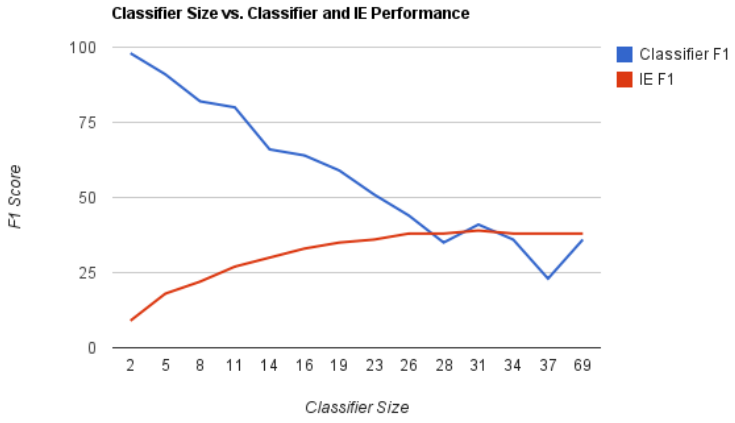
I also extended my IE model to create a foolishly simple summarization system. I simply stored the values for each attribute in a set so that there were no duplicate values per attribute.

VII: INFORMATION EXTRACTION / SUMMARIZATION EVALUATION

The camera database was also used for evaluating the performance of my Information Extraction (IE) model. Specifically, I compared the attribute-value pairs in the database to those produced by my IE model for each camera. With all of the parameters specified above, my IE model performs, on average, with 35.6% precision, 40.9% recall, and 38.1% F1. Be advised that I split my data to properly evaluate my classifier; however, my IE model is evaluated on all of the data, which is mathematically dubious but happened to be the easiest to implement.



I investigated how the size of my classifier affected extraction performance. I define the 'size' of a classifier as the number of attributes that it must discriminate between; for instance, the classifier mentioned throughout this paper has a size of 31 because it has 31 attributes to choose between. I chose the size of the classifier by the attribute's example frequency; for instance, my classifier above chose the 31 attributes with more than 40 examples. Therefore, I vary this cutoff number to choose the classifier size. As demonstrated by the graph on below, the larger the classifier, the worse the classifier performs, but the better the IE model performs. The IE model reached a maximum F1 performance of 39%.



VIII: SOFTWARE ACKNOLEDGEMENTS:

I used several existing software. Starting from the beginning, I used both Mechanize and Beautiful Soup for web scraping; both are Python libraries. Next, I used MySQL for storage and querying the scraped data. Then, I extracted text from PDF documents using PDFBox. Next, I used the Stanford NLP English Tokenizer for text tokenization. Additionally, I used the cs224n.util.Counter class to create my vocabulary. Then, I used the Porter2 libstemmer for word stemming. Then, I used Octave to train my model parameters; specifically, I used the CS229A conjugate gradient (linear optimization) implementation, fmincg.m, for my own implementation of Logistic Regression.