2016-05-01

CONFIRM: Clustering of Noisy Form Images using Robust Matching

Christopher Alan Tensmeyer
Brigham Young University

CONFIRM: Clustering of Noisy Form Images Using Robust Matching

Christopher Alan Tensmeyer

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Tony Martinez, Chair
Ryan Farrell
Parris Egbert

Department of Computer Science

Brigham Young University

May 2016

ABSTRACT

CONFIRM: Clustering of Noisy Form Images Using Robust Matching

Christopher Alan Tensmeyer
Department of Computer Science, BYU
Master of Science

Identifying the type of a scanned form greatly facilitates processing, including automated field segmentation and field recognition. Contrary to the majority of existing techniques, we focus on unsupervised type identification, where the set of form types are not known *apriori*, and on noisy collections that contain very similar document types. This work presents a novel algorithm: CONFIRM (Clustering Of Noisy Form Images using Robust Matching), which simultaneously discovers the types in a collection of forms and assigns each form to a type. CONFIRM matches type-set text and rule lines between forms to create domain specific features, which we show outperform Bag of Visual Word (BoVW) features employed by the current state-of-the-art. To scale to large document collections, we use a bootstrap approach to clustering, where only a small subset of the data is clustered directly, while the rest of the data is assigned to clusters in linear time. We show that CONFIRM reduces average cluster impurity by 44% compared to the state-of-the art on 5 collections of historical forms that contain significant noise. We also show competitive performance on the relatively clean NIST tax form collection.

Keywords: Clustering, Historical Documents, Form Recognition

Contents

List of Figures

List of Tables

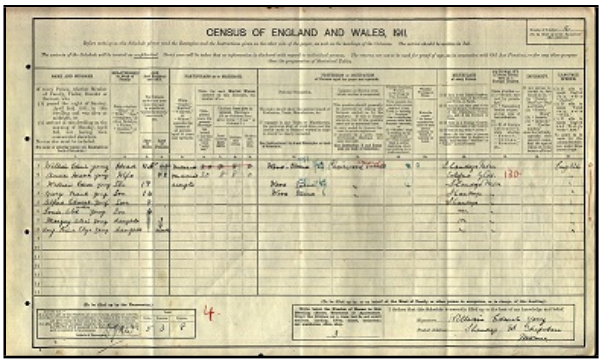
Chapter 1

Introduction

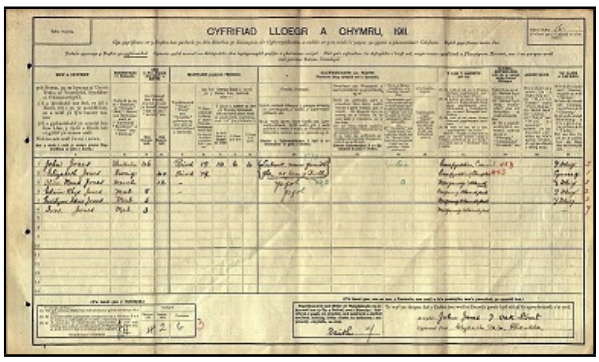
Document segmentation into logical snippets, such as text lines or table cells, is often used as a preprocessing step for text recognition systems [26]. For forms, which are defined by a rigid structure of fields, accurate segmentation can be acheived via alignment with a form template which encodes the relative spacing of the segmentation cuts for that form's particular structure or *type* [7, 29]. Such form templates can even be created automatically by simultaneous registration of many forms of the same type [19]. Therefore, the problem of form segmentation, given a set of predefined types and templates, can be reduced to supervised form recognition and template alignment. Contrary to form recognition, this work focuses on the unsupervised case where information about the types of forms in a collection is not known *apriori*. Thus, our task is to first discover the types of forms in a collection and then assign each form to one of the discovered types.

While form recognition is a well-studied problem (e.g. [5, 10, 16, 20, 28, 32, 37]), fewer works, have considered the task of form clustering. The goal in this case is to partition the collection by form types (not known *apriori*), which we define to be the exact layout structure of the form encompassing preprinted lines, text, and figures. Note that we exclude the user-filled field contents from our definition of structure.

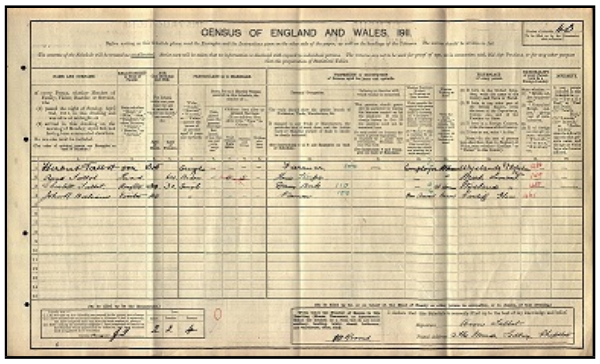
This work presents CONFIRM (Clustering Of Noisy Form Images using Robust Matching), an approach which performs unsupervised form type discovery and assignment in the presence of significant noise. This is done by clustering form images using novel collection-dependent features derived from matching Optical Character Recognition (OCR)



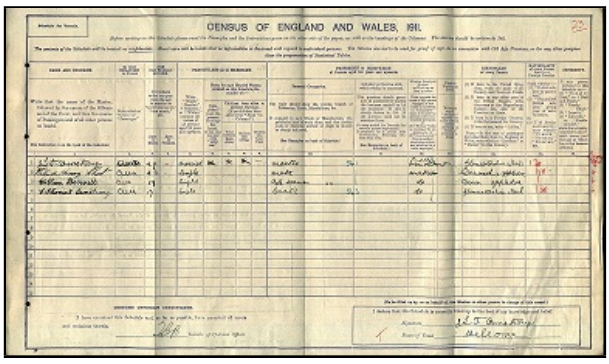
(a) Census Form in English



(b) Census Form in Welsh



(c) Census Schedule for Households



(d) Census Schedule for Vessels

Figure 1.1: Examples of similar, but distinct, form types from the *Wales* datasets. While supervised methods may learn salient differences from labeled examples, similar types are more difficult to distinguish in a clustering setting.

transcriptions and detected horizontal and vertical rule lines against exemplars chosen from the dataset. These domain specific features, combined with unsupervised structural similarity learning [23], allow CONFIRM to cluster discriminatively between similar, yet distinct, form types.

In order to scale to large datasets, CONFIRM employs a 3-stage bootstrapped clustering approach. In the first stage, a subset of the data is clustered directly using our proposed features. These clusters are individually split in the second stage to produce more pure subclusters, with each subcluster representing a discovered form type. In the final stage, the initially clustered forms are used to train a supervised classifier, which classifies each remaining form as one of the discovered types. Though a supervised classifier is employed,

the target values used for training are the latent types discovered in the initial clustering, making CONFIRM an entirely unsupervised method.

We define two form types be be *similar* if their structure differs by relatively few page elements. Examples of similar form types in the England and Wales 1911 Census are shown in Figure 1.1. These form types have the same basic layout: a header composed of a title and instructions, a body composed of a many-column table with text heavy column headers, and a footer with two components for totals and signatures. However, the small structural differences are consequential if template matching segmentation or recognition is used. For example, Figures 1.1a and 1.1c are visually similar, but the former has an additional column entitled *Language Spoken*, requiring a different segmentation template. Similarly, Figures 1.1a and 1.1b are the same form in English and Welsh respectively, so while segmentation might be the same, the field recognition process would differ.

The main difference between Figure 1.1d and Figure 1.1c is the footer, but depending on the end application, this may not be significant enough to warrant two different types. As a general method, CONFIRM does not make assumptions about the end application, but aims to partition the collection into fine-grained clusters of form types. If the end application needs less granularity, over-partitioned form clusters can be merged together in application-specific post-processing.

Evaluation for form classification and clustering typically is performed with the relatively clean NIST tax forms dataset SPDB 2 and/or SPDB 6 [11]. However, several recent techniques report perfect classification accuracy [6, 16, 21, 32] or perfect clusterings [23] on this dataset, making comparison between methods difficult. Furthermore, all form types in NIST are structurally and visually distinct, so more challenging datasets with similarly structured forms are needed to evaluate our method's performance. In addition to experiments on NIST tax forms, we show that CONFIRM reduces average cluster impurity by 44% compared to the state-of-the-art *HVP-RF* model of [23] on 5 collections of noisy historical forms that contain similar form types.

This work is organized as follows: Chapter 2 reviews the relevant literature. Chapter 3 describes form representation and the three clustering stages of CONFIRM. The algorithms for condensing the form representations into a fixed length feature vector are detailed in Chapter 4. Our evaluation protocol, including datasets used, is described in Chapter 5. Chapter 6 describes the experimental results and presents discussion. Chapter 7 provides concluding remarks.

Chapter 2

Related Work

This chapter presents related works on both form clustering and form recognition. Though this work targets form clustering, the feature representation and similarity metrics employed in form recognition techniques are directly related.

2.1 Form Image Clustering

In this work, the criterion for form clustering is by structure, so other clustering criteria, e.g. by author [9], field values, or for document reconstruction [4], are outside the scope of this work.

Saund [35] uses line detection to obtain horizontal and vertical line segments for each form. From these, global histograms over line junction patterns are computed and compared using the Common-Minus-Difference similarity metric to recover pairwise similarities among all forms. The forms are then clustered using an iterative greedy approach based on these similarities. On the NIST Tax Forms dataset [11] (11,185 tax forms of 20 types), Saund reports 100% cluster purity for 21 clusters. He claims that this form-specific feature representation makes the types very separable, which is true for the NIST tax forms dataset, but other real world datasets may have forms that have similar line structure but differ in the textual labels (e.g. Figure 1.1a and 1.1b). In contrast, CONFIRM uses both line and text features, and uses the spatial layout of the lines to compute a robust similarity metric.

The *HVP-RF* model of Kumar and Doermann [23] is a Bag of Visual Words (BoVW) approach which encodes form images as histograms of quantized SURF [3] features over

hierarchically partitioned regions of the image. Pairwise structural similarities between feature vectors are learned in an unsupervised fashion and then clustered using Spectral Clustering [36]. Using the Silhouette clustering metric [31] to determine K, the number of clusters, *HVP-RF* exactly recovers the ground truth partitioning of the NIST tax forms. CONFIRM incorporates the unsupervised structural similarity learning of *HVP-RF* (see Section 4.3), but uses novel matching features that better capture the structure of forms, and employs a 3-stage clustering approach to improve scalability. As [23] is the current state-of-the-art in form image clustering, we have reimplemented the *HVP-RF* model for direct comparison with CONFIRM.

Mariani et al [27] cluster document images based on layout similarity for a retrieval task. Though they do not evaluate over forms, their representation could be applied to form clustering with little modification. They encode document layouts using modified XY-trees and extract feature vectors from the Modified XY-trees using n-grams over discrete tree nodes. These feature vectors are clustered using a Self-Organizing Map (SOM) to create a retrieval index.

2.2 Form Image Classification

Form images have been represented in a large variety of ways for classification tasks (see [5] for a survey). These representations include statistics of image connected components [37], BoVW [6, 24, 38], OCR features [2, 32, 33], pyramids of average gray-scale values [18, 32], Viola-Jones features [34], Hidden Tree Markov Models [10], sequences of line segments [12, 20], sequence of line gap ratios [28], run length histograms [16], Shape Context Features [22], and most recently learned features from Convolutional Neural Networks [17, 21, 38].

Like [2, 32], CONFIRM combines both visual and textual features, but uses early fusion, rather than late fusion, because structural patterns can be learned that incorporate both the text and line segments.

Sako et. al. [33] classify bank forms by comparing the form text to predefined templates of keywords using a constellation matching algorithm. In CONFIRM, text matching uses all OCR detected text and matching occurs between form instances with no predefined templates. Additionally, CONFIRM performs prefix and suffix matching to correct OCR segmentation errors.

Chen et. al. [6] point out the difficulty of discriminating between similar form types in a supervised setting. However, this task is even more difficult for an unsupervised model because supervised models can learn to focus on the few key features that correlate with the given labels, while unsupervised models cannot.

In [12, 20, 28], forms are represented as sequences of vertical and horizontal rule lines, which are compared using a similarity metric such as edit distance or clique finding in an association graph. While these methods discretize or ignore the position or length of lines, CONFIRM performs a novel edit distance directly on a continuous representation of line segments, making it more robust to line detection errors. Additionally, CONFIRM uses the edit operations of the edit distance to merge together line sequences to compute the prototypical center of a cluster of forms.

Chapter 3

Overview of CONFIRM

This chapter gives a detailed overview of CONFIRM, which takes as input a collection of form images of unknown types and outputs a partitioning of the form images into types based on structure. The details on the features and matching algorithms are given in Chapter 4.

Algorithm 1 CONFIRM

```
1: function CONFIRM(I, K, n, e, m)
2:     Where I is a set of form images of unknown types,
3:     K is the number of initial clusters,
4:     n is the size of the initially clustered subset of I,
5:     e is the number of exemplars for feature matching,
6:     m is the size of the smallest allowed subcluster.
7:
8:     for all I_i ∈ I do
9:         F_i ← DetectPageElements(I_i)                    ▷ Section 3.1
10:    end for
11:    F ← {F_i | ∀i}
12:
13:    S ← RandSubset(F, n)                                 ▷ |S| = n
14:    C ← InitialCluster(S, K, e)                          ▷ Section 3.2
15:    C' ← RefineClusters(C, m)                            ▷ Section 3.3
16:    t ← Bootstrap(F, S, C', m)                           ▷ Section 3.4
17:    return t                                             ▷ t_i is the type of I_i
18:
19: end function
```

Algorithm 1 gives high level psuedocode for CONFIRM. First, page elements (text and rule lines) are detected in each form image (line 9), and a small subset of forms is chosen for an initial clustering (line 13). After initial clustering (line 14), clusters are refined on an

individual basis (line 15) to increase cluster purity. After refinement, a bootstrap procedure (line 16) uses the refined initial clustering to induce a clustering over the entire set of forms. In bootstrapping, a supervised classifier is trained using the initially clusted forms as inputs and the index of the containing refined cluster as the target class. Then each form in the whole collection, including those in the subset, is classified into a cluster, which yields the final clustering.

One key idea of CONFIRM is that at each of the three clustering stages, distinct feature vectors are computed by performing *exemplar matching* between each form and a set of *exemplars*. We use the term *exemplar matching* because the word *template* is normally reserved for known ground truth types, which the exemplars are not. In the initial stage, the exemplars are a subset of the initially clustered forms. In refinement and final clustering stages, exemplars are chosen to be the cluster prototypes of the clusters in the previous stage. By using distinct features in each stage, CONFIRM is better able to distinguish between similar form types in the refinement stage and classify data into clusters in the final stage.

We now describe each stage, leaving the details of exemplar matching and cluster prototype construction to Chapter 4.

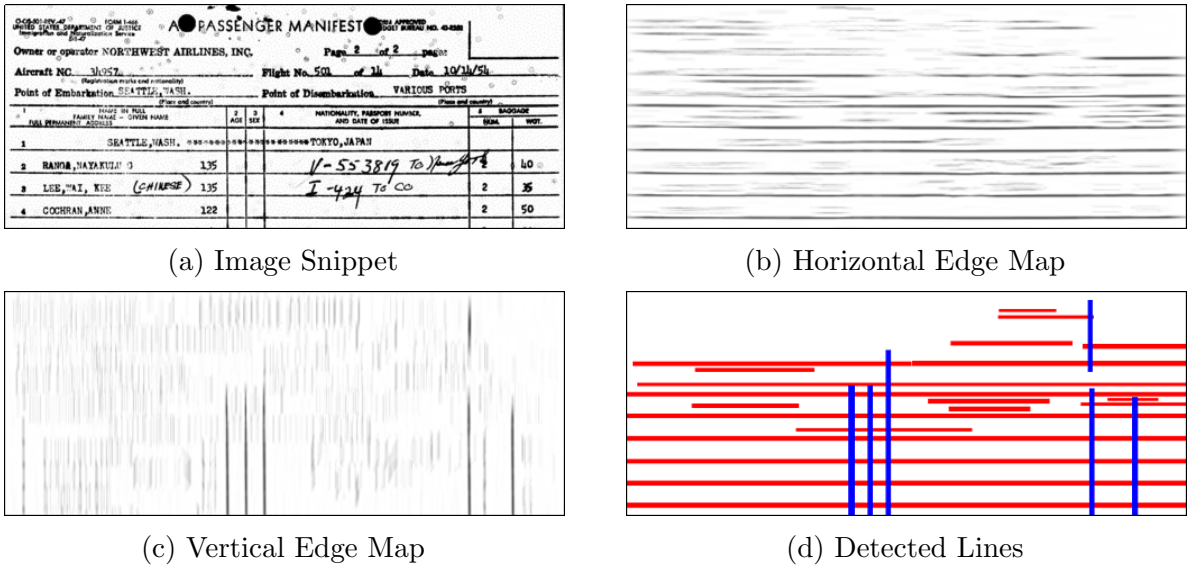
3.1 Page Element Detection

CONFIRM begins by detecting and recognizing type-set text lines and performing rule line detection.

3.1.1 Text Line Detection

Text line detection is done using OCR. In our experiments, a commercial OCR engine (ABBYY FineReader[1]) is used to detect and recognize text line transcriptions and bounding boxes, though other OCR systems may be used if they provide bounding boxes on the transcriptions.

[1]http://abbyy.com/finereader Version 10



(a) Image Snippet

(b) Horizontal Edge Map

(c) Vertical Edge Map

(d) Detected Lines

Figure 3.1: Simpe rule line detection using smoothed edge filters and connected component analysis. While results are somewhat noisy, CONFIRM is still able to match corresponding line segments between forms.

OCR engines may attempt to interpret non-typeset elements (e.g. handwriting, figures, rule lines) as text, introducing noise into otherwise mostly correct transcriptions. To filter out these erroneous transcriptions, we retain only the alphabetical characters and space character from the transcriptions and remove resulting text lines with length 3 or less. Empirically we found this routine to remove many, but not all, gibberish text lines, though it also removed some very badly transcribed real text, which may have been of some use to the algorithm. In practice, better filtering criteria could be used to remove/correct more noise, but one of our purposes is to demonstrate that CONFIRM can handle errors in the OCR transcriptions.

3.1.2 Rule Line Detection

We employ a single line segmentation detection algorithm that uses only basic image processing techniques. The resulting detections are likely nosier than what would be achieved with state-of-the-art line detection in documents, but this helps demonstrate CONFIRM's robustness to noise in the detection process. We assume that the form images have been de-skewed so that horizontal and vertical lines are rectified.

First, horizontal and vertical edge filters are convolved with the image to produce smoothed edge maps (see Figure 3.1). The edge maps are then thresholded using a high threshold for high precision of detected line segments. To obtain entire line segments, the original edge maps are thresholded with a low threshold and connected component analysis is done to find the full extent of each line segment that was detected with the high threshold. We also discard connected components that are very short or do not exceed an aspect ratio of 3.

Some of the common errors in line detection include false horizontal lines if text is printed too densely (see Figure 3.1), and missed or broken lines due to faded ink or poor digitalization. Nevertheless, CONFIRM is robust to these kinds of errors.

Algorithm 2 Initial Clustering

function InitialCluster(S, K, e)
  $E \leftarrow$RandSubset(S, e) ▷ $|E| = e$
  $\mathbf{V} \leftarrow$Match(S, E) ▷ Section 4.1
  $\mathbf{M} \leftarrow$StructureSimilarity($\mathbf{V}$) ▷ Section 4.3
  $C \leftarrow$SpectralCluster($\mathbf{M}$, K)
  return C
end function

3.2 Initial Clustering

The first stage (see Algorithm 2) of CONFIRM is a clustering algorithm, which operates on a previously chosen subset of forms (denoted by S in Algorithms 1 and 2). The forms not in S are eventually assigned to clusters by a classifier in the final bootstrap stage of CONFIRM. The rationale is that for n instances, comparison based clustering typically has runtime complexity $O(n^2)$ or $O(n^3)$ and space complexity $O(n^2)$, whereas supervised prediction is typically $O(n)$ time and $O(1)$ (for parametric models) space after model fitting. If $|S|$ is small relative to the entire dataset, CONFIRM can give a large speed up compared to directly clustering the whole dataset. For very large collections (e.g. UK 1911 Census contains over 14

million forms), directly clustering the entire dataset may not be feasible with single machine hardware, though for small datasets, S may be chosen to be the entire dataset.

CONFIRM attempts to find distinct form types within S, so it is important that $|S|$ be large enough to include several examples of each type. Thus the ideal $|S|$ is not dependent on the absolute size of the collection, but on the number and distribution of actual types. Our experiments show that $|S| \geq 1000$ is large enough to make bootstrapping as accurate as direct clustering (see Section 6.3), though in many cases, smaller $|S|$ works well too. This result holds for datasets with both balanced and skewed type distributions with up to 30 actual types.

From S, a small number (e.g. 50) of forms are sampled without replacement and denoted as *exemplars* (E in Algorithm 2). These exemplars are used to create *feature vectors* for all forms in S using the matching procedure described in Chapter 4.1. Essentially, each dimension of the feature vector is associated with a unique page element (text or rule line) of one exemplar in E, and the value indicates how well that page element was matched.

These feature vectors may be high dimensional because each exemplar may contain hundreds of page elements. These features may also be sparse, but this depends on the similarity of the types in the collection. Distance-based clustering algorithms struggle with high dimensional data due to the *curse of dimensionality*. We employ the unsupervised structural similarity learning algorithm of [23] (see Section 4.3) to produce similarity scores for each pair of forms.

CONFIRM then uses Spectral Clustering (SC) to cluster the forms. We choose SC because the geometric shape of the clusters is unknown, and SC can handle arbitrarily shaped clusters of different densities. The only input parameter for SC is K, the number of clusters, though the true number of types or clusters is generally unknown. During cluster refinement, the initial clusters are further partitioned to create $K' \geq K$ clusters if clear subclusters can be identified. Our experiments show that if K is set too small by the user, it is often increased to approximately the correct number of clusters.

One common approach for automatically selecting K is to perform a grid search over K and take the highest scoring clustering according to some clustering metric. Following [23], we attempted to do this using the Silhouette metric [31]. The Silhouette metric is based on measures of cluster compactness and separation, which favor dense spherical clusters. In our case, Silhouette scores were insufficient to identify the true K, most likely due to non-spherical clusters produced by SC. Indeed, The Silhouette score for the ground truth clusters is roughly half the score of the clusterings produced by CONFIRM, making the metric inappropriate for detecting K in this application. Because of this, we compare models over a range of K, using $max(2, K^* - 5) \leq K \leq K^* + 5$, where K^* is the number of actual types.

Algorithm 3 Cluster Refinement

function RefineClusters(C, m)
  Let C be a clustering, such that each C_i
  is a set of forms

  for all $C_i \in C$ **do**
    $p \leftarrow$ Prototype(C_i) ▷ Section 4.2
    $P \leftarrow \{p\}$
    $\mathbf{V} \leftarrow$Match(C_i, P) ▷ Section 4.1
    $\mathbf{M} \leftarrow$StructureSimilarity($\mathbf{V}$) ▷ Section 4.3
    $C'_i \leftarrow$ OPTICS($\mathbf{M}$, m)
  end for
  $C' \leftarrow \bigcup_i C'_i$
  return C'
end function

3.3 Cluster Refinement

After the initial clustering, CONFIRM refines clusters independently, splitting each cluster if clear subclusters are found, producing $K' \geq K$ clusters. Pseudocode is given in Algorithm 3. One goal of refinement is to increase cluster purity because these clusters serve as training data for a supervised classifier that maps cluster members to clusters, so purer clusters lead to less noisy training data. Our experiments show that including this stage increases purity

by 2.3% at the expense of a few additional clusters. Another related goal is to provide robustness to the user chosen number of clusters, K. Empirically, if K is set too low, then cluster refinement is able to approximate the correct number of clusters (see Section 6.2).

Each cluster is split independently of all other clusters based on a feature representation unique to that cluster. First, the cluster prototype, representing the *average* form, is computed by merging together the page elements of each cluster member. The page elements that are retained in the prototype are ones frequently found among the forms in the cluster. See Section 4.2 for more details on this process.

Once the prototype is constructed, it serves as an exemplar to creating feature vectors for the cluster members using the matching algorithm described in Section 4.1. While many exemplars are used in initial clustering, only a single exemplar (i.e. the cluster prototype) is used per cluster for refinement. If the initial clustering is good, then mostly forms of the same or similar types were put in the same cluster. If there are large numbers of two or more types in a cluster, then elements of all types appear in the prototype and hence in the feature vectors used for splitting the cluster. Because only one exemplar is used, there is a higher percentage of features that can discriminate between the similar types that were initially clustered together.

As in initial clustering, pairwise structural similarities between the feature vectors for each cluster are learned using the technique of [23]. Then OPTICS clustering [1] is applied to these learned similarities to create to one or more subclusters. OPTICS is a density-based clustering algorithm where data density determines the number of resulting clusters, subject to a user set parameter, min_pts (m in Algorithm 3), which specifies the minimum number of points a (sub)cluster must have. If no subclusters can be found of at least min_pts, then the cluster is not split.

After refinement, the prototypes for all of the subclusters can be computed and used as representatives of the learned form types. While a prototype does not have a corresponding input image (as a medoid would), the features used in CONFIRM are page elements and can

be rendered in image form. This gives a user some idea of what was learned by CONFIRM. Some example prototypes are shown in Figure 4.2 in Section 4.2. While CONFIRM is completely unsupervised, additional low-effort user input could be used at this point to merge together clusters that correspond to the same form type before bootstrapping this clustering to the rest of the dataset.

Our experiments show that refining the initial clustering yields better results than simply performing the initial clustering with a larger value of K. We believe this occurs because the feature vectors used in cluster refinement better highlight the subtle differences between similar forms within a cluster, differences which are not as apparent when a large number of dissimilar exemplars are used for initial clustering. Additionally, the refinement process reduces the sensitivity to the user set number of initial clusters. Without this process, the user might be tempted to set K too high as a precaution, but as we show, refinement tends to increase the number of clusters to at least the true number.

Algorithm 4 Bootstrap Classification

function BOOTSTRAP(F, S, C', m)
  $P \leftarrow \emptyset$
  for all $C'_i \in C'$ **do**
    if $|C'_i| \geq m$ **then**
      $p_i \leftarrow$ PROTOTYPE(C'_i)
      $P \leftarrow P \cup \{p_i\}$
    end if
  end for

  Let c_i be the index for the cluster containing S_i
  $\mathbf{X} \leftarrow$MATCH(S, P) ▷ Training Data
  $\mathbf{y} \leftarrow [c_0, \ldots, c_{|S|}]$ ▷ Training Labels
  $Model \leftarrow$TRAINCLASSIFIER($\mathbf{X}$, $\mathbf{y}$)

  $\mathbf{X}' \leftarrow$MATCH(F, P) ▷ Test Data
  for i in $0 \ldots |F|$ **do**
    $t_i \leftarrow Model$.PREDICT($\mathbf{X_i}$)
  end for
  $\mathbf{t} \leftarrow [t_0, \ldots, t_{|F|}]$
  return t
end function

3.4 Bootstrapped Classification

At a high level, this stage uses the refined clusters from the previous section to train a supervised classifier, which then classifies each form in the entire collection into one of the refined clusters to produce a final clustering over all the data (see Algorithm 4). The main advantage of bootstrapping over a direct clustering is speed, though our experiments show it also increases cluster purity in many cases.

To construct a training set composed of data observations and labels, prototypes for each of the refined clusters are computed via page element merging (Section 4.2) to create a set of exemplars. As in previous stages, these exemplars are used to create feature vectors for each form in the initially clustered subset of forms, S. The label for each observation is the index of the refined cluster containing the form. Together, these features and labels are used to train a supervised classifier. The trained classifier then predicts a cluster for all forms in the original collection, which induces a clustering over the data.

Normally, supervised learning involves fitting a model to a human-curated training set with gold standard labels. Ideally, there should be a one-to-one mapping from the clusters to actual types, though this would require a perfect initial clustering. We have identified two potential problems that arise from our use of bootstrapped labels derived from clustering:

1. Some forms belong in different clusters even after refinement, so their labels derived from the clustering are wrong. This is commonly known as *label noise*.
2. There are more clusters than types, leading to 2 or more clusters (and therefore learnable classes) that correspond to the same actual type.

One strategy for dealing with label noise is to use a classifier that is less sensitive to this problem [15] such as Random Forest (RF) [13, 14]. We initially used RF, but later switched to Logistic Regression (LR) with L_2 regularization because it consistently performed better. We believe this is due to the second issue; RFs are more prone to overfit than a simpler model like LR. When there are two identical classes representing the same target

concept (in our case, the same actual types), the classifier attempts to discriminate between these two identical classes based on statistical noise, leading to overfit. This is less of a problem for LR which uses a *one-vs-all* classification scheme because not all *one-vs-all* models are affected by the two identical classes, whereas each tree in RFs are affected.

Clusters are predicted even for the forms comprising the training data. Though many supervised models learn to exactly reproduce the labels for their training data, we have observed that LR does not in all cases. Empirically, this increases the purity of clusters and can be seen as correcting initially mis-clustered forms using a discriminative bias.

The main reason that CONFIRM performs clustering in multiple stages is for speed. Suppose that there are N total forms in the collection, and that we choose a subset of size $n \ll N$. It is important that all types be represented in the subset, so n is affected by the distribution of types, not necessarily by N itself. The time complexity of initial clustering and cluster refinement is dominated by the eigen decomposition step of Spectral Clustering, which is $O(n^3)$. Constructing the training set is an $O(nK')$ operation because each of n forms must be matched against the prototypes of each of $K' \geq K$ clusters. Once the classifier is trained, each classification decision is an $O(K')$ operation because each form is matched against the K' prototypes to create its feature vector. Therefore overall, the complexity is $O(n^3 + nK' + NK')$, which under the assumption of $n \ll N$, reduces to $O(NK')$. We also note that the majority of the steps in CONFIRM can take advantage of parallel architectures for further speed up.

Chapter 4

Features

In CONFIRM, forms are represented as lists of page elements, so that forms of different complexity can be modeled appropriately [39]. In this work, type-set text and rule lines are used, though this algorithm can be extended to other kinds of page elements (e.g. handwritten text, figures, areas of white space) if appropriate match and merge functions are provided. In various stages of CONFIRM, fixed-length real valued vectors are extracted from the page element representation for use in standard clustering and classification algorithms. This vectorization occurs by matching page elements of the form with the page elements of an *exemplar* to find corresponding elements. This process is detailed in Section 4.1. The merging algorithm used to compute cluster prototypes is explained in Section 4.2.

Algorithm 5 Feature Matching

```
function MATCH(F, E)
    Let X be a matrix of |F| rows with x_i denoting
    the i^th row.
    Let || be vector concatenation
    The functions Match{Text,Horz,Vert} are
    described in Sections 4.1.1 and 4.1.2

    for all F_i ∈ F do
        x_i ← ()                                   ▷ Empty Vector
        for all E_i ∈ E do
            x_i ← x_i || MATCHTEXT(F_i, E_i) ||
            MATCHHORZ(F_i, E_i) || MATCHVERT(F_i, E_i)
        end for
    end for
    return X
end function
```

4.1 Feature Matching

Feature matching is the process of finding corresponding elements between two forms. The result is called a *match vector*, which encodes how well each page element of one of the forms (typically the exemplar) was matched. Each value in the vector takes on a value in $[0, 1]$, where 1 indicates perfect match and 0 indicates no match. Text and rule lines have separate matching algorithms, though vertical and horizontal lines can be treated as rotated cases of each other.

4.1.1 Text Matching

Each form has a number of text lines detected by OCR (see Section 3.1). Each text line is represented as a 6-tuple (s, c, l, t, b, r), with the following definitions.

1. s is a string or sequence of characters. This is the OCR transcription with non-space, non-alpha characters removed.
2. l, t, b, r (left, top, bottom, right) are bounding box coordinates of the transcribed text.
3. c is a *count* variable that is initialized to 1. It is approximately the number of times the text line has been matched during the merging process.

In Algorithm 5, the subroutine $MatchText$ takes as input two sequences of text lines, and outputs a binary vector indicating if each text line in the first sequence was matched by some text line in the second sequence. The length of the output vector is equal to the number of text lines in the first input sequence, which is typically an exemplar used for feature extraction. Matches are made greedily by iterating over all pairs of text lines between the two sequences. Once a text line has been matched with another, it is no longer eligible for further matches. Because of greediness, the resulting matching is order dependent, but preliminary experiments did not show any improvement in overall results when using a less greedy method.

Pairs of text lines that have both similar bounding boxes and similar strings are accepted as matches, with similarity thresholds being set by the user. For our results, we tried a small number of values for these similarity thresholds on the *Wales-Small* dataset and observed marginal differences in performance. Based on this, we chose thresholds (values noted below) that we think have high recall of true matches and we did not tune these parameters for the other datasets.

Bounding boxes are considered similar if the Euclidean distances between corresponding corners are less than some user-defined a. Even forms of the same type might have different margins or imaging geometry, so that corresponding elements are not found at the exact same (x, y) coordinates. As noted previously, initial results were not sensitive to a, so we have set $a = \frac{max(h,w)}{4}$, where h and w are the height and width of the image respectively. An additional area requirement for bounding box similarity is $\frac{4}{5} \leq r \leq \frac{5}{4}$ where $r = \frac{area_1}{area_2}$.

For a string similarity criteria, we use a normalized edit distance cost with unit cost for insert/delete and substitution operations. This similarity metric models the character errors made by OCR. The unnormalized edit distance [25] of two strings s_1 and s_2 of lengths n and m respectively is given by:

$$NormEditDistance(s_1, s_2) = \frac{d(n, m)}{max(n, m)}$$

$$d(i, j) = min \begin{cases} d(i-1, j) + 1 \\ d(i, j-1) + 1 \\ d(i-1, j-1) + \mathbb{1}(s_{1i} \neq s_{2j}) \end{cases}$$

with $d(i, j) = 0$ if $i = 0$ or $j = 0$ as the base case.

Normalizing the edit distance is intuitive because the expected number of OCR character errors is proportional to the length of the string and the error rate. If we set the

acceptance threshold to $\frac{1}{u}$, i.e. we require

$$NormEditDistance(s_1, s_2) \leq \frac{1}{u},$$

then we allow at most 1 character error for every u characters in the longer string. For strings shorter than u, we allow up to 1 error for matching, rather than requiring string equality. For this work, we set $u = 5$ to prevent falsely matching correctly transcribed, but similar strings.

After all text line pairs have been examined for matches, remaining unmatched text lines are considered for prefix and suffix matches. This accounts for OCR segmentation errors, where a text line is erroneously split into two separate text lines, or when two text lines are erroneously concatenated (e.g. two table cells in the same row, but adjacent columns). If two text lines from the same sequence are adjacent in the image and when concatenated match a text line in the other sequence (using the same conditions as above), then all three text lines are marked as matched.

4.1.2 Rule Line Matching

Rules lines are organized into two disjoint sequences: one for horizontal lines and one for vertical lines. Both sequence is matched and merged independently of the other, but in the same fashion. This is accomplished by a 90° rotation of horizontal lines to make them vertical. Each rule line is represented as a 4-tuples (x, y, l, c) with an assummed vertical orientation. These are defined as:

1. (x, y) is the coordinate of the closest line segment endpoint to the origin located in the upper-left corner of the image. Line segments are sorted first by x, then by y.
2. l is the length of the line segment, measured in pixels, making $(x, y + l)$ the coordinate of the other endpoint.
3. c is a *count* variable and defined the same as for text lines.

Operation	Description	Cost
Deletion	a is deleted	a_l
Match	a matches b	0
Contain	a contains b	$a_l - b_l$
Overlap	a and b overlap	len of non-overlap
Connect	a bridges gap between b and b'	Two Overlaps or Contains
Transpose	a matches b' and a' matches b	0

Table 4.1: Edit operations for sequences of parallel lines. Let a and b be lines from different sequences. Let a' be the line immediately following a (b' defined similarly). Operation costs are scaled by the *count* of a and/or b.

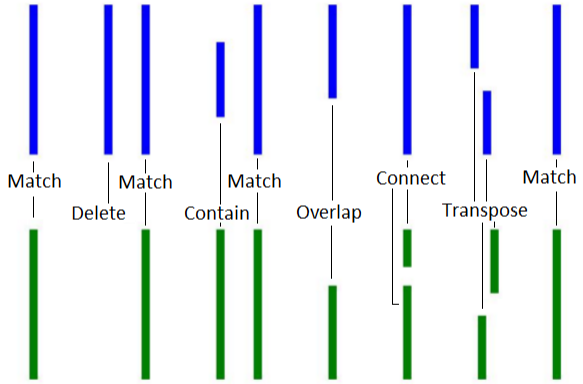


Figure 4.1: Alignment and edit operations detected with rule line edit distance algorithm for two sequences of vertical lines. (top/blue and bottom/green). Corresponding lines are aligned vertically and labeled with the detected edit operation.

While text lines frequently have clear matches based on string content, finding corresponding rule lines is more ambiguous, so we resort to a novel global alignment scheme similar in nature to string edit distance. String edit distance finds the lowest cost sequence of operations necessary to transform one string into the other. While strings are sequences of discrete symbols and use discrete edit operations and costs, lists of rule lines are sequences of multi-dimensional continuous symbols. However, the edit distance framework can be adapted to use edit operations that compensate for common line detection errors and use edit costs based on the continuous line representation (see Table 4.1). We introduce the edit operations of *Overlap*, *Contain*, and *Connect*, while adapting the traditional string edit operations of *Match*, *Deletion*, and *Transpose* to our representation of line segments. Figure 4.1 shows an example of these edit operations applied to two simple sequences of lines.

The general rule for the edit operation cost is the length in pixels of the parts of the lines that do not overlap. This base cost is scaled by the *count* of each of the lines involved. Thus lines with higher count are given priority and more greatly influence the global alignment of the line sequences. As cluster prototypes are constructed by merging cluster members, the count variables of lines that are matched (using this edit distance) is increased. Lines that are consistently matched among cluster members are likely part of the underlying form type and not detection errors.

After using Dynamic Programming (DP) to find the optimal sequence of edit operations and costs, those edit costs are used to construct the match vector. Each line, a_i, in the first (exemplar) sequence is associated with exactly 1 edit operation and cost, denoted $cost(a_i)$. To transform this value to one that indicates matching, we define $match(a_i) = 1 - \frac{cost(a_i)}{del\_cost(a_i)}$, where $del\_cost(a_i) = a_{ic}a_{il}$ is the cost of deleting a_i and is the maximal possible edit cost. The resulting match vector $\mathbf{m}$ is defined by $\mathbf{m}_i = match(a_i)$.

The criteria for what edit operation may be applied to any given pair, triple (for Connect), or quadruple (for Transpose) of lines depends on some heuristics of line position and length. For each (vertical) line, a_i, we can compute its distance to the first matched line, a_s, $\Delta a_{ix} = a_{ix} - a_{sx}$. For a_i to be eligible for a non-Deletion operation with a line b_j, we require $|\Delta a_{ix} - \Delta b_{jx}| < v$, where v is a predefined threshold. In our experiments, we set v to be 5% of the largest image dimension. As with the text matching similarity thresholds, we tried a few values on the *Wales-Small* dataset and found that it had only marginal impact on performance.

Using Δa_{ix} instead of a_{ix} introduces global translation invariance which is necessary because the two forms may have translated coordinate systems due to, e.g., misaligned scanning or different paper sizes. However, the use of relative distances cannot be used for determining the first match, so potential first matching pairs use absolute coordinates (e.g. a_ix and b_jx) and incur a small penalty if the absolute distance between the pair is large.

This penalty acts as a prior that the two coordinate systems are close, but allows for larger differences if a better alignment can be obtained with a large shift.

Additional heuristic include:

- Matching lines a_i and b_j must also satisfy $|\Delta a_{iy} - \Delta b_{jy}| < v$ and $\frac{5}{6} \leq r \leq \frac{6}{5}$, where $r = \frac{a_{il}}{b_{jl}}$.
- Transposition, i.e. swapping the sequence order of two adjacent lines, must result in two pairs of matches.
- For a_i to connect two lines b_j and b_{j+1}, we require $b_{(j+1)x} - b_{jx} \leq \frac{max(h,w)}{100}$, where h and w are image height and width respectively. Also, the length of the combined b_j and b_{j+1} must be similar to a_{il}.
- a Contains b if both endpoints of b are between the two end points of a. If neither line contains the other (and are not a Match), then they are eligible for an Overlap operation.

Algorithm 6 Prototype Construction

```
function PROTOTYPE(F)
    P ← F_1
    for i in 2 ... |F| do
        P ←MERGE(P, F_i)                ▷ Section 4.2
        P ←PRUNE(P)                     ▷ Section 4.2
    end for
    P ←FINALPRUNE(P)                    ▷ Section 4.2
    return P
end function
```

4.2 Feature Merging

Cluster prototypes are single representations that summarize the entire contents of a cluster. While a centroid or medoid representation is simple to compute for clusters of feature vectors, in CONFIRM, forms are represented as lists of page elements, so it is not obvious how

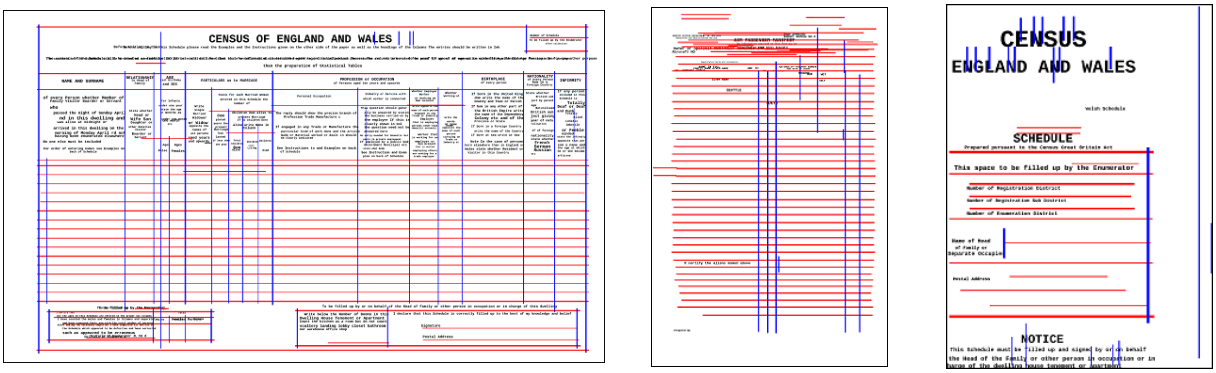


Figure 4.2: Image renderings of cluster prototypes obtained by merging together form representations within clusters. Horizontal lines are rendered in red, vertical in blue. Text lines are rendered with the largest font size that does not exceed the bounding box. These prototypes come from clusters of 10-20 forms from the *WashPass* and *Wales* datasets.

to obtain cluster prototypes. We have chosen to compute a psuedo-average by merging together page element representations, so that if multiple types are present in a cluster, then the prominent page elements of all types are found in the resulting prototype, which aids cluster refinement. While the goal is to merge together all forms in a cluster, prototypes are constructed though iterative pair-wise merging of all cluster members (see Algorithm 6).

Merging a pair of forms is based on finding corresponding page elements between the two forms using the matching algorithms described in Section 4.1. Page element pairs that match are merged together by combining their descriptors and adding their *counts*. Unmatched elements from both input forms are retained unmodified in the merged representation. After each merge, CONFIRM performs pruning by decrementing the *count* of each page element by a small value, t, and removes elements with $count < 0$. Thus $\frac{1}{t}$ is a lower bound on the percentage of forms in a cluster that must contain a page element for it to be part of the final prototype. Infrequent elements that are removed may be erroneous detections, true elements of a mis-clustered form of another type, or natural intra-type variation (e.g. type-set field content). For text lines, $t = \frac{1}{15}$ and for rule lines, $t = \frac{1}{10}$. On the few values we tried, these numbers performed best on *Wales-Small* dataset and reflect that matching text lines is more robust than matching rule lines.

After all pair-wise merges, a final pruning is accomplished by removing all elements with *count* less than 10% of the largest *count*. Pruning relative to this threshold ensures that page elements found in the final prototype are matched by a significant percentage of the cluster members.

As with feature matching, text, horizontal, and vertical line sequences are each merged independently.

4.2.1 Text Merging

To merge two matching text lines, a and b, to create a new text line $z = Merge(a, b)$, the coordinates of the average bounding box are computed as

$$z_t = \frac{a_c * a_t + b_c * b_t}{a_c + b_c}$$

with similar definitions for z_l, z_b, z_r. Additionally, $z_c = a_c + b_c$.

The resulting text line also includes a string transcription, z_s, which cannot be a simple average of a_s and b_s. To remedy this, each prototype text line keeps a histogram of all of the merged transcriptions, which we denote z_h. To choose a single z_s we use the most frequent string in z_h. If two strings tie for most frequent in z_h, we compute an approximate string median of z_h. Computing an optimal string median, a string that minimizes the sum of edit distances to a set of strings, is an NP-Complete problem [8].

When a complete text line has a prefix/suffix match, the prefix and suffix are concatenated and they are merged as a complete match. Unmatched text lines in either input form are copied to the merged representation.

4.2.2 Rule Line Merging

Merging sequences of rule lines is done by merging together pairs, triples, or quadruples of rule lines according to the edit operations computed by applying the matching algorithm

described in Section 4.1.2. In all cases, rule line merges preserve the total sum of each line's length times its *count*, such that the total cost of deleting the resulting sequence is equal to the sum of costs of deleting both input sequences.

How lines are combined depends on the edit operation. For $z = Merge(a, b)$, if a and b are *Matched*, then like text lines,

$$z_x = \frac{a_c * a_x + b_c * b_x}{a_c + b_c}$$

with similar definitions for z_y and z_l. Again $z_c = a_c + b_c$. *Transpose* operations are handled as two *Matched* operations. *Deletions* of single lines are handled just as unmatched text lines; the deleted line is simply included in the resulting line sequence.

For the *Contain* and *Overlap* operations, the merge is not symmetric: the lines from one sequence are tweaked to become more like the lines of the other sequence. This is because a long line might contain or overlap a very short line (because it is less costly than just deleting the lines) and a strict average in this case is not representative of the ideal sequence. To minimize damage done by these cases, the change in length of the lines is proportional to the ratio of line lengths. Intuitively, the closer the lines are in length, the more the merge looks like that of a *Matched* operation. *Connect* operations are handled as two *Overlap* or *Contain* operations by splitting the encompassing line down the middle and pairing its parts with the two lines it connects in the other sequence.

4.3 Unsupervised Similarity Learning

This section summarizes the unsupervised similarity learning technique proposed in [23] that learns pairwise similarities from structured high dimensional feature vectors. For more details, see the original paper [23]. This is used by CONFIRM in both initial clustering and cluster refinement. By basing similarity on structure or correlated features, we bias the clustering algorithm toward our human notion that forms are defined by rigid structure.

If D is an NxM data matrix (N instances, M features), then we first sample an NxM random matrix A, such that each A_{ij} is set to a random value from the j^{th} column of D. Note that the j^{th} column of A has the same univariate statistics of the corresponding column in D, though any correlations (i.e. structure) present among the columns of D has been lost by sampling all values within each row independently. For example, consider a row d in D and a row a in A. In general, elements of d are not independent: $\forall i, j \;\; p(d_i|d_j) \neq p(d_i)$. However, because of the construction of A, ignoring bias introduced by finite sampling, $\forall i, j \;\; p(a_i|a_j) = p(a_i)$.

A Random Forest (RF) classifier is then trained with rows of D as positive instances and rows of A as negative instances. Given the construction of A, classifiers that independently consider each dimension (e.g. linear classifiers) are unable to learn this task better than chance. The RF must exploit the structure present in D, i.e. $p(d_i|d_j)$, in order to learn this task. Each tree in the RF maps an input to a class-specific leaf node based on a subset of input features learned by each tree. Two inputs must share a common subset of features in order to be mapped to the same leaf node in any given tree. By training the RF to distinguish between D and A, each tree learns groups of features that correlate in D. The structural similarity of two inputs is computed as the percentage of trees in the RF where they arrive at the same leaf node.

Chapter 5

Evaluation Protocol

This chapter details the experimental setups used to validate our work, including the metrics used for cluster evaluation, models tested, and the datasets used.

5.1 Evaluation Criteria

For the datasets we utilize, there are ground-truth form type labels, which we use solely for evaluation purposes and not as input to CONFIRM. Our chosen metric is cluster *purity*, which can be calculated as

$$\text{purity}(C) = \frac{100}{N} \sum_{i=0}^{K} \max_{j}(C_{ij})$$

where N is the total number of instances, and C_{ij} is the number of instances with label j in the i^{th} cluster. Purity can also be thought of as classification accuracy if we assign each instance a label equal to the majority label for its cluster. Our targeted use case is clustering forms into types and then treating each cluster as a distinct type for further processing (e.g. template creation, segmentation, and recognition). Purity then measures the percentage of forms that receive the correct downstream processing.

Measuring cluster purity without taking into account K, the user-supplied number of clusters, is problematic because the trivial clustering where every instance has its own cluster yields perfect cluster purity. If K^* is the true number of clusters for a given dataset, then ideally we would set $K = K^*$, but in a real application K^* is not known. A typical method (e.g. [23]) for selecting K is to try a range of values and see which one optimizes

a blind clustering metric such as the Silhouette measure [31]. Initially we attempted to set K in this fashion, but this often yielded poor choices of K (e.g. $K = 2$ when $K^* = 11$). We then examined the Silhouette score for the ground truth clusterings (i.e. 100% pure clusterings with $K = K^*$) and found they were roughly half that of the imperfect clusterings recovered by CONFIRM and the *HVP-RF* model [23]. This reflects the difficulty of finding form representations that cluster into dense spherical clusters that can be evaluated on the criteria of compactness and separability. Therefore, we first evaluate all models over a broad window of K centered on K^*, i.e., $K \in [max(2, K^* - 5), K^* + 5]$ For the remainder of our experiments, we set $K = K^*$.

We recognize that in many scenarios it is better to over-cluster (i.e. $K > K^*$) than to under-cluster ($K < K^*$). If clusters are manually reviewed (e.g. for template creation), a user can easily merge two clusters that correspond to the same form type, but it is more difficult for a user to split a cluster containing forms of several types. In automatic systems, there may be some additional overhead incurred for having more clusters, but more forms will be processed correctly because the cluster purity is higher. We do show that when K is set too low by the user, the cluster refinement stage increases the resulting number of clusters.

The min_pts parameter in cluster refinement is always set adapatively. For a cluster of size N, we set $min\_pts = max(30, \frac{N}{10})$. This can be interpreted as clusters must be at least 30 instances and no cluster can be split into more than 10 sub-clusters. Allowing smaller fractions of large clusters increases the chance of splitting an already pure cluster based on noise. We experimented with setting min_pts adaptively as given, or using a fixed value, and adaptively works better because datasets and clusters are different sizes. We believe 30 is a good minimum because smaller clusters might provide too few instances for the supervised classifier in bootstrapping to learn what that cluster is. How min_pts is set could certainly be optimized further and is a subject of future work.

Except where noted, we use 50 random exemplars for initial clustering, and cluster the entire dataset during initial clustering. For each experiment and choice of parameters, each

model is run 5 times and the mean purity is reported. Initially, we also evaluated CONFIRM using V-measure [30] because, unlike purity, it penalizes over clustering; however, we omit V-measure results because they proved to be redundant with the average-purity measure across a fixed range of clusters.

5.2 Models

For CONFIRM, we examine and compare the purity clusterings produced after each stage: Initial Clustering, Cluster Refinement, and Bootstrapped Classification. In our experiments, we denote these models as *C-Initial*, *C-Refine*, and *C-Bootstrap* respectively. While the user can choose K for the initial clustering, $K' \geq K$ clusters are produced in Cluster Refinement, so it is unfair to directly compare the initial clustering and the clustering after refinement. For fair comparison, we run *C-Refine* and *C-Bootstrap* with smaller K, and average purity for clusterings where K' falls into the chosen evaluation range. We also evaluate other CONFIRM variants to help show the contribution of each part. *C-No-Refine* is the model where bootstrapped classification is performed directly on the initial clustering with no cluster refinement stage. *C-Text-Init* and *C-Line-Init* are initial clustering models that differ from *C-Initial* by only using text or rule line page elements respectively.

For comparison with the state-of-the-art, we have reimplemented the *HVP-RF* model [23]. With our implementation, we have reproduced the key reported result, which is a perfect clustering on the NIST tax forms dataset. We tuned the dictionary size of *HVP-RF* by cross validation on the *Wales-Small* dataset. The optimal dictionary size was 1000 though the original work used 300. We point out that the difference between the initial clustering of CONFIRM and *HVP-RF* is the choice of features used. CONFIRM uses features based on exemplar matching (Chapter 4), while *HVP-RF* uses BoVW features derived from SURF [3]. Both algorithms employ the same unsupervised similarity learning and use Spectral Clustering.

Dataset	Types	Forms
NIST† [11]	20	11,185
*WashPass**	2	2,000
Wales-Large†ˆ*	30	29,410
*Wales-Balanced*ˆ*	24	4,800
Wales-Small†ˆ*	11	6,354
PADeaths†ˆ*	5	4974

†Skewed type distribution ˆSimilar types
*Provided by Ancestry.com

Table 5.1: Summary of datasets used in experiments.

5.3 Datasets

In addtion to the standard benchmark *NIST* tax forms [11], we also evaluate over a number of proprietary datasets of historical forms with a variety of characteristics (see Table 5.1). Many prior works evaluate solely on *NIST* and report perfect classification accuracy [6, 16, 21, 32] or 100% pure clusterings [23, 35] on the SPDB 2 and/or SPDB 6 versions (we use both SPDB 2 & 6), showing that clustering and classifying this dataset is a solved problem. Because CONFIRM aims to cluster noisy forms in collections containing similar form types, additional datasets are needed.

The *WashPass* dataset contains two form types, with examples shown in Figure 5.1 at the end of the chapter. The two types are manifest lists for vessels and airlines respectively. These forms are from the 1940s and were typed with a typewriter, which means that the field values (e.g. person names, destinations, dates) are readable by OCR and thus are included as text line page elements detected by CONFIRM. We show that CONFIRM is able to ignore these as irrelevant features and produce good clusterings over these types.

The *Wales-** datasets (see Figure 1.1 for example images) are all subsets of the 1911 England and Wales Census. *Wales-Large* is used to demonstrate scalability in number of forms and number of types when the distribution of forms over types is skewed. *Wales-Balanced* shows a large number of types for a smaller number of forms per type, but with a balanced type distribution. *Wales-Small* is a subset with fewer number of types, but with a skewed

distribution. These three datasets contain very similar form types with some pairs of forms having identical or near identical rule line structure. Additionally, many types share column header text, making it difficult to distinguish forms based only on text lines or only on rule line.

PADeaths (see Figure 5.2 at the end of the chapter) is a collection of death certificates from the state of Pennsylvania. Four of the five form types differ by one or two fields, as fields were added or removed over a span of several years. Two of the types have less than 200 instances, while the largest type has over 2000 instances. This extremely skewed type distribution and high degree of type similarity makes this dataset very challenging.

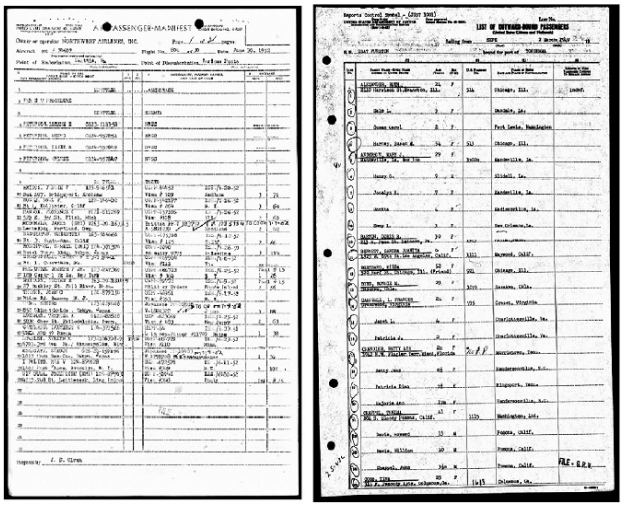


Figure 5.1: The 2 form types for the *WashPass* datasets. The typed field entries are readable by OCR, but useless for determining the form type. CONFIRM identifies what text is relevant to type identification by examining how text lines match between forms.

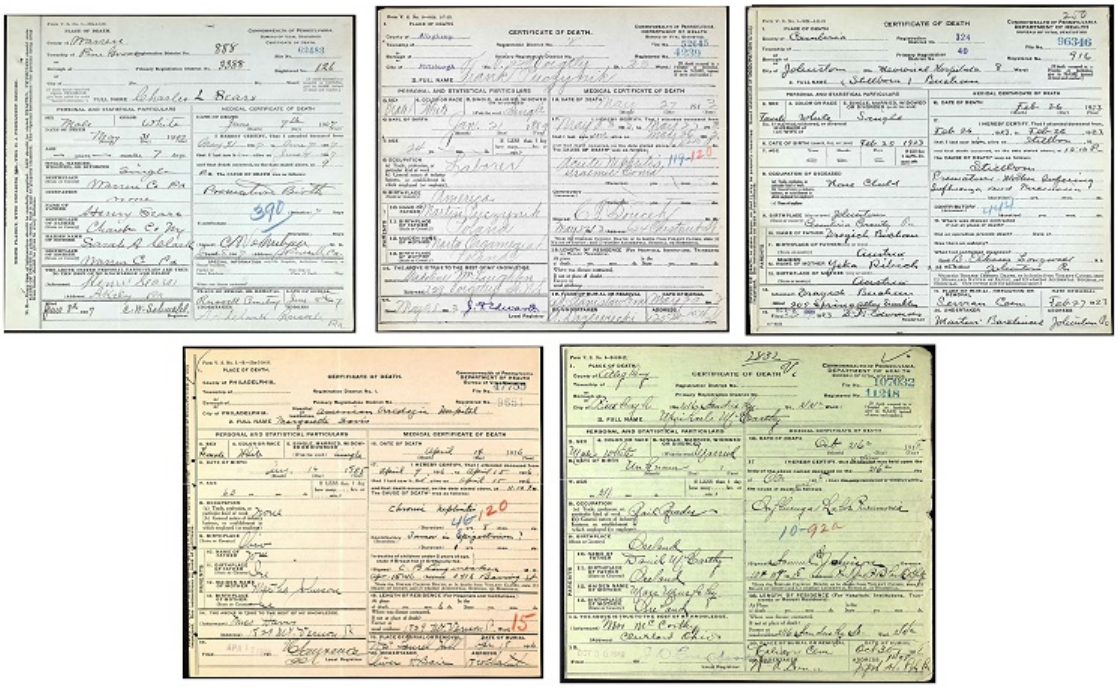


Figure 5.2: Example images of the 5 form types in *PADeaths* dataset. These types are highly similar, some differing by only a single field, making this a challenging dataset.

Chapter 6

Experiments

This chapter gives the setup, results, and discussion of the experiments undertaken to validate CONFIRM.

6.1 Overall

Our main experiment evaluates CONFIRM and variants across all six test datasets and provides comparison with the state-of-the-art *HVP-RF* model. We average purity scores across a wide range of K, i.e., $max(2, K^* - 5) \leq K \leq K^* + 5$, where K^* is the number of types in the ground truth labels. Results are shown in Table 6.1.

Overall, *C-Initial* provided an average 44% reduction in impurity compared to *HVP-RF* across all 6 datasets. We attribute this performance increase to the domain specific features derived from the robust matching of page elements. These features allow the salient structural patterns that differentiate similar form types to be learned.

C-No-Refine out-performed *C-Initial* by an average of 0.3 percentage points, showing that the bootstrap classification step marginally improves the initial clustering, even when no refinement is performed. In bootstrapping, LR discriminatively learns which critical page elements define each cluster, so it is able to reassign some forms that were originally assigned to clusters based on non-critical page elements.

C-Bootstrap similarly improves over *C-Refine*, though it does not outperform *C-Initial* in most cases. As explained in Chapter 5, *C-Refine* and *C-Bootstrap* create $K' > K$ clusters, so for this experiment, both models were run with very small K, so the resulting K' would

	NIST	*WashPass*	*Wales-Small*	*Wales-Balanced*	*Wales-Large*	*PADeaths*	Average
K Range	$15-25$	$2-7$	$6-16$	$19-29$	$25-35$	$2-10$	
C-Initial	**97.87**	95.57	92.56	84.72	88.97	87.52	91.20
C-Refine	96.09	**95.14**	94.32	**84.55**	88.48	85.88	90.74
C-Bootstrap	96.13	**95.44**	**94.52**	**84.88**	88.87	86.09	90.99
C-No-Refine	**97.87**	**95.87**	92.92	**84.99**	**89.50**	87.65	**91.47**
C-Text-Init	94.66	86.55	88.49	72.44	75.24	**90.64**	84.67
C-Line-Init	**97.84**	95.44	89.45	81.45	87.26	80.49	88.66
HVP-RF	97.66	89.80	88.61	69.57	77.37	81.37	84.06

Table 6.1: Purity of each dataset averaged over $max(2, K^* - 5) \leq K \leq K^* + 5$, where K^* is the true number of types. While *C-Refine* and *C-Bootstrap* create $K' \geq K$ clusters, only results where K' is in the appropriate range are included for fair comparison. Bolded entries indicate best performing models and are statitically significant according to pairwise Wilcoxon signed rank tests with significance measured at the $p < 0.05$ level.

fall into the evaluation range. This means that the initial clusterings for both models in this case were poor, leading to higher variance in results and slightly lower average performance.

For all datasets except *PADeaths*, *C-Line-Init* performed better than *C-Text-Init*, but not better than *C-Initial*. We believe this is because rule line detection is more accurate than OCR for noisy documents. This also shows that the combination of both types of page elements is essential for optimal performance.

While the highest performing model on each dataset is not always statistically significant (given the similarity between CONFIRM variants), *C-Initial*, *C-Refine*, *C-Bootstrap*, and *C-No-Refine* statistically outperformed *HVP-RF*. The only exception is on *NIST* (which is practically a solved problem), where only *C-Initial* and *C-No-Refine* were statistically better. *C-Refine* and *C-Bootstrap* had much higher variances than other models (e.g. *C-No-Refine*), so their pairwise ranks were not consistent, though their mean performance was often lower than that of *C-No-Refine*.

Model	Avg. Purity	ΔK
C-Initial	90.49	0
C-Refine	92.07	5.3
C-Bootstrap	92.33	5.3
HVP-RF	83.6	0

Table 6.2: Average Purity of 10 clusterings of each dataset when $K = K*$, along with the average number of additional clusters produced during cluster refinement. Cluster refinement increases cluster purity by 2.3% on average at the expense of a few more clusters.

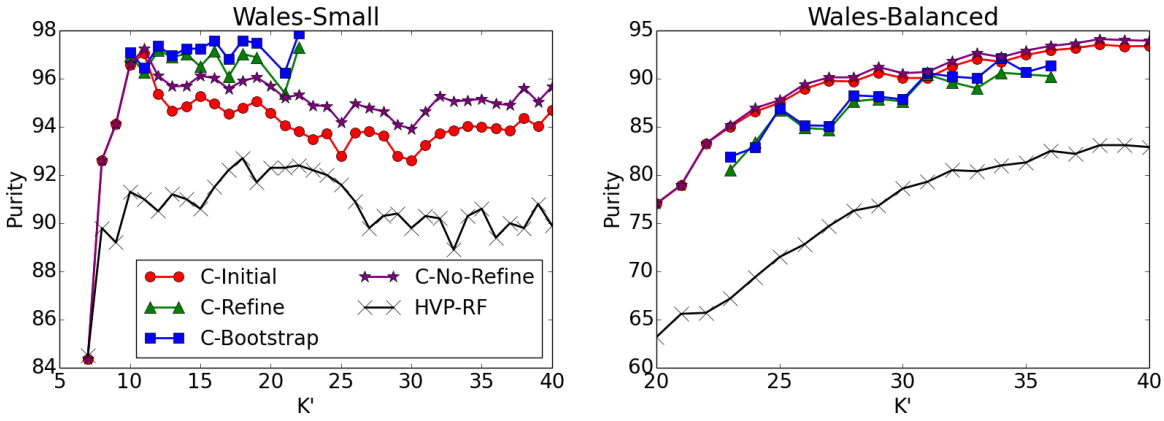


Figure 6.1: Purity vs number of resulting clusters (K'). For *Wales-Small*, it is better to under-cluster first and then refine the clusters. The opposite holds true for *Wales-Balanced*.

Table 6.2 shows summary results across all datasets for when $K = K*$ and K' is not constrained to be in any given range. *C-Refine* produces on average 5.3 additional clusters with an increase in purity of 2.3%. Depending on the end application, this tradeoff between number of clusters and cluster purity may be significant.

6.2 Refined Clusters

In these experiments, we examine cluster refinement more closely using the *Wales-Small* and *Wales-Balanced* datasets. We ran *C-Refine* and *C-Bootstrap* using $K^* - 5 \leq K \leq K^* + 5$, though K' sometimes falls outside this range. For comparison we ran *C-Initial*, *C-No-Refine*, and *HVP-RF* with $K \leq 40$ and the resulting plots are shown in Figure 6.1.

For *Wales-Small*, *C-Refine* gives a significant boost in purity over *C-Initial* for $12 \leq K' \leq 22$. There is a pair of similar types that are almost always clustered together by *C-Initial* (even with high K), but these types are frequently separated after cluster refinement. *C-Bootstrap* and *C-No-Refine* yield further marginal improvement over *C-Refine* and *C-Initial* respectively, which shows that the LR model is able to correctly classify some forms that were initially misclustered. This is significant because the assignment made by the LR model disagrees with the training label, which helps overcome the label noise problem inherent to bootstrapping.

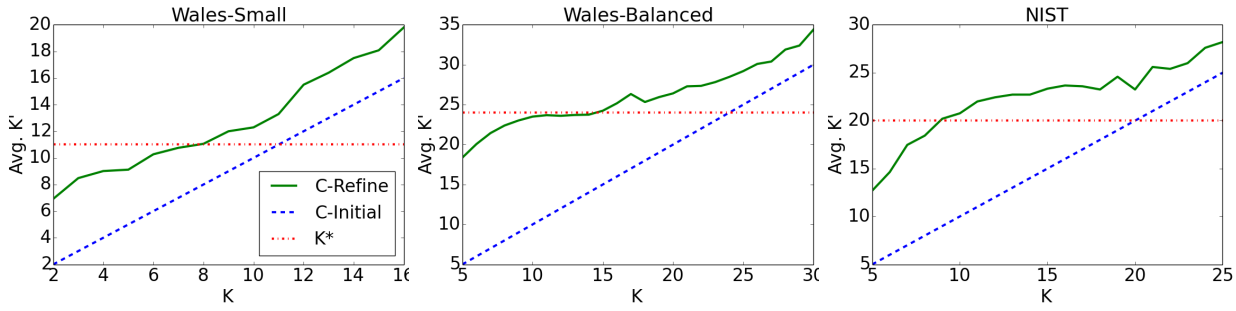


Figure 6.2: Average number of clusters (K') produced by cluster refinement for *Wales-Small* (left) and *Wales-Balanced* (right). If the user choses K to be too small, cluster refinement provides robustness by creating more clusters in a data-dependent fashion.

For *Wales-Balanced*, *C-Refine* does not improve the purity of *C-Initial* if a good value of K can be set by the user. However, in Figure 6.2, we show the relation between K and K' for *C-Refine*. When $K < K^*$, more clusters are produced than when $K > K^*$, which provides some robustness to the user chosen K. This is especially pronounced for *Wales-Balanced* and *NIST*, where if $\frac{K^*}{2} \leq K \leq K^*$, K' is often very close to K^*. This provides a good alternative to the standard method of choosing K based on a clustering metric such as Silhouette score [31].

6.3 Bootstrapping

In this experiment we show that CONFIRM is able to accurately assign novel instances to clusters in the bootstrapped classification stage (Section 3.4). While in previous experiments entire datasets are used in initial clustering, here we vary the number of initially clustered forms and measure how it affects the purity of the clustering induced (via bootstrapping) over all the data. We set $K = K^*$. The results are shown for *C-Initial*, *C-No-Refine*, and *C-Bootstrap* across all datasets in Figure 6.3.

The reported purity for *C-Initial* is measured only over the initially clustered subset, while results for *C-No-Refine* and *C-Bootstrap* are over the entire dataset. Because *C-Bootstrap* creates additional clusters during cluster refinement, we also report the average

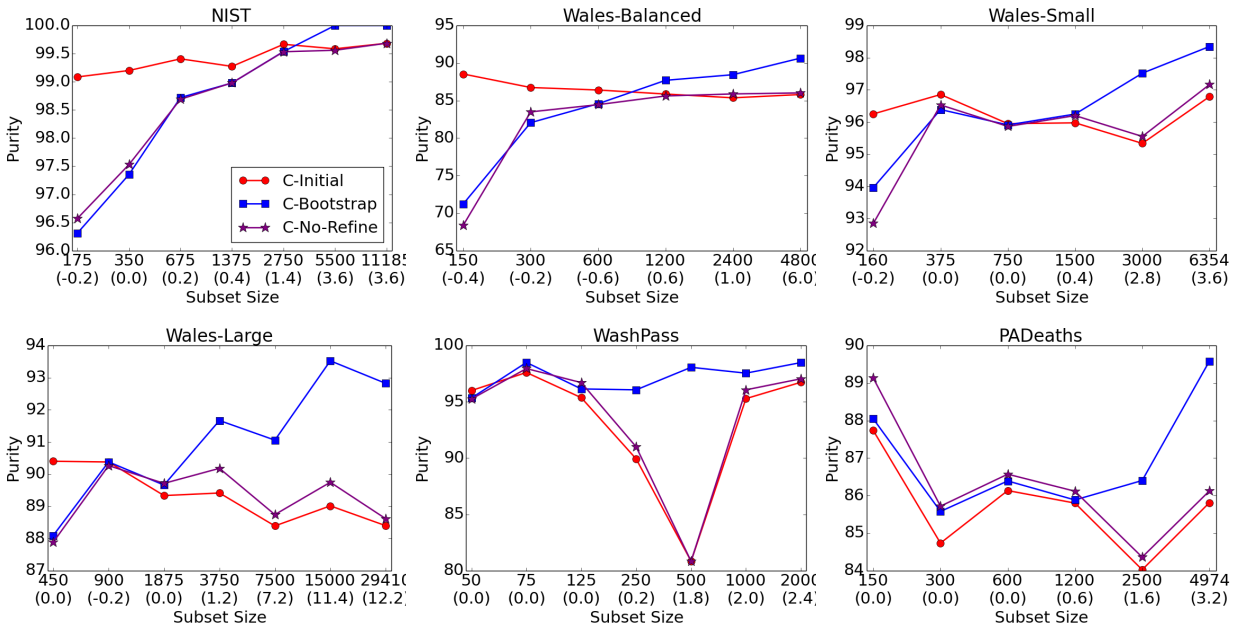


Figure 6.3: Average Purity as a function of the size of the initially clustered subset ($K = K^*$). Note that *C-Initial* is evaluated over only the initially clustered subset, while *C-No-Refine* and *C-Bootstrap* are evaluated over the entire dataset. The numbers in parenthesis along the x-axis indicate how many additional clusters on average were created by *C-Bootstrap* during cluster refinement.

number of additional clusters ($\Delta K = K' - K$) for each subset size in parenthesis below the x-axis.

For *NIST* and all *Wales* datasets, the trends are similar. With small initial subsets, *C-Bootstrap* and *C-No-Refine* have lower purity on the whole dataset than *C-Initial* does on the small initial subset. This is to be expected because the LR classifier will not generalize well if it has few examples from each target type to learn from. This problem is magnified when the distribution of types is skewed because clusters representing infrequent types are smaller than average. For *NIST*, *Wales-Large*, *Wales-Small*, and *PADeaths*, the most frequent type is more than 10x more frequent that the least frequent type.

Nevertheless, at around 1000 initially clustered instances (less than 3% of *Wales-Large*), we observed good generalization performance, regardless of the actual dataset size. When not all data is used for initial clustering, the cluster purities of *C-No-Refine* and *C-Bootstrap* are good estimates of generalization accuracy. This means we could apply *C-No-Refine* or

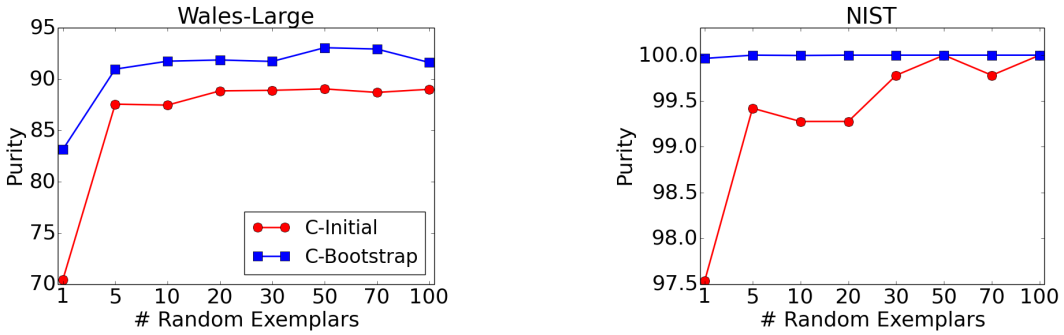


Figure 6.4: Average Purity of vs. number of random exemplars, E, for *Wales-Large* (left) and *NIST* (right) datasets. With the exception of $E = 1$, all choices of E perform similarly though there is a slight increase in purity as E increases.

C-Bootstrap to much larger super sets of these datasets and expect the same cluster purity. This evidences our previous claim that the number of initially clustered forms need only depend on the distribution of types, not on the size of the dataset. Note that for larger subsets, *C-Bootstrap* outperforms *C-No-Refine* because more clusters are created.

For *WashPass*, sometimes the initial clustering is extremely poor, perhaps due to the irrelevant typeset field entries and the small subset sizes. However, cluster refinement is able to correctly parition the clusters in each case, leading to robust performance for *C-Bootstrap* at the expense of a couple extra clusters. With *PADeaths*, no clear trend emerges, but *C-No-Refine* is able to generalize even at very small subsets, and *C-Bootstrap* is able to perform well when there is a larger amount of training data.

6.4 Choosing Exemplars

These experiments show that choosing exemplars randomly during initial clustering performs only slightly worse than choosing exemplars using an oracle (i.e. using the ground truth labels). The oracle method is not a feasible option for real applications, but it provides an upper bound on expected performance by intelligently choosing exemplars. We also show that performance is robust to the number of exemplars chosen.

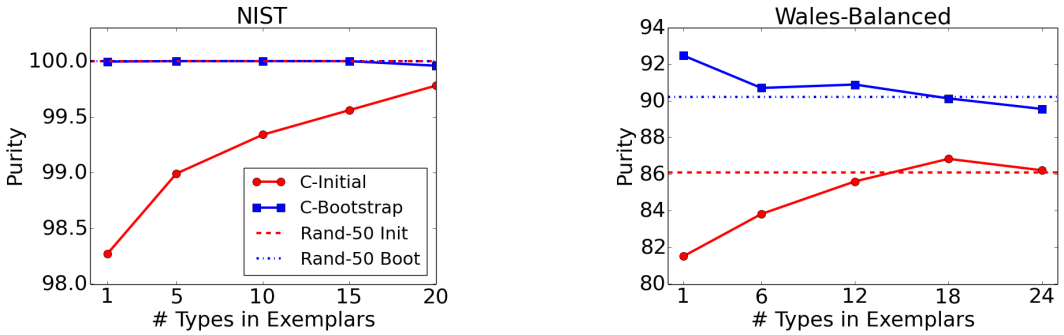


Figure 6.5: Average Purity of oracle method of choosing exemplars vs randomly choice for $E = 50$ exemplars (horizontal lines). For *NIST*, both Rand-50 baselines overlap at $Purity = 100$.

6.4.1 Random Exemplars

First, we examine how the number of random exemplars affects performance with $E \in \{1, 5, 10, 20, 30, 50, 70, 100\}$. Plots of cluster purity vs E for the *Wales-Large* and *NIST* datasets are shown in Figure 6.4. Results on the other datasets exhibit similar trends and are omitted for brevity.

As expected, using $E = 1$ results in significantly worse performance because the page elements of a single form are not representative of the page elements in the whole collection. Performance becomes reasonable with $E = 5$, and steadily climbes as more exemplars are used. While the dimensionality of the feature vectors used for initial clustering increases with E, this is not a problem (as with Euclidean distance based clustering) due to unsupervsied structural similarity learning. As more exemplars are used, more significant structural patterns are included, though we observe diminishing returns in terms of resulting purity.

6.4.2 Oracle Exemplars

First, we compare choosing E random exemplars to choosing E exemplars using the ground truth labels. For K^* form types, we choose $T \leq K^*$ types and at least $\lfloor \frac{E}{T} \rfloor$ forms of each type, so we have E total exemplars. If we choose $T = K^*$, we obtain the optimal situation where all form types are equally represented in the exemplars. When exemplars are chosen at random, there is a chance that not all form types are included among the exemplars. By choosing

$T < K^*$, we observe whether excluding some form types from the chosen exemplars hurts performance. In our experiments, we set $E = 50$, $K = K^*$, and $T \in \{1, \frac{K^*}{4}, \frac{K^*}{2}, \frac{3K^*}{4}, K^*\}$. Results are presented in Figure 6.5.

For *Wales-Balanced*, the purity of *C-Initial* increases with T and levels out slightly above a random choice of 50 exemplars. When the classes are balanced, the chance of omitting a significant percentage of types in the exemplars is small, and empirically we see no advantage in ensuring that all types are represented. If some types are omitted, similar page elements can be found in other similar types and matching against those page elements is sufficient to distinguish the form type omitted from the exemplars. The downward trend of *C-Bootstrap* can be attributed to fewer clusters being produced in refinement as T increases.

Chapter 7

Conclusion and Future Work

In this work we have presented CONFIRM, a novel algorithm for clustering noisy form images. CONFIRM outperforms the current state-of-the-art by 44% impurity reduction across 6 datasets by using domain specific features based on novel text and rule line matching algorithms. Five of these datasets are composed of noisy historical forms with similar types that are challenging to distinguish in a clustering setting. We have also presented a strategy to provide robustness to the user set number of clusters K by adaptive subclustering. CONFIRM scales well to large datasets that are difficult to cluster directly by training a supervised classifier using an initial clustering as training data. Our experiments show that if 1000 instances are used for initial clustering, the classifier is able to generalize (at the same or better purity) to a larger superset of the data.

The framework contained in CONFIRM is also extensible in that different clustering algorithms and additional features may be used. We leave as future work the evaluatation of other sets of features and corresponding matching and merging algorithms. For example, rectangular regions of background also encode document spatial layouts and could be included as additional page elements. We point out that traditional vectoral features, such as BoVW, can be trivially included into this framework using element-wise aboslute distance for producing match vectors and element-wise averaging for merging. Because of the unsupervised similarity learning, this multi-modal data is fused together despite the various source representations. Depending on the features and nature of the matching algorithm, CONFIRM could be extended to other tasks such as generalized document clustering.

References

[1] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[2] Olivier Augereau, Nicholas Journet, Anne Vialard, and Jean-Philippe Domenger. Improving classification of an industrial document image database by combining visual and textual features. In *Document Analysis Systems (DAS)*, pages 314–318. IEEE, 2014.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[4] Sukalpa Chanda, Katrin Franke, and Umapada Pal. Clustering document fragments using background color and texture information. In *IS&T/SPIE Electronic Imaging*, pages 82970U–82970U. International Society for Optics and Photonics, 2012.

[5] Nawei Chen and Dorothea Blostein. A survey of document image classification: problem statement, classifier architecture and performance evaluation. *International Journal of Document Analysis and Recognition (IJDAR)*, 10(1):1–16, 2007.

[6] Siyuan Chen, Yuan He, Jun Sun, and Satoshi Naoi. Structured document classification by matching local salient features. In *International Conference on Pattern Recognition (ICPR)*, pages 653–656. IEEE, 2012.

[7] Robert Clawson, Kevin Bauer, Glen Chidester, Milan Pohontsch, Douglas Kennard, Jongha Ryu, and William Barrett. Automated recognition and extraction of tabular fields for the indexing of census records. In *IS&T/SPIE Electronic Imaging*, pages 86580J–86580J. International Society for Optics and Photonics, 2013.

[8] Colin de la Higuera and Francisco Casacuberta. Topology of strings: Median string is np-complete. *Theoretical Computer Science*, 230(1):39–48, 2000.

[9] Markus Diem, Florian Kleber, Stefan Fiel, and Robert Sablatnig. Semi-automated document image clustering and retrieval. In *IS&T/SPIE Electronic Imaging*, pages 90210M–90210M. International Society for Optics and Photonics, 2013.

[10] Michelangelo Diligenti, Paolo Frasconi, and Marco Gori. Hidden tree markov models for document image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(4):519–523, 2003.

[11] D. L. Dimmick, M. D. Garris, and Wilson C. L. Nist structured forms reference set of binary images (sfrs). Online, 1991. URL `http://www.nist.gov/srd/nistsd2.cfm`.

[12] Kuo-Chin Fan, Yuan-Kai Wang, and Mei-Lin Chang. Form document identification using line structure based features. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 704–708. IEEE, 2001.

[13] Andres Folleco, Taghi M Khoshgoftaar, Jason Van Hulse, and Lofton Bullard. Identifying learners robust to low quality data. In *IEEE International Conference on Information Reuse and Integration (IRI)*, pages 190–195. IEEE, 2008.

[14] Andres Folleco, Taghi M Khoshgoftaar, Jason Van Hulse, and Lofton Bullard. Software quality modeling: The impact of class noise on the random forest classifier. In *Evolutionary Computation. (IEEE World Congress on Computational Intelligence)*, pages 3853–3859. IEEE, 2008.

[15] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.

[16] Albert Gordo, Florent Perronnin, and Ernest Valveny. Large-scale document image retrieval and classification with runlength histograms and binary embeddings. *Pattern Recognition*, 46(7):1898–1905, 2013.

[17] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. Evaluation of deep convolutional nets for document image classification and retrieval. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 991–995. IEEE, 2015.

[18] Pierre Heroux, Sebastien Diana, Arnaud Ribert, and P Trupin. Classification method study for automatic form class identification. In *International Conference on Pattern Recognition (ICPR)*, volume 1, pages 926–928. IEEE, 1998.

[19] Luke AD Hutchison and William A Barrett. Fourier–mellin registration of line-delineated tabular document images. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):87–110, 2006.

[20] Yasuto Ishitani. Model matching based on association graph for form image understanding. In *International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 287–292. IEEE, 1995.

[21] Le Kang, Jayant Kumar, Peng Ye, Yi Li, and David Doermann. Convolutional neural networks for document image classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3168–3172. IEEE, 2014.

[22] Florian Kleber, Markus Diem, and Robert Sablatnig. Form classification and retrieval using bag of words with shape features of line structures. In *IS&T/SPIE Electronic Imaging*, pages 902107–902107. International Society for Optics and Photonics, 2013.

[23] Jayant Kumar and David Doermann. Unsupervised classification of structurally similar document images. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1225–1229. IEEE, 2013.

[24] Jayant Kumar, Peng Ye, and David Doermann. Learning document structure for retrieval and classification. In *International Conference on Pattern Recognition (ICPR)*, pages 1558–1561. IEEE, 2012.

[25] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[26] Laurence Likforman-Sulem, Abderrazak Zahour, and Bruno Taconet. Text line segmentation of historical documents: a survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(2-4):123–138, 2007.

[27] Simone Marinai, Emanuele Marino, and Giovanni Soda. Tree clustering for layout-based document image retrieval. In *Second International Conference on Document Image Analysis for Libraries*, pages 253–262. IEEE, 2006.

[28] George Nagy and Daniel Lopresti. Form similarity via levenshtein distance between ortho-filtered logarithmic ruling-gap ratios. In *IS&T/SPIE Electronic Imaging*, pages 902106–902106. International Society for Optics and Photonics, 2013.

[29] H Nielson and W Barrett. Consensus-based table form recognition of low-quality historical documents. *International Journal of Document Analysis and Recognition (IJDAR)*, 8 (2-3):183–200, 2006.

[30] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Empirical Methods in Natural Language Processing (EMNLP)*, volume 7, pages 410–420. Citeseer, 2007.

[31] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[32] Marçal Rusiñol, Volkmar Frinken, Dimosthenis Karatzas, Andrew D Bagdanov, and Josep Lladós. Multimodal page classification in administrative document image streams. *International Journal on Document Analysis and Recognition (IJDAR)*, pages 1–11, 2014.

[33] Hiroshi Sako, Minenobu Seki, Naohiro Furukawa, Hisashi Ikeda, and Atsuhiro Imaizumi. Form reading based on form-type identification and form-data recognition. In *International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 926–926. IEEE Computer Society, 2003.

[34] Prateek Sarkar. Image classification: Classifying distributions of visual features. In *International Conference on Pattern Recognition (ICPR)*, volume 2, pages 472–475. IEEE, 2006.

[35] Eric Saund. A graph lattice approach to maintaining dense collections of subgraphs as image features. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 1069–1074. IEEE, 2011.

[36] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[37] Christian Shin, David Doermann, and Azriel Rosenfeld. Classification of document pages using structure-based features. *International Journal on Document Analysis and Recognition (IJDAR)*, 3(4):232–247, 2001.

[38] Marcel Simon, Erik Rodner, and Joachim Denzler. Fine-grained classification of identity document types with only one example. In *International Conference on Machine Vision Applications (MVA)*, pages 126–129. IEEE, 2015.

[39] Toyohide Watanabe. Guideline for specifying layout knowledge. In *Electronic Imaging'99*, pages 162–172. International Society for Optics and Photonics, 1999.