ArchivIT Documentation

April 2014

The University of British Columbia

Media and Graphics Interdisciplinary Centre

Supervisor: Dr. Victoria Lemieux

Mahshid Zeinaly

Contents

Overview

In this project Visual Analytics is applied to Archival Study domain to assist archivists in their domain tasks [5]. The software is designed based on an in-depth filed study with two archivists to understand their cognitive process [4]. The study suggested that in order to categorize their files, archivists follow the structure of the forms [4]. The Archi-V design enhances this domain process by enabling the users to make categories and manually move files into appropriate categories. The tool guides the user to classify the files in two different ways:

- Thumbnail representation that displays the thumbnail of each file showing the form structures to the users at a glance. This feature is divided into to modules: a python program that creates the thumbnails from pdfs, and a web interface that represents the thumbnails and allows the user to manually move them.
- Automatic default categorization that categorizes the files into different classes also according to their form structures. The above feature then can help the user to manually change the files as needed based on their domain knowledge and experiences. Please note that this feature has not been implemented in the current version of the tool.

The interface is consistent with the ICA-AtoM open source project [2] and meant to be used as part of that project.

The Thumbnail Creator Module

The ThumbinalCreator program manipulates information on PDF files in order to make thumbnails for the purpose of this project. In Archi-V project, the domain specialists are looking for a document outlines and templates in order to classify the documents according to their patterns.

The ThumbnailCreator program is written in Python that is a high-level programming language. The program also uses pdfminer Python library (see Listing 1). This library allowes us to extract information from PDF documents.

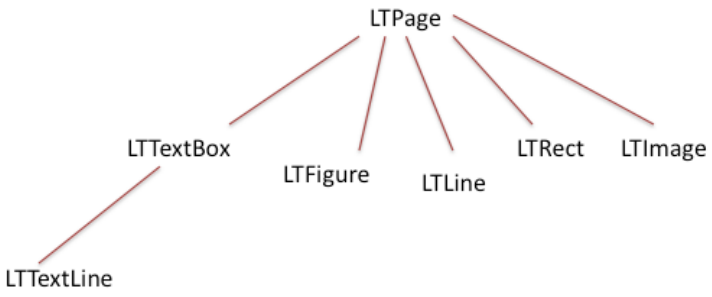
Listing 1: Python Script for adding PDF Miner features into the program.

```
from django.conf.urls import patterns, url
from documentViewerApp import views
from django.views.static import serve
from django.conf import settings

urlpatterns=patterns('',
        url(r'^$', views.index, name='index'),
        url(r'^.*categoriesall.*$',views.CategoriesAll, name='categoriesall'),
        url(r'^.*thumbnailmaker.*$',views.ThumbnailMaker, name='thumbnailmaker'),
)
```

PDFMiner can give us information about the page layout objects. Each “LTPage” contains objects such as lines, images, rectangles, etc within itself. The Thumbnailcreator Python program generates a Matrix with the size of the original pdf document page size. It fills each cell of the Matrix with the background color specified by the user. The program then goes to each element of the “LTPage” and changes the colours of the cells in the range of that object’s layout. Generally, this program searches for an ”hbox” that can be either “LTTextBox”, “LTFigure”, “LTLine”, “LTRect”, “LTImage”.

Image below shows the layout structure found in PDFMiner documentation page.



For example, if an “hbox” is an instance of “LTTextLineHorizontal”, then the program extracts the top left corner and the bottom right corner of the box around that text. Then it uses the desired colour for the texts, specified in the user parameters, and changes the Matrix cell information in that range. Using the PIL library, the program then makes an image with the desired thumbnail size out of the Matrix.

How to Execute the Program

To run the program, you need to change the two parameters INPUT_DIR_NAME, and OUTPUT_DIR_NAME addresses to your desired folder locations. The program reads all the files it found in INPUT_DIR_NAME (PDF files), and creates and saves thumbnail jpg files into OUTPUT_DIR_NAME.

Another parameter is the size of jpg images that you want to create. The two parameters THUMBNAIL_WIDTH and THUMBNAIL_HEIGHT are set to 128 as default width and height.

The PAGE_COUNT parameters shows the number of pages in a PDF that you want to create thumbnails from. For example, if it is set to 1, the program will only create the thumbnail of the first page.
There are also parameters to set the line widths, line spaces, and the colours of the thumbnail layout shapes. The default colour is set to black shapes (lines ,...) and white background.
Listing 2 shows a list of parameters that the user can change to customize the output of this program.

Listing 2: The parameters

```
INPUT_DIR_NAME='./pdfs/'
OUTPUT_DIR_NAME='./'+'Thumbnails'+'/'

THUMBNAIL_WIDTH=128
THUMBNAIL_HEIGHT=128

PAGE_COUNT=4

TEXT_LINE_WIDTH_RATIO=0
LINE_LINE_WIDTH_RATIO=2
RECT_LINE_WIDTH_RATIO=3

BACKGROUND_COLOR=[int(255),int(255),int(255)]
TEXT_COLOR=[int(0),int(0),int(0)]
LINE_COLOR=[int(0),int(0),int(0)]
RECT_COLOR=[int(0),int(0),int(0)]
IMG_COLOR=[int(100),int(100),int(100)]
FIG_COLOR=[int(100),int(100),int(100)]
```

On Terminal, the user can type $ python PDF2Thumbnail_ImageExtraction.py and that will create the destination folder containing the thumbnail files.

The Program User Interface

The program interface is written in a web2.0 application framework called Django [1]. In order to install Django, make sure Python 2.6.5 or upper is installed properly in your computer. Download and install the installation package from Django website. There might be some troubles while installing if the Python installation is in a package that your PATH does not look. One way of resolving this issue is to create a synlink between where the Python folder is to where it is looking to find it in the PATH variable(/usr/local/lib in my computer).

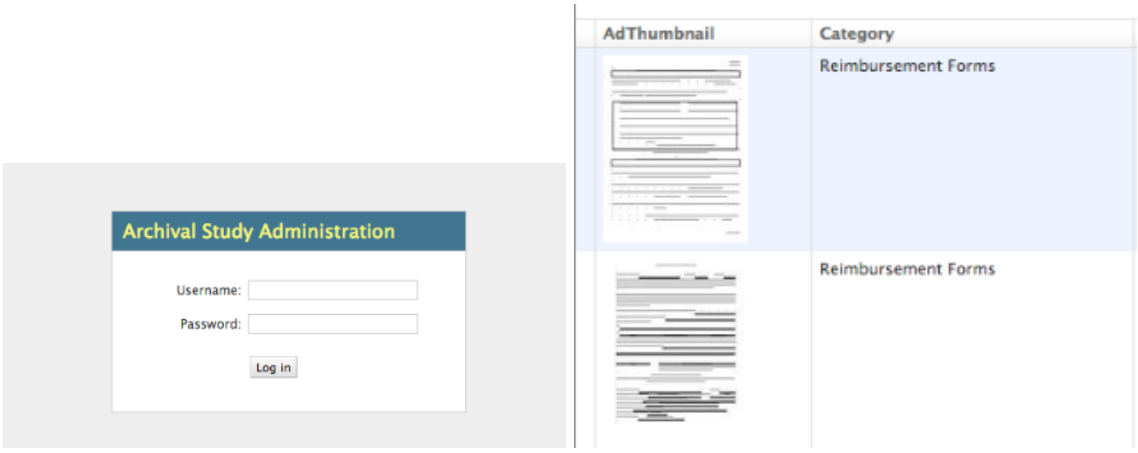
The Archi-V Project Sites

As every section of any projects in Djnago has its own App, we needed to create an App for viewing and manipulating the documents that are stored in the database. Current version of the Archi-V web application has two Apps: 1. The Admin site 2. The Document Viewer App.

Figure 1: Archi-V front page.

The Admin Site

This part of the application is for the admin user to be able to manually change the database. The admin can add or delete records to the database tables. Django Admin App interface is part of Django framework but it is not activated by default. We also need to register the admin interface with our own database model.



(a) The admin site front page (b) The admin user can view and change the database

Figure 2: The admin site

The Document Viewer Site

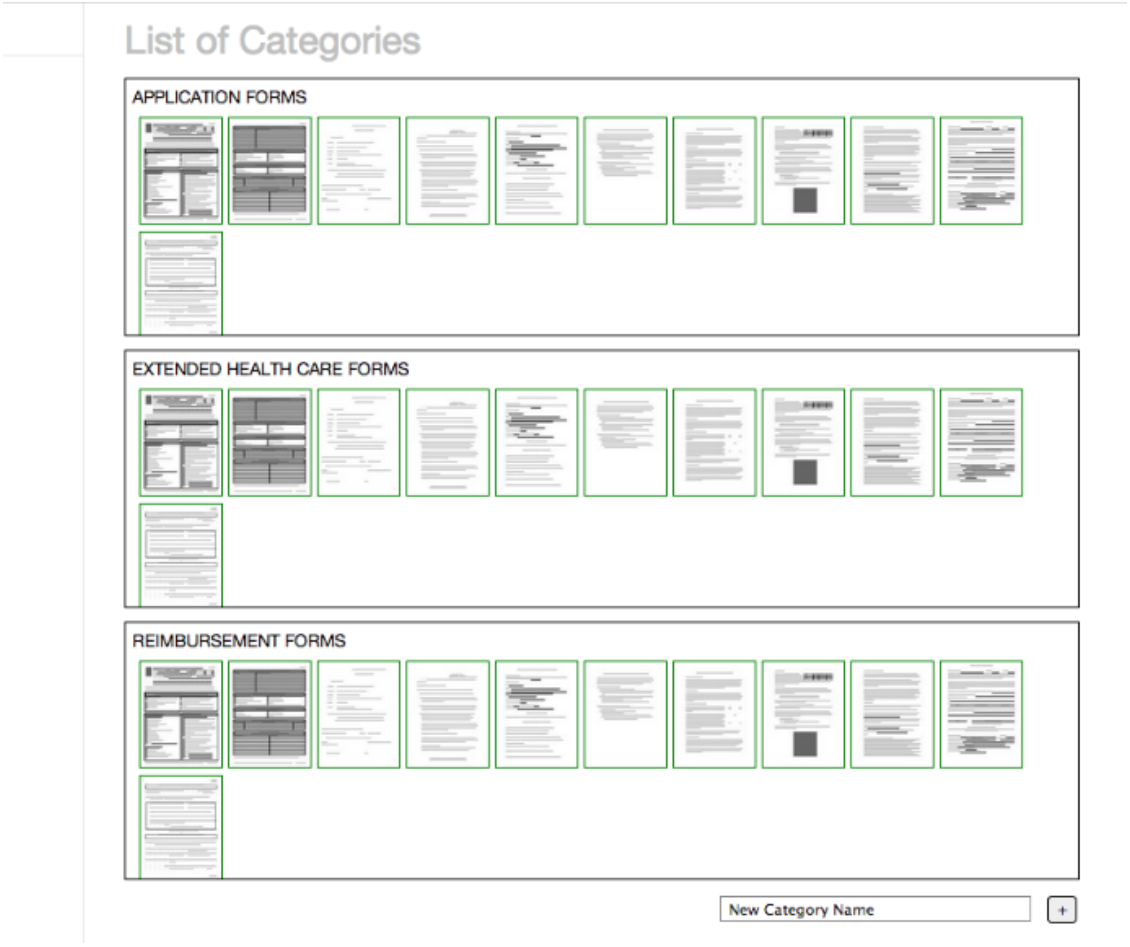


Figure 3: The system shows the categories created by the user and/or saved into the database. In this demo, the system reads all the files and does not read only the files that belong to each category (a bug that needs to be fixed)

The front page of the Arch-V site has some information about the project as well as the menu to go to the different parts of the system. One of the menu items is the Admin site that described above. The category page shows the different categories as well as the files inside each of them. The user can drag and drop the files into different categories. The plus button is designed so that the user can make a new category and give that a name. This functionality is still under construction. Also for future improvement, when the automatic categorization module is developed, then all the system suggested categories need to be shown. The user then can give them a name, and that will add them to the database. Also, if the user manually change a file category, the file's class need to remain unchanged (the system can not define a new automatic category for the file), until the user makes further changes.
Figure 4 shows another feature of the system. The ThumbnailCreator Python program needs to be integrated to Archi-V system. This will allow the user to create the thumbnails from the website rather than manually change the parameters and run the program. There is also an interface for the user to be able to upload the thumbnails to the database. To do so, the user needs to enter the admin password. The functionality of this feature has not been implemented and only the interface is put on the website.

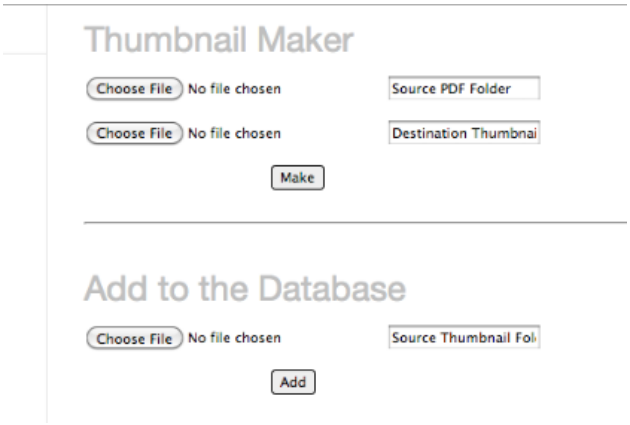


Figure 4: This page allows the user to specify the input and output locations to make thumbnails and add them into database directly from the website.

How to Run the Program

Django framework comes has a lightweight built-in web server that you can use to test the program while developing. It is not recommended to use this server for deployment purpose.
Once the Django is installed, on Terminal, cd to the Django project folder, on Terminal type $ cd cd DjangoWebApplication/ This folder should contain manage.py file.

On terminal type $ python manage.py runserver and this will start the web server on the computer. To exit from the server type $ ctrl + C While the server is running, you can type $ http://127.0.0.1:8000/documentViewerApp/ to go to the main index page.

Project Settings

The project has a main folder called "DjangoWebApplication" that contains a folder for the "DocumentViewerApp" and a separate folder called "ArchivalStudy". Inside he "ArchivalStudy" folder, settings.py needs to be set according to our machine destinations. Listing 3 is an example of my settings to run the project on my Mac computer.

Listing 3: Django project settings

```
# Absolute filesystem path to the directory that will hold user-uploaded files.
PROJECT_ROOT = os.path.dirname(os.path.realpath(__file__))

# URL that handles the media served from MEDIA_ROOT. Use a trailing slash.
MEDIA_ROOT = os.path.join(PROJECT_ROOT, '/media/')
MEDIA_URL = '/media/'
ADMIN_MEDIA_PREFIX='/static/admin/'

# Absolute path to the directory static files should be collected to.
STATIC_ROOT = os.path.join(PROJECT_ROOT, '/static/')

# URL prefix for static files.
STATIC_URL = '/static/'

# Additional locations of static files
STATICFILES_DIRS = ()
```

```
ROOT_URLCONF = 'ArchivalStudy.urls'

# Python dotted path to the WSGI application used by Django's runserver.
WSGI_APPLICATION = 'ArchivalStudy.wsgi.application'

TEMPLATE_DIRS = ('/Users/mahshid/Django_Web_Application/ArchivalStudy/templates')
```

The Project Architecture

The project folder structure is shown in figure 5. We already looked at the settings.py in order to run the program. urls.py contains information about the project urls: documentViewerApp, Admin, and the media folder that contains the image files. Listings 4 shows how I specified these there urls in urls.py inside "ArchivalStudy" folder.

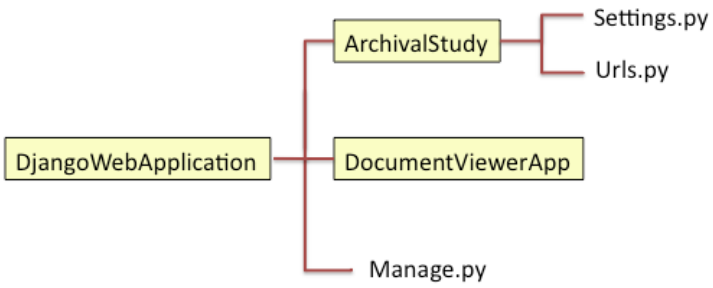


Figure 5: Django web application project structure.

Listing 4: Django project urls

```
from django.conf.urls import patterns, include, url
from django.conf import settings

# The next two lines enables the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
 url(r'^documentViewerApp/',include('documentViewerApp.urls')),
 url(r'^admin/', include(admin.site.urls)), # to enable the admin:
 url(r'^media/(?P<path>.*)$','django.views.static.serve',
     {'document_root':settings.MEDIA_ROOT}),
)
```

The Django framework is based on Model-View-Controller (MVC) design that separates content from presentation. In Django case, "Mode"l is the database classes, "View" describes the data that gets presented to the user, so it basically has quires to the database. "Url" means the webpages that the user can go to from the browser. "Templates" means the user interface representations (html files). In order to better separate the codes, we need to make an static folder and put all the JavaScripts and CSS files into that folder. The Django project knows where the static folders are and html files inside the "template" folder can read JS

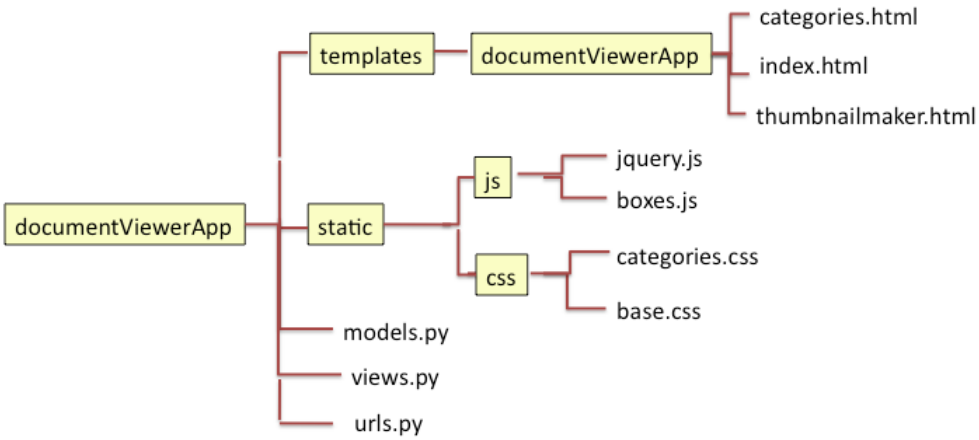


Figure 6: documentViewerApp structure.

and CSS files. Figure 6 shows the structure within the "documentViewerApp" folder and how the project enables us to separate the codes into appropriate sections.
In the next following sections, I described how the "documentViewerApp" is developed following the above structure.

The Models

Inside model.py, we defined the two database tables, one for holding the files, and the other one for the categories. Listing 5 shows that we created a table called "Document" and each row in that table has a name, a pdfFile attached to it, and the thumbnail of the pdf attached to it. The database admin is going to show an image for each thumbnails in the "Document" table. An item also has a category which is a ForeignKeyto the "Category" table. The "Category" table has a name and a description for the category.

Listing 5: documentViewerApp Model

```
from django.db import models

class Document(models.Model):
    name=models.CharField(max_length=200)
    pdfFile=models.FileField(upload_to='original pdf file')
    thumbnail=models.ImageField(upload_to='thumbnail image')
    def admin_thumbnail(self):
        return u'<img src="%s" />' % (self.thumbnail.url)
    category=models.ForeignKey('Category')
    def __unicode__(self):
        return self.name

class Category(models.Model):
    name=models.CharField(max_length=200)
    description=models.TextField(blank=True)
    def __unicode__(self):
        return self.name
```

The Views

The View is defined inside views.py that creates all the necessary queries to the database defined by methods and pass these information into templates that are going to render them. Listing 6 shows one example of a view that is going to query all the categories and their documents from the database and shows that that categories.html is going to use that information.

Listing 6: documentViewerApp View

```
from django.shortcuts import render_to_response, render
from django.template import RequestContext, Context, loader
from documentViewerApp.models import Category, Document
from django.http import HttpResponse

def CategoriesAll(request):
     categories=Category.objects.all().order_by('name')
     documents=Document.objects.all()
     context={'categories':categories,'documents': documents}
     return render_to_response('documentViewerApp/categories.html',
                    context,context_instance=RequestContext(request))
```

The response object will have “categories” variable as well as “documents” variables. This object is going to pass into “categories.html ” template for display. The “categories” variable contains all the categories (from Category table in Model (database table)) sorted by name. The “documents” variable contains all the items in the Document Model (database table). These variable names will be used inside the template files (htmls).

The Urls

Listing 7 contains examples of the pages that we go to from “documentViewrApp”. This example shows how the view and url are connected.

Listing 7: documentViewerApp Url

```
from django.conf.urls import patterns, url
from documentViewerApp import views
from django.views.static import serve
from django.conf import settings

urlpatterns=patterns('',
        url(r'^$', views.index, name='index'),
        url(r'^.*categoriesall.*$',views.CategoriesAll, name='categoriesall'),
        url(r'^.*thumbnailmaker.*$',views.ThumbnailMaker, name='thumbnailmaker'),
)
```

Table 1 shows the url address of associated pages in the web application. Note that it is not the name of the urls that will put in the address. It i

Table 1: The relationship between url names and page addresses.

url name	url address
'index'	http://127.0.0.1:8000/documentViewerApp/
'categoriesall'	http://127.0.0.1:8000/documentViewerApp/categoriesall
'thumbnailmaker'	http://127.0.0.1:8000/documentViewerApp/thumbnailmaker

The Templates

The template files consists of html files that they load css and javasrcipt files that are stored inside the static folder. Inside the settings.py in project root, we already set where this folder is going to be. Listing 8 shows how an html file can load the static files in the html header. This will copy all the contents of those static files at the beginning of that html file.

Listing 8: documentViewerApp Template (css and javascript)

```
<html>
<head>
{% load staticfiles %}
  <link rel="stylesheet" type="text/css" href="{% static "css/base.css" %}" />
  <link rel="stylesheet" type="text/css" href="{% static "css/categories.css" %}"/>

  <script src="{% static "js/jquery.js" %}" type="text/javascript"></script>
  <script src="{% static "js/boxes.js" %}" type="text/javascript"></script>
</head>
<body>
...
</body>
</html>
```

In this project, I have used jquery [3] library in order to be able to access to the DOM elements easily. This library also helps to resolve problems with when the DOM is ready but we can not access to the elements because of the delay in rendering.
Listing 9 is one example for representing how a template file can work with variables defined in a view file using the Django Template language [1]. In this example, both "categories" and "documents" variables are used.

Listing 9: documentViewerApp Template (html)

```
<div id="content">
<h1>List of Categories</h1>
     {% block extrahead %}

{% if categories %}
     {% for c in categories %}
     <div id="container">
        <div>
       {{c.name|upper }}
        </div>
      <div id="imagecontainer">
     {% for d in documents %}
```

```
                <img  id="img" src="{{MEDIA_URL}}{{d.thumbnail}}" />
     {% endfor %}
      </div>
     </div>
     {% endfor %}
{% else %}
     <p> No Categories Found. </p>
{% endif %}

{%endblock %}
```

Note that it is recommended that if the project consists of many Apps, we organize the files in the template folder so that each html file goes into a folder with a name of that App. Only the index.html and base.html (that are going to be included in each html), as well as error handling pages will be placed directly inside the templates folder.

Linking Admin App and the Model

The admin app interface is already built in the Django project and we only need to uncomment some lines to activate that interface. We also need to register the interface to our model data tables. Listing 10 shows how inside admin.py, we registered the two data tables that we created in our Model. In this part of the code, we created two classes and linked them to the two database classes that we had.

Listing 10: Archi-V project : the Admin App

```
from django.contrib import admin
from documentViewerApp.models import Category,Document
class CategoryAdmin(admin.ModelAdmin):
     list_display=('name','description')
     search_fields=['name']

class DocumentAdmin(admin.ModelAdmin):
     list_display=('name','pdfFile','admin_thumbnail','category','thumbnail')

admin.site.register(Category,CategoryAdmin)
admin.site.register(Document,DocumentAdmin)
```

Document Classification

This section provides the details of our classification algorithm that is going to be used for automatic classification to guide the domain users. In this Visual Analytics tool we need to guide the tool users to group the documents into different categories. The software purpose is to automatically classify the documents and display the results to the user. The domain analysts can move the document between the categories by looking at the document thumbnails and based on their domain knowledge and expertise.
The classification process consists of several steps:

Feature Extraction

The goal is to use spacial information about the layouts of the documents. The information that we are going to use is lines, boxes and images. We decided to divide each document into four equal subsections. So for each feature definition, we will have four features. Bellow is our initial feature identifications:

- line density. We have information about the line positions in the Thumbnail Creator module.
- box density
- box size - mean
- box sizes - standard deviation
- image density

The above features are defined for each portion, so we will have 20 features per document. At the end of this process, we will have a [20 (feature dimensions) by the number of documents] matrix containing the information about the features across all the documents.

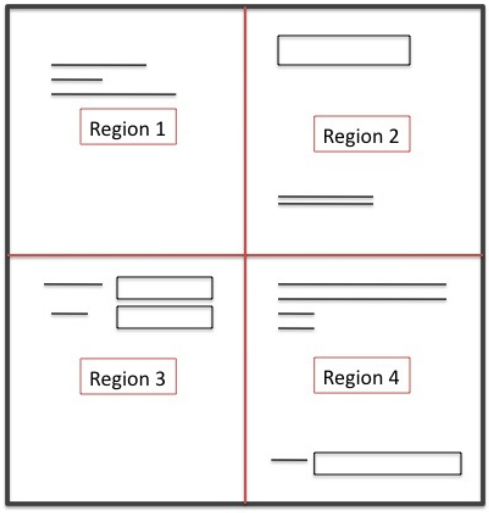


Figure 7: The Features are defined for each part of the document.

Feature Selection

In order to understand which of the defined features (dimensions) can separate the documents, we can use genetic algorithms or built-in random forest, and eliminate or keep some of the identified features.

Classification

The feature matrix as well as the class labels for the documents are input to the classification algorithm. At this point, we decided to use SVM or Random Forest. The implementation of these algorithms usually contains the valuation module as well.

Predication

At this step, we need to give the algorithm the classification model that we created as well as the document labels. The algorithm will divide the documents into train and test and we can then figure out whether our identified features were successful or not (cross validation). If the features could not categorize the documents accurately, we then need to iteratively define the new features and repeat the process. In the feature definition, we can use Image Processing literature as needed.

Future Directions

There are some parts that need to be developed further from the User Interface website. This section reviews all the bugs and parts that need to be fixed in the next version.

- The drag and drop feature for each file needs to be added into the list of categories page. This feature is already developed and tested outside of Django and only need to be part of the website.
- The categories page only represents files within each category (filter them) and not all the items from the documents data for each category.
- The thumbnail creator python file needs to be integrated into the Django project so that the users can run this program using the thumbnail creator page.
- If the user wants to change the database, either by adding a new category or by uploading files into the database from the thumbnail creator page, the system needs to verify its access to the database.
- The new category implementation need to be developed. There is only the UI that exists.
- The user needs to click on the thumbnails and view the entire data. The url for this feature is designed and coded into urls.py

In terms of the logic of Archi-V project, an important future direction is to design an algorithm that classifies the files automatically according to their form representations. The archivists then can change the file categories based on their observations of the thumbnail views as well as their domain knowledge and experiences. This feature will guide the archivists in their organization tasks. Note that the files that the domain users moved and changed their categories will be excluded from the automatic classifications.

Using a version control system (Git) and a collaborative environment to work on the code (GitHub)

Using Git as a version control system

The following steps describes the steps in which the project was first uploaded in GitHub under a free account.

1. Installed Git in my local Mac computer. Typing $ git version tells us if the command line tools are installed. If git is not added to the path, I would do the following steps in order to set the path:
   - on Terminal, typed $ vim .bash login
   - Find the Git path using locate git on Terminal to figure out where it is installed.
   - Add the Git path to PATH=/somepath/somepath:/somepath/git/bin
   - log-out and log-in to Terminal
   - To test, type $ git status or $ git version
2. Next, I created a folder called ArchivIT on my local computer, and put all the project files and folders that I want to transfer to the repository.
3. On Terminal, type $ cd ArchivIT to go to that folder.
4. On Terminal, type $ git init ArchivIT. As a result of this command, a local repository gets created. There is no server running. All it does it to use file system, and creates a top level folder called ArchivIT, and a nested folder inside it called .git. If you want to delete that repository just delete that .git folder.
5. Type $ git add . to tell Git that you are now tracking all the files in this folder.
6. $ git status now should show all the green statements.
7. $ git commit -m my fist commit , uploaded all the added files to the local repository.
8. $ git log shows the first commit.

Uploading the Project on GitHub

The following steps describes the steps in which the project was first uploaded in GitHub under a free account.

1. Set up a free GitHub account at https://github.com.
2. On GitHub website, I created a New Repository called ArchivIT. I also initialized Read-Me.
3. In order to make a secure connection, I created an SSH key that I combined with HTTPS.
   - I first changed the settings on my Mac to show hidden files and directories in the following steps:
     - on Terminal, typed defaults write com.apple.Finder AppleShowAllFiles YES
     - Pressed Enter
     - Relaunched Finder
   - Found .ssh folder and backed up the files inside it because the steps after might override some of the files.
   - On Terminal, typed: $ ssh-keygen -t rsa -C my email address . This will make an RSA key.

- Pressed Enter for the next two steps.
- set up a passport.
- In .ssh on my Mac, I had a private key and a public key. In order to share the public key with GitHub. I copied the content of public key
- On GitHub, under account settings, SSH key, pasted the public key.
- To ensure that SSH key is working, in Terminal type $ ssh T gitgithub.com. Typed yes and entered the passport.

4. To created the connected to GitHub, and push what we you have done locally on our computer (pushed the codes) do the following steps:
   - $ git remote add origin git@github.com:..../ArchivIT.git
   - This tells Git that I want to add a remote repository. origin is a common name for that remote repository.
   - on Github.com, on the repository page, SSH tab, copied the repository address: git@github.com:...
   - At this point my local Git know where origin is.
   - on Terminal $ git push -u origin master
   - It will ask for SSH password that I created before. and upload the files.
   - In some cases that we have some files in GitHub that do not exist in the local folder. In that case, we have to merge the two first using $ git pull
5. Every time that we changed something, we need to say $ git add . $git commit -m more changes $ git push origin master
6. Another useful command is changing the code colours. $ git config global color.ui true
7. To remove the remote repository $ git remote rm origin

Participating in ArchivIT project on GitHub

If you receive an email with the GitHub address of the project, how could you join the project and participate in coding: Note that if you have readonly access, you still can begin coding on this open-source project and later, ask for code review and permission from the owner to publish it.

1. Click Fork button. That allows you to make a copy of the project to your personal account.
2. On Github, copy the html address of that copied repository.
3. On terminal, cd to your desired place.
4. On terminal, type $ git clone https://github.com/.... This copies the project on your local computer.
5. On terminal, cd to the project folder (ArchivIT)
6. Asked about this project status by typing $ git status
7. Now you can also ask about all the branches of this project by typing $ git branch -a
8. Start a new branch by typing $ git branch a descriptive name of the branch
9. Type $ git checkout branch name. Now you are switched to this branch. Although there is no differences between this and the master branch yet.

10. Make a change or add a new file to the folder. You could type the following list of commands to
    - git add ...
    - git commit -m a new branch file
    - git push -u origin name of the branch without the coatation
11. Note that if on terminal you type $ git checkout master, you would see the latest updates of the master branch not your branch. And all the files you created and changed, will disappear from the Finder window!!
12. When you want to make the changes on your new branch to appear on the master branch do the following:
    - Click on the Pull Request on Github. In Write tap, write a description to what you are offering to the core project maintainer. Send Request, and wait for approval.
    - The owner of the project can see this pull request, and click Merge pull request. There is also another box on Github, the owner can tell you why they are rejecting this pull request, and give you some comments.

References

[1] Django project.

[2] Ica-atom.

[3] jquery.

[4] Victoria L. Lemieux. Envisioning a sustainable future for archives: A role for visual analytics? *International Council on Archives Congress*, 2012.

[5] Victoria L. Lemieux. How archivists think: Exploring the archival reasoning process using cognitive task analysis and verbal protocols. 2013.