Capstone Project 2: Project Proposal

Deep Learning and Heathcare

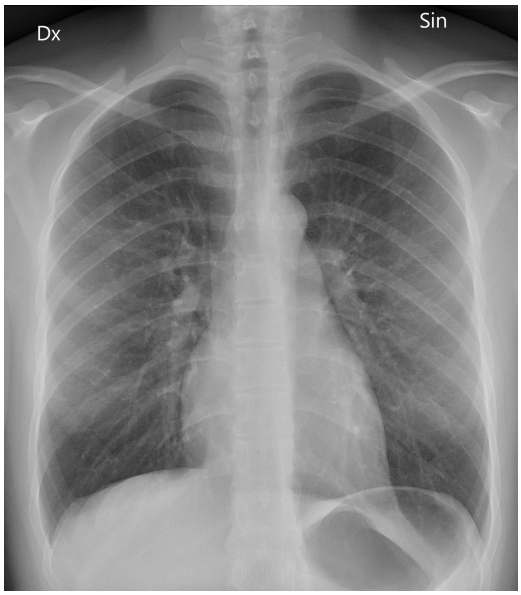


Image By Mikael Häggström

BACKGROUND

For the past two years, I have been a CrossFit coach. I've spent thousands of hours helping others create healthier and happier lives. I don't say that lightly. It has allowed me to see people go from having difficulties doing simple activities like getting around the house to being able to play with their kids (or grandkids), which has been an incredible experience!

While my last project was undoubtedly fascinating, and in a sense helped me rediscover the passion I had for the game of basketball, the impact it has outside of the NBA is next to nothing. For my next project, I wanted to get back to my roots and revisit my original motivation for pursuing a career in data science which was to figure out how to help people live healthier and happier lives.

Recently, machine learning researcher's at Stanford developed CheXNeXt, a deep learning algorithm that can concurrently detect a range of diseases in chest radiographs. Its performance in detecting specific pathologies was comparable to a group of practicing radiologists. Perhaps more impressively, it was able to do this in significantly less time (~two minutes vs. ~4 hours for radiologists).

Why is this so important? To begin, two-thirds of the world's population has no access to 'basic' X-ray or ultrasound examinations according to the World Health Organization. Even most developed countries around the world are experiencing a shortage of qualified radiologists, with up to 97% of radiology departments unable to meet diagnostic reporting requirements. With an increased workload brings the increased chance of error due to stress or fatigue.

In summary: there are not enough doctors to meet the ever-increasing demand for medical image analysis.

However, the primary issue when it comes to deep learning and medical image analysis has been that it requires a lot of data, which hadn't been accessible to train a reliable model. Then in 2017, the NIH Clinical Center released its ChestX-ray14 data set, which contained over 112,000 frontal-view chest radiographs from nearly 31,000 unique patients.

Not to be outdone, Stanford released CheXpert, a public dataset consisting of 224,315 chest radiographs from 65,240 patients with MIT co-releasing its MIMIC-CXR with nearly 372,000 chest x-rays! Additionally, the Radiological Society of North America released a data set on Kaggle for a competition to see who could build the best algorithm to detect potential pneumonia cases. So with more and more data becoming available, the opportunity to create a reliable deep learning model that can accurately identify pathologies is there, as evidenced by CheXNeXt.

In my opinion, this is an enormous opportunity to not only become familiar with a growing technology but to also become a part of the current revolution in healthcare. Additionally, it presents a chance to become familiar with working on a cloud computing platform as Google has the NIH data set available through its Cloud Healthcare API.

SUMMARY

What is the problem you want to solve?

- Create a model that can detect pathologies from medical images at the same level as practicing radiologists.

Who is your client and why do they care about this problem? In other words, what will your client do or decide based on your analysis that they wouldn't have done otherwise?

- My clients would be healthcare systems, and more specifically radiologists, who need to review an ever-growing number of medical images. A model that shows evidence it can perform at a high-level could be used as a tool to assist in examining images, allowing for a more efficient workflow. This has a direct impact on patients as well, who are able to receive a diagnosis (and potentially treatment) much sooner.

What data are you using? How will you acquire the data?

- There are a few data sets available, all of which are publicly available.
    - CheXpert
    - MIMIC-CXR
    - RSNA's Kaggle data set
    - NIH's ChestX-ray14
- I'm currently in the process of examing the data and have not determined which one would be the most appropriate for this project
- Want to quickly note as well that deep learning will be utilized for this project and NIH's data is easily available via Google Cloud's Cloud Healthcare API.

Briefly outline how you'll solve this problem. Your approach may change later, but this is a good first step to get you thinking about a method and solution.

- I'll begin this project by looking further into convolutional neural networks, which was the basis for CheXNext to see if I can replicate their performance. While I hope to develop something that can at least attempt to predict all the pathologies laid out by the team at Stanford, the complexity may be too much for me at this point. To keep things simple, the algorithm that I develop may only predict for a specific disease, like pneumonia, for example. However, only time and further exploration will tell.

What are your deliverables?

- In the end, I plan to deliver my code (via Jupyter Notebooks), a comprehensive paper that details the entire process and a slide deck to present my findings. More importantly, I hope to take a step towards helping others live healthier lives.

Head in the clouds?

Data Acquisition and Cloud Computing Set-up for Capstone #2

For my second capstone project, the primary objective for my second capstone project is to create a deep learning model that can detect a pathology[1] from medical images at approximately the same level that Stanford's Machine Learning group achieved.

Achieving an AUC of 0.907 is not an easy feat, but I'm very excited about this project, primarily because it'll allow me to begin to experiment with both cloud computing and deep learning, two of the 'hottest' subjects in technology right now. Also, the two mostly go hand-in-hand as deep learning requires a significant amount of computing power and hardware via GPU's[2] that most home computers like my iMac don't even compare. Even though I was slightly intimidated at first to jump into what I perceived was the deep end of the technological swimming pool, I closed my eyes and took that leap of faith.

Before I go further in regards to setting up my data and compute instance in the cloud, I first want to discuss why I chose CheXpert over the other data sets I mentioned previously in my project proposal.

Firstly, gaining access to the data was relatively simple. All one has to do is go to this webpage, scroll to the bottom, fill out the necessary information and you'll be emailed a link to download the zip file that contains all the images from the study. In contrast, acquiring the MIMIC-CXR data set from MIT was a multi-step process that involved passing multiple ethics quizzes and an approval process that could take weeks.

The ChestXray14 dataset was also relatively easy to access[3], but upon further investigation, it looks like this particular data set doesn't seem quite up to snub. Luke Oakden-Rayner, a Ph.D. candidate and Radiologist, pointed out multiple problems with CXR14, one of them being a problem with the labeler that extracted the pathologies for each image from the associated medical report.

Upon further visual inspection of the images, Oakden-Rayner also found many inaccurate labels from randomly chosen sets of images for the pathologies indicating the potential of a high error rate in the labeling process[4].

---

[1] Or potentially pathologies; with an additional number of potential classes the model building process does get more complicated and I am working with limited resources

[2] Or TPUs

[3] It can be download via Kaggle or accessed via GCP's Cloud Healthcare API

[4] See the section titled 'Part 1: Visual label accuracy in ChestXray14' in Oakden-Rayner's post for in-depth analysis

Instantly, my 'garbage in, garbage out' senses were tingling.

Upon some further investigation, I came upon a few other complaints[5] about CSXR14's quality, which ultimately led me to decide not to use it for my project despite it being readily available.

However, I want to stress that CheXpert is not perfect. While a significant improvement on CXR14's labeler, CheXpert labels were also extracted via NLP[6], which has an irreducible error due to inaccurate/nondescriptive medical reports. Additionally, there are several repeat scans which reduce the effective size of the data set along with issues related to the accurate visual interpretation of the images due to downsampling[7].

However, Oakden-Rayner comments that this dataset is a significant improvement upon CXR14 and that CheXpert is quite possibly the best medical imaging data set that is publically available. In summary, this data set may not be quite ready for training full-fledged AI models -- after all, we are talking about life and death -- but it does present an excellent opportunity for continuing research in utilizing deep learning for medical image analysis.

Now, with data in hand, the next question is: what cloud service to use? Amazon Web Services is by and large the leader when it comes to cloud computing, with other options like Microsoft Azure, GCP[8], IBM, and Oracle taking up the rest of the market.

For this project, I've decided to utilize Google Cloud Platform mainly due to the $300 credit that Google gives new users to try out their platform. Another influence was Google's focus on AI as a company and perhaps more so than the other providers, has optimized its cloud platform with this approach in mind. I'd also been utilizing the GCP platform for fast.ai, a MOOC created by Jeremy Howard in hopes of democratizing deep learning. The combination of ease-of-use, previous hands-on experience, and the $300 credit made picking GCP a no-brainer.

With data downloaded and cloud account setup, the next step was uploading the data set (i.e., CheXpert) to 'the cloud.' As a beginner, I found this first task to be quite daunting; however, after reading plenty of documentation combined with a little playing around, the process was relatively straight forward.

The first step is to create a bucket, which are the primary containers that hold your data in GCP. A user can create a bucket via the GCP console, command line using gsutil, the REST API, or even with Python code!

Additionally, I would like to point out that I utilized a multi-regional storage class. What does this mean and what even is a storage class? It is a property of your bucket that applies to how objects (i.e., data) added to the said bucket are stored.

For this project, I utilized multi-regional storage, which is best for data that is frequently accessed, otherwise known as "hot" objects[9]. The benefit of utilizing multi-regional storage is that it provides the highest data availability, which also means that it comes at the highest cost. The storage cost was moot though; the cost was only $0.026[10] versus $0.020 for the regional storage[11]. Since my data was only ~11GB, storage cost was

---

[5] See here and here for a few examples
[6] Natural language processing
[7] Images are 8-bit png images with 256 grey levels
[8] Google Cloud Platform
[9] More information on storage classes can be found here
[10] Per GB per month

going to be minimal no matter which option I selected. However, if I were to have stored hundreds of GBs or even multiple TBs of data, then I would have had to take storage classes into further consideration.

After my bucket was created -- with the original name of 'joes-capstone2-data' -- I could then upload my CheXpert data as a zip file. By drag and dropping the file from my local desktop process and the upload started. After approximately 1-2 hours - due to a somewhat limited personal internet bandwidth - my data was ready to be accessed.

At this point, we have our data in the 'cloud' but you may be wondering: how do we, you know, access it and begin to analyze it via a tool like Jupyter Notebooks?

Answer: Google's Compute Engine!
Compute Engine allows us to create high-performance virtual machines, which you can think of, in a sense, as your personal computer in the cloud. The benefit of this setup is that it allows you to personalize the hardware you're using, which, in our case, allows us to attach GPU's to our virtual machine. GPUs, or graphics processing units, are beneficial when it comes to deep learning as they are more efficient, in both processing and cost, than regular CPUs[12].

The setup was more complicated than the previous steps and did require more patience. There are two primary routes you can take when creating a VM instance: command line or the GCP console. The command line route requires the user to install Google's CLI (command line interface) and configure the Google Cloud SDK on your local machine. After this, using some command line magic, you can create an instance.

The other option is via the GCP console and setup is similar to when we created the bucket, albeit with different options we need to select. After you click on 'Create Instance,' the user interface takes you to a screen that gives you options such as Region, Zone, and Boot disk, amongst others. The most important feature though is the Machine type, which is where we pick the hardware of our virtual computer.

In the end, this could be a frustrating part of the project, especially for individuals that are not too familiar with cloud computing (of which I belong to!). Luckily though fast.ai made this process more straightforward and was instrumental in helping me get my compute instance up and running.

Once you have the instance setup, it's (almost) as easy as hitting the play button! After navigating to the Compute Engine/VM instances interface in GCP, the user sees a list of their available VMs. Check the box next to the one you wish to use and hit the play button on the top of the page. Your instance is ready once there is a green check mark next to it, after which you enter the following command:

```
gcloud compute ssh --zone=$ZONE jupyter@$INSTANCE_NAME -- -L 8080:localhost:8080
```

After this, a Jupyter Lab/ Notebook is at your disposable! You can then access your bucket, download and unzip the CheXpert data and begin analysis. This entire process can be seen in more detail in the following Jupyter Notebook.

[11] Data is stored in a narrow geographic region
[12] More information can be found here

‘Chex’ this out!

Exploratory Data Analysis on CheXpert Data Set

For my second capstone, I’ve utilized the CheXpert data set via Stanford’s Machine Learning Group. This particular data set represents one of the most significant steps in recent years to advance research in regards to deep learning within the healthcare space. Before we jump into creating our model, let’s take a more in-depth look at the patients themselves. After downloading CheXpert, you see that it includes a train.csv and valid.csv file (see below).

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 223414 entries, 0 to 223413
Data columns (total 19 columns):
Path                            223414 non-null object
Sex                             223414 non-null object
Age                             223414 non-null int64
Frontal/Lateral                 223414 non-null object
AP/PA                           191027 non-null object
No Finding                      22381 non-null float64
Enlarged Cardiomediastinum      44839 non-null float64
Cardiomegaly                    46203 non-null float64
Lung Opacity                    117778 non-null float64
Lung Lesion                     11944 non-null float64
Edema                           85956 non-null float64
Consolidation                   70622 non-null float64
Pneumonia                       27608 non-null float64
Atelectasis                     68443 non-null float64
Pneumothorax                    78934 non-null float64
Pleural Effusion                133211 non-null float64
Pleural Other                   6492 non-null float64
Fracture                        12194 non-null float64
Support Devices                 123217 non-null float64
dtypes: float64(14), int64(1), object(4)
memory usage: 32.4+ MB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 19 columns):
Path                            234 non-null object
Sex                             234 non-null object
Age                             234 non-null int64
Frontal/Lateral                 234 non-null object
AP/PA                           202 non-null object
No Finding                      234 non-null float64
Enlarged Cardiomediastinum      234 non-null float64
Cardiomegaly                    234 non-null float64
Lung Opacity                    234 non-null float64
Lung Lesion                     234 non-null float64
Edema                           234 non-null float64
Consolidation                   234 non-null float64
Pneumonia                       234 non-null float64
Atelectasis                     234 non-null float64
Pneumothorax                    234 non-null float64
Pleural Effusion                234 non-null float64
Pleural Other                   234 non-null float64
Fracture                        234 non-null float64
Support Devices                 234 non-null float64
dtypes: float64(14), int64(1), object(4)
memory usage: 34.8+ KB
None
```

Above are info() outputs from EDA notebook (see here)

Above you can see the high-level information from the provided training and validation CSV files. The training set, which we'll use to train our deep learning model, has nearly 223,414 entries with 19 columns. Most of these columns are devoted to the specific pathologies like enlarged cardiomediastinum and atelectasis to support devices (i.e., this is technically not a pathology but was included by the team at Stanford based on its prevalence within the images).

The `Path` column represents the 'path' to the image, which will come in handy once we get to the model training portion of the capstone. The information in this column assists the learner to connect an observation from the CSV file to the associated image file within our data folder.

There are two somewhat distinct columns - sex and age - that represents whether the patient was male or female and the patient's age, respectively.

The next two columns indicate the positioning of the medical image. In this specific data set, x-rays were done from either the front (i.e., frontal) or the side (i.e., lateral), which is indicated by the Frontal/Lateral column. Lastly, the AP/PA column shows whether or not the patient was standing or lying down when the imaging was done (which we'll dive further into a little later).

Another critical thing to make a note of is that the training set had quite a few NaNs, meaning that there was missing information. However, upon further investigation of Stanford's Machine Learning GitHub, we can clarify that missing values mean that no mention of that particular pathology was extracted via the labeler. Stanford

treated those cases as negative (i.e. `0`) when they created their model so we did the same and relabelled them as `0` for these pathological columns.

What does AP, PA mean?

Within the data set there was one column -- AP/PA -- where ~14% of the observations were missing values, so we decided to dig deeper. While 14% may not sound like a lot when you consider there are approximately 223k observations that 14% adds up to a little over 32,000 missing values. For most data sets, there are specific techniques you can use to address missing values - replace with the mean or median, use linear regression to predict the value, etc. - but with this data set, we're somewhat stuck because we're dealing with images and can't merely input a number.

As a reminder AP/PA has to do with the positioning of the medical image so if we wanted to maintain the integrity of our data (at least within the CSV file), we'd have to go through each image and manually label them one by one, preferably with the assistance of a domain expert (i.e., board-certified radiologist(s)). Now, what exactly do PA and AP mean?

PA stands for posterior-anterior, which is obtained when the patient is in a standing position, facing the cassette and the x-ray tube is approximately 72 inches away.

AP stands for anterior-posterior, which is when the patient is lying down, and the x-ray tube is only 40 inches from the patient.[13]

In summary, it indicates whether or not a patient was sitting or standing when the image(s) were taken. In a traditional setting (i.e., a hospital or healthcare facility), radiologists have access to top of the line medical imaging equipment, which allows them to analyze images at a very high pixel rate. Unfortunately, we do not have access to these high-resolution images because that would require a significant amount of storage. The small downsampled version is 11 gigabytes, compared to 439 gigabytes for the 'original' version of the images, which are not downscaled but reduced to 8-bit png images. What does this mean? These 8-bit images have 256 grey levels; for comparison, clinical x-rays are stored at 16-bit or 65,536 grey levels!

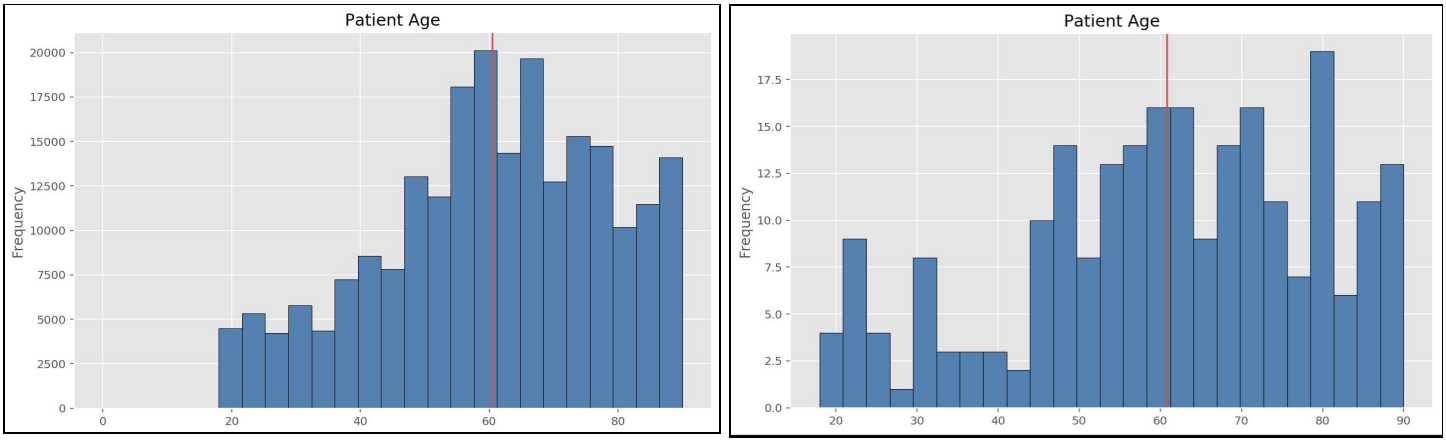
Additionally, a tiny number of observations were labeled as either ‘LL’ or ‘RL’. What did these mean? From what I could gather from my investigation is that these particular observations were done laterally in that the central ray passes from one side of the body to the other through the axial plane.[14]

However, I could not find the following patients in the below data frame upon further inspection. Within the downloaded ‘train’ folder, there were no patients with the corresponding patient ID numbers. It looks as if these images were not included which is odd, but due to the minimal number of observations (~17) that were labeled either ‘LL’ or ‘RL’, I'm not too worried about it. Even if there were some information, its effect on the model would be negligible at best.

---

[13] Source
[14] Source

Observations: Patient Age



Histograms showing age distribution of patients within training set (left) and validation set (right)

The first aspect that sticks out about the histograms above is that around age 50, there is a significant uptick in the number of cases as evidenced by the higher bar heights. The bins climb steadily until the age 60 bin (which contains a range of ages around 60), which serves as the peak in the training set (top histogram) and represents the second highest peak in the validation set (bottom histogram).

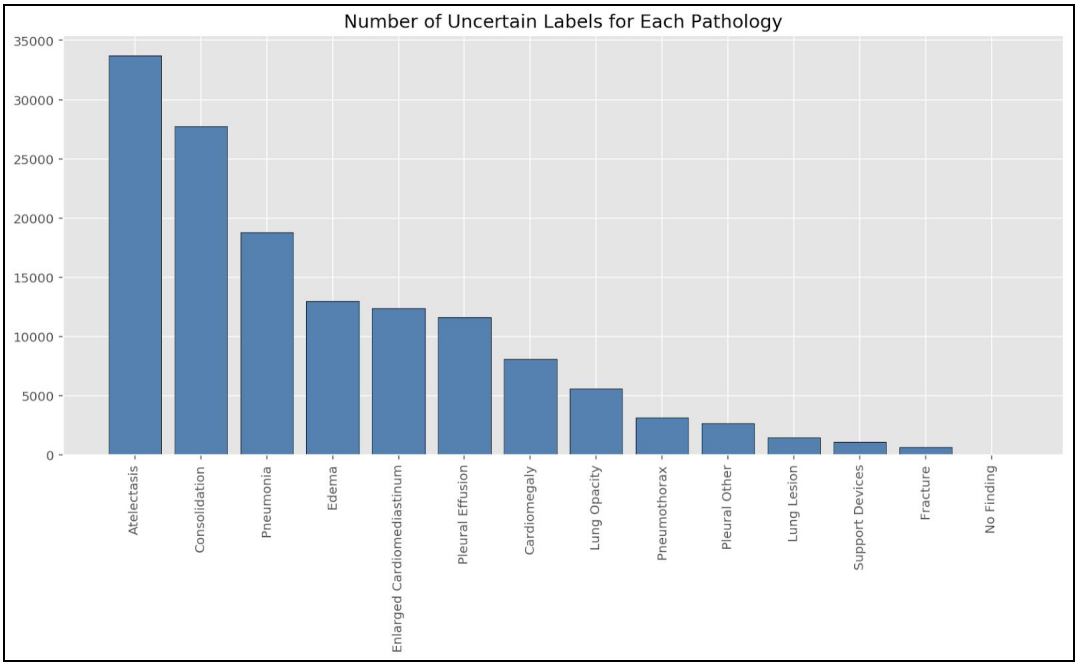
The red line that vertically cuts through both the histograms represents the mean age of that particular cohort. They both look to be around the same at slightly over 60 years of age. When we take a look at the training sets (i.e., top) histogram, we can see that the values on the x-axis start at 0. Why was this? An initial hypothesis was that there were observations that may have an age of '0.' It turns out this wasn't entirely the case, but there were three observations below the age of 18, which was skewing the histogram.

When we compare the two histograms, we can see some slight differences in their distribution. While both appear to peak at approximately the same point (around age 60) the distribution to either side is different. For ages greater than 60, the training set has a downward trend with a spike in the number of individuals whose ages are in the late-80s. For the validation set's histogram, there are spikes around every decade, i.e., age 70, 80, and 90.

Now let's take a look at the lower end, the observations whose age is less than 60. From the ages of 20 to 40, the number of observations in the training set remains pretty consistent with a slight uptick from approximately ages 40 to 50. This is slightly different however from the validation set which again shows an uptick around the decade marks of age 20 and 30, then a sudden spike around the late-40s.

There is one thing that we have to keep in mind though when comparing the training and validation set: the training set is ~223,000 observations while the validation set is only 234, which is approximately ~0.1% of the training data. Overall, the two histograms are not the same but are relatively close in shape, which is pretty spectacular, considering the gap in the number of observations between the two data sets.

Observations: Uncertainty Amongst Pathologies



Graph indicating the total number of uncertain labels for each pathology

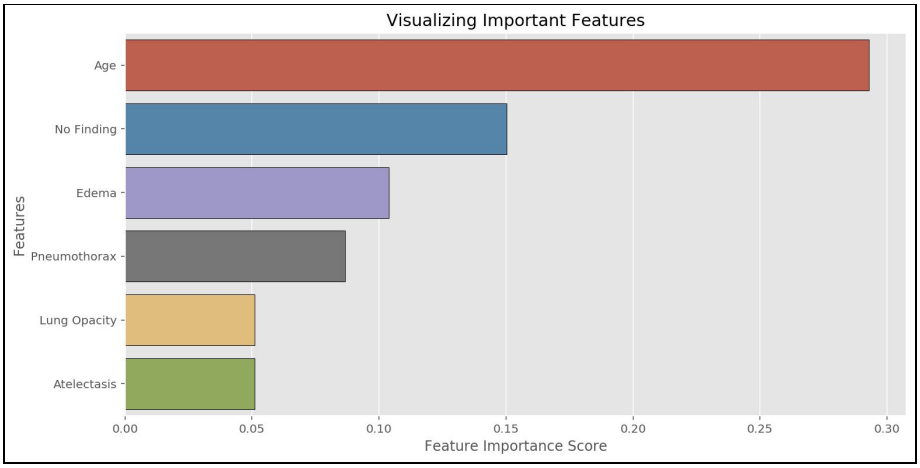
The above graph shows in descending order the pathologies with the most uncertain labels. Now is a good chance to mention how the team at Stanford went about labeling the chest x-rays. In the accompanying radiology reports, they developed and applied an automated, rule-based labeler to extract the labels. Observations mentioned at least once were positively classified (i.e., 1) while the presence of at least one negatively classified mention resulted in a negative classification (i.e., 0). In the case that a pathology was mentioned but could not be confirmed as either positive or negative, it was assigned an uncertain label (i.e., -1).

As we can see from the graph above, atelectasis, consolidation, and pneumonia were the top three pathologies in terms of uncertain labels. Because they are technically neither positive nor negative, this begs the question: what do we do with these uncertain labels?

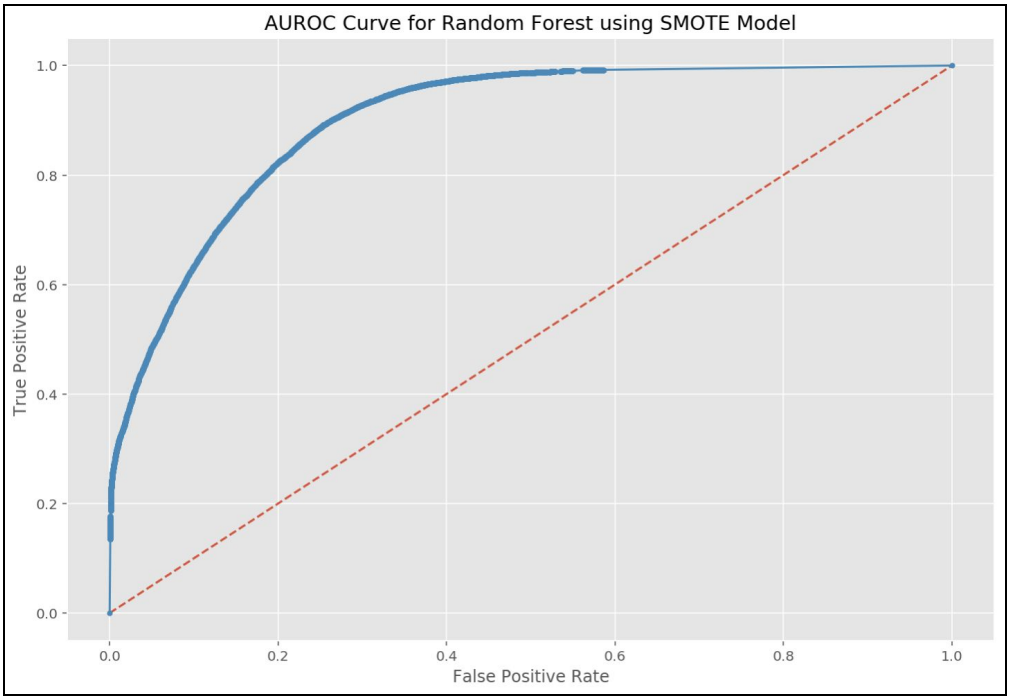
There were a few different strategies that the Stanford team suggested:
Ignoring: simply drop the uncertain labels during training
Binary Mapping: map all uncertain instances to either negative (U-Zeroes) or positive (U-Positive)
Self-training: build a baseline model with the ignoring approach, then relabel uncertain labels according to predictions made by the model
3-Class Classification: treats uncertainty as a class, turns original binary classification problem into a multi-class classification problem

Motivated by the self-training technique, we asked the question: what if we could generate a machine learning model from the information provided so far (i.e., from just the CSV file)?

Can we use ML to label uncertain pathologies?



Feature importance scores from Random Forest using SMOTE model



AUROC curve of Random Forest model that utilized SMOTE due to highly unbalanced label distribution of cardiomegaly

For this portion, we decided to use the random forest classifier due primarily to is versatility in handling different types of features, it requires minimal input preparation and is routinely one of the best performing machine learning algorithms. However, we remembered the adage ‘garbage in, garbage out’ and decided to clean up certain parts of the data to hopefully help generate a more accurate model. Below are a few of the modifications we made to the data set:

- Removed all observations that contained any uncertain pathological labels
- Converted all the pathology columns to categorical types
- Dropped Path, Frontal/Lateral, AP/PA, uncertain_features columns
- Converted categorical variable within Sex column into dummy/indicator variables

After this, we created the variable X, which contained all the features of the data set, and the variable y, which included our target variable -- the cardiomegaly column.

We created three models: a baseline random forest, a random forest utilizing an oversampled data set, and another random forest that utilized an oversampled data set using SMOTE. The reason for the oversampling was due to the relatively poor performance of the baseline model. The baseline random forest had a mean ROC-AUC of 0.743 on the test set but a mean score of 0.363 and 0.109, respectively, for the precision and recall metrics. Due to the highly uneven nature of the cardiomegaly labels -- approximately 88% of the observations were negative -- we decided to utilize oversampling, which is a technique that adds more examples from the minority class (i.e., the positive label). The hypothesis was that it would help the algorithm better detect that patterns that led to an observation being labeled as either positive or negative.

Using the sample function, which randomly sampled with replacement from our current data, we were able to generate a data set that had the same number of positive cases of cardiomegaly as negative by using the observations already available to us. Utilizing this strategy, we were able to significantly improve on the baseline, with a ROC-AUC score of 0.848, 0.727 for precision, and 0.88 for recall.

For the last model, we utilized SMOTE, which stands for synthetic minority oversampling technique. In contrast to the sample function, SMOTE takes it a step further by synthesizing points for the minority class from those that already exist. It does this by randomly picking a point from the minority class and then computing the k-nearest neighbors for that point. After that, it creates synthetic points between the chosen point and its neighbors. Essentially, we are creating ‘new’ patients!

With this technique, we were able to achieve the best performing model: a random forest that had a mean ROC-AUC of 0.896, mean precision of 0.776, and a mean recall of 0.886. Additionally, the hyperparameters of the random forest were not further tuned, so there is potential for achieving even better performance.

While this is showing promise, I have yet to determine if I want to utilize it to re-label the uncertain labels. I believe the best course would be to start with one of the uncertainty methods that the Stanford team utilized (i.e., ignoring, binary mapping, etc.) and depending on the results, then experiment with random forest mentioned above (after hyperparameter tuning).