Predicting H1B Labor Condition Application Petition Success: Capstone 1 Milestone Report

# 1. Problem and Client

The problem I have chosen for my first capstone project is attempting to predict the outcome of the Labor Condition Application (LCA), a pre-requisite for H1B visa applications. The LCA is the final step before applicants may submit their H1B visa application to the US Department of Labor. Once the LCA is approved and the visa application is submitted, the next steps involve a lottery and therefore the decisions are essentially taken out of the hands of the applicants. For this reason, it is essential that applicants have their LCA approved as it is the last time that their choices are taken into consideration.

There are two prospective clients for this work. One would be a foreign citizen who would like to obtain employment in the United States. This person might want to know which types of jobs, what employment levels, and in what parts of the country they should be looking for employment to best maximize the chances that their H1B visa is ultimately approved. The other prospective client is an American company who may be looking to hire a specific foreign citizen. They may want to ensure that the combination of position, salary, employment status, etc. that they are offering their prospective employee is likely to succeed the application process.

The main dataset that I will use for this project consists of LCA petitions filed with the US Department of Labor between 2011 and 2016. I have obtained these data from the Kaggle H-1B Visa Petitions 2011-2016 dataset (<https://www.kaggle.com/nsharan/h-1b-visa>). This represents roughly 3 million LCA petitions. It should be noted that the data made available through Kaggle do not include all fields that are required on the LCA petition. However, they do include the final case status, employer name, standard occupational classification (SOC) system name, job title, whether the position will be full time or part time, the prevailing wage for that position, the city and state of the position, as well as the latitude and longitude of the employer. Other datasets include census block information and county population data available from the FCC Area API and the Census Population Estimates API.

# 2. Data Wrangling and Cleaning

The dataset provided by Kaggle was already relatively clean. With that said, it was necessary to do some cleaning and rearranging of the data to suit the needs of my analysis. After downloading the data and reconciling field names, my first step was to inspect the field “CASE\_STATUS.” This is the field that contains the outcome of the LCA process and is the outcome that I will eventually attempt to predict. An online search revealed that there are only four valid values for this field: certified, certified-withdrawn, withdrawn, and denied. For the purposes of this analysis, I decided only to keep certified, certified-withdrawn, and denied.

While it is helpful to know the outcome when an application was certified but then withdrawn, it is not helpful for prediction if all I know about the application is that it was withdrawn. Therefore, I believe that it is appropriate to drop records where CASE\_STATUS is “withdrawn.” I also wanted to condense these results so that I had a binary success-failure field. I created a new field (“CERTIFIED”) where all records with CASE\_STATUS values of “certified” or “certified-withdrawn” have the value “certified” and records with the CASE\_STATUS value of “denied” have the value “denied.”

Other cleanup included changing the datatype of YEAR from a float to an integer. I split the field WORKSITE into two separate fields so that I now have one field with cities (“CITY”) and one field with sates (“STATE”). Finally, I created a new column (LOG\_WAGE). This is populated with log10 value of the PREVAILING\_WAGE field. PREVAILING\_WAGE has values that span from 0 to over 10^9 so I determined that it would be easier to visualize this spread of the data on a log-scale.

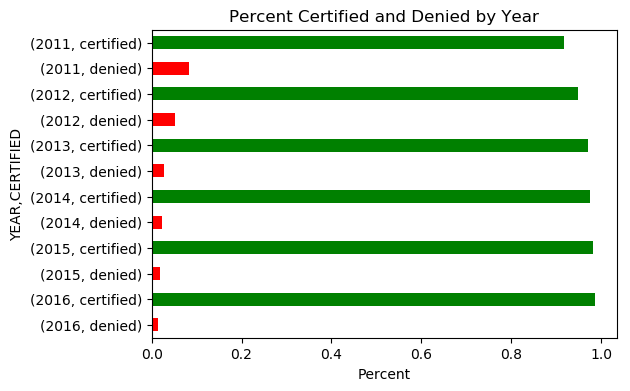
In addition to the cleanup of the data obtained from Kaggle, I generated two other datasets via calls to the FCC Area API and the Census Population Estimates API. First, I obtained a unique list of latitude and longitude pairs from the cleaned H1B dataset. I then used these to call the FCC Area API. This resulted in a dataset that contains latitude, longitude, census block Federal Information Processing Standard (FIPS) code, population of the census block, and other county and state identifiers including the county FIPS code. I then used this FCC census block dataset and, with the unique list of county-FIPS codes, called the Census Population Estimates API. The resulting dataset included the population of each county in my dataset. Finally, I merged these two datasets so that I had one overall csv containing location information including FIPS codes for counties and states, as well as county population, and census block population.

All code that I used to clean the Kaggle dataset as well as make the calls to the FCC and Census APIs can be found on git hub here: <https://github.com/Liptoni/Springboard/tree/master/H1B_Capstone/python>.

# 3. Initial Data Exploration

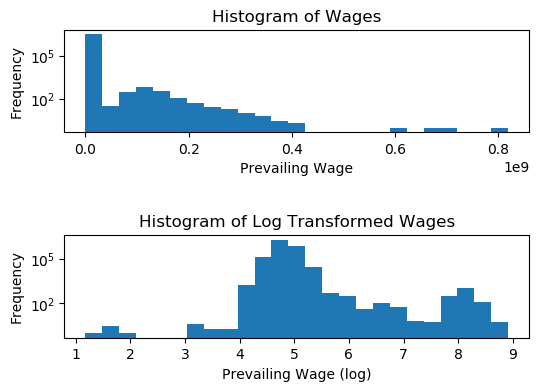
My initial exploration of the data revealed a number interesting items. The total number of certified applications is 2,818,282 while there were only 94,346 denied applications. This large discrepancy in the number of certified vs. denied applications leads me to believe that there may be some outside factor influencing which applications are submitted to the LCA process. It is possible that the service that I am proposing already exists in some form. When looking at the percent of applications that were certified by year, there was an increasing trend of certified applications from 2011 to 2016. In 2011, 91.6% of applications were certified. That number grew to 98.5% in 2016 (Figure 1).

Figure 1. Certified applications by year



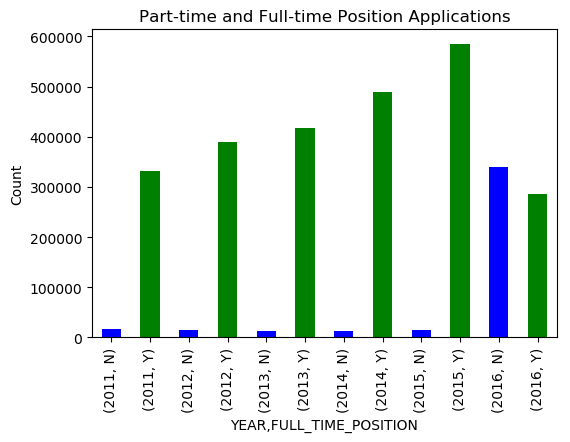
Next, I looked at the distribution of wages in general. There were two values over $1 billion dollars. While this may not be outside of the realm of possibility, the positions being applied for were physical therapist and software developer. I was forced to determine that these were erroneous records and removed them from future analysis. The mean wage, excluding wages that were reported as $0 was $141,144.20 while the median was $65,000. A histogram of prevailing wages revealed what the summary statistics indicate, which is wages that are heavily right skewed. This is not unsurprising as there would not be many incredibly-high paying jobs. Figure 2 shows the distribution of prevailing wage in general as well as on a log10 scale. The log transformation indicates that the data may be bi-modal with a cluster of very high wages.

Figure 2. Wage histograms.



Finally, I was interested in determining the number of full-time and part-time positions that were applied for each year. This turned out to be very interesting. In each year from 2011-2015, the number of full-time positions far exceeded the number of part-time positions. Then in 2016 the number of part-time positions being applied for suddenly exceeded the number of full-time positions (Figure 3). This extreme shift in the relationship between full-time and part-time positions being applied for indicates to me that there may have been some guidance issued by the Department of Labor indicating that certain positions would be more likely to be approved.

Figure 3



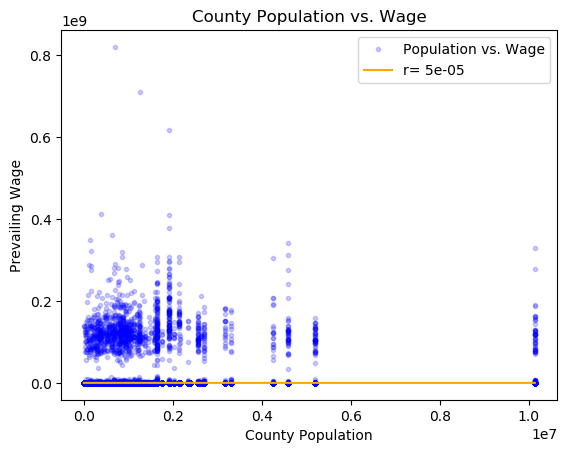
# 4. Preliminary Statistical Analysis

Following up this initial exploration I chose to do some preliminary statistical analysis of the dataset. The first test I ran was to determine whether the mean prevailing wage was the same for applications that were certified and denied. The results of the t-test provided a p-value of essentially zero. This led me to reject the null hypothesis that the means of the two sets of applications were the same. I ran a similar test to determine if the mean wages for full-time and part-time positions were equal and again obtains a p-value of essentially zero leading me to reject the hypothesis that the means of these two groups were equal.

I ran one more t-test. This time I was interested in testing whether the county populations for certified and denied applications were equal. The mean population for certified applications is 1,720,650 (n=2,721,739) and the mean for denied applications is 1,993,234 (n=85,107). The resulting p-value of the t-test was nearly zero leading me to reject the null hypothesis that the county populations for the two groups were equal.

Finally, I wanted to explore the relationship between county population and the prevailing wages. I plotted the two against each other and fit a least-squares line (Figure 4). The resulting Pearson correlation coefficient was 0.00005 indicating that there is essentially no correlation between county population and the wages of jobs being applied for in those counties.

Figure 4. County Population vs. Prevailing Wage.



All of the code I used for my data exploration and preliminary statistical analysis can be found in the jupyter notebooks here: <https://github.com/Liptoni/Springboard/tree/master/H1B_Capstone/jupyter_notebooks>.

# 5. Classification Analysis

For the classification analysis, I dropped all incomplete records. This left me with 2,790,057 records with witch to train and test my model. For the parameters that I eventually ended up choosing (see section 5.2) I feel as though it would not be too difficult to have all pieces of this information for any application being processed. Since this is a classification problem, and based on the size of the dataset, I decided to use the SGD Classifier from Scikit-learn to build a classification model.

## 5.1 Cross-Validation

The code that I used to perform the cross-validations can be found here: <https://github.com/Liptoni/Springboard/blob/master/H1B_Capstone/python/cross_validation.py>

### 5.1.1 Feature Selection

My first problem was choosing the features to use for the model. I wanted to start with the largest set of features and then work my way down to smaller models. The largest set of features I used consisted of FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, block\_fis, block\_pop\_2015, county\_fips, county\_pop, state\_code. I chose these features because there were no obvious correlations between them. Some of the other features such as city, and state would have been correlated with the longitude and latitude. The categorical features JOB\_TITLE and EMPLOYER\_NAME had 272,378 and 212,642 unique value respectively. I determined that there would not be enough unique information from these fields to consider them useful for the model.

### 5.1.2 Hyperparameter Tuning

There are many different hyperparameter options to tune for an SGD Classifier. I limited those that I wanted to tune to four: loss, penalty, alpha, and max\_iter. Loss is the loss function that the algorithm uses. Scikit-learn has five loss functions geared toward classification: hinge, log, modified huber, squared hinge, and perceptron. I chose to use test all five. The penalty hyperparameter is the regularization term used by the model. The default SGD Classifier uses L2 but there are also options to use L1 and elasticnet loss. I chose to test all three. Alpha is a constant that is multiplied by the regularization constant. The default is 0.0001 so in my first runs I started with a set ranging from 0.0001 to 1 by orders of 10. After running the cross-validation a few times, I was consistently seeing optimal alphas of 0.1 or 1. I decided to shift the range of alphas to test to 0.01 to 10 by orders of 10 to determine if larger values of alpha would be selected by the cross-validation. I settled on this range for further testing. Finally, max\_iter determines the number of passes over the training data. I chose four values to test: 5, 10, 100, 1000.

Testing and cross-validation is computationally expensive so I decided to do cross-validation using a random subset of 10,000 records. I used a random seed of 24 to ensure that I consistently used the same subset of 10,000 records for all cross-validation iterations. I split the data in to training and testing sets holding out 25% for testing the optimal hyperparameters identified during cross validation. I used Scikit-learn’s GridSearchCV function and five-fold cross-validation on the training set to determine the optimal combination of hyperparameters. I used the mean accuracy of the test data and labels to compare models with different sets of features.

### 5.1.3 Other Modeling Parameters

In addition to the hyperparameters of the SGD Classifier, I tested other options and methods to determine if I could refine my model even more. I attempted to use imputation, testing all three of mean, median, and mode imputation, fitting and not fitting an intercept for the model, and using kernel approximation (RBF Sampler, Nystroem). I tested each of these separately, using the same cross-validation steps described in section 5.1.2.

## 5.2 Testing Models on the Full Dataset

Interestingly, there were several models that produced the same score during cross-validation. I believe that this is because the model has a very hard time predicting “denied” applications but is fairly good at predicting “certified” applications. To determine the *best* model, I decided to use the full dataset to test all of the models produced by cross-validation that had the same high accuracy score. Since the full dataset is so large, I needed to use the partial\_fit method of the SGD Classifier to fit individual chunks of 10,000. As with the cross-validation step, I split the data into training and testing sets, holding 25% of the data for testing the fit of the model. I used the accuracy score as well as Scikit-learn’s classification report to determine the accuracy of the models. The classification report consists of a precision, recall, and f1-score for each label as well as a weighted score for each category for the whole model. Table 1 shows the features, hyper-parameters, and scores for each of the full models I tested. The code that I used to run the full model can be found here: <https://github.com/Liptoni/Springboard/blob/master/H1B_Capstone/python/analysis.py>

Table 1: Full Model Results

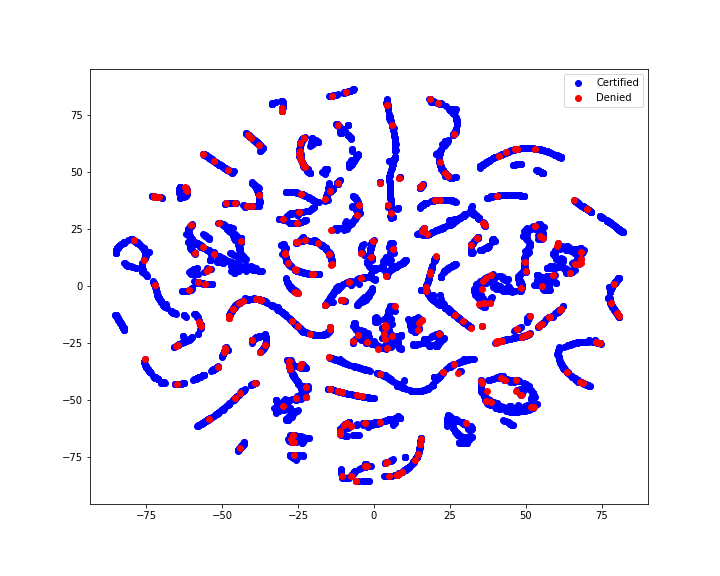
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Features** | **Alpha,**  **Loss,**  **Max\_Iter,**  **Penalty** | **Kernel** | **Fit Intercept** | **Accuracy** | **Precision (certified; denied; weighted)** | **Recall (certified; denied; weighted)** | **F1 Score(certified; denied; weighted)** |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, county\_fips, county\_pop, state\_code | 1  Hinge  5  L2 | NA | TRUE | 0.96967 | 0.97; 1.00; 0.97 | 1.00; 0.00; 0.97 | 0.98; 0.00; 0.95 |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, county\_fips, county\_pop, state\_code | 1  Perceptron  100  L1 | NA | TRUE | 0.96965 | 0.97; 0.00; 0.94 | 1.00; 0.00; 0.97 | 0.98; 0.00; 0.95 |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, county\_fips, county\_pop, state\_code | 0.01  Hinge  5  L2 | RBFSampler | TRUE | 0.96965 | 0.97; 0.00; 0.94 | 1.00; 0.00; 0.97 | 0.98; 0.00; 0.95 |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, county\_fips, county\_pop, state\_code | 0.01  Hinge  5  L2 | Nystroem | TRUE | 0.96965 | 0.97; 0.00; 0.94 | 1.00; 0.00; 0.97 | 0.98; 0.00; 0.95 |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat, county\_fips, county\_pop, state\_code | 1  Hinge  5  L2 | NA | FALSE | 0.63599 | 0.99; 0.06; 0.96 | 0.63; 0.69; 0.64 | 0.77; 0.10; 0.75 |
| FULL\_TIME\_POSITION, PREVAILING\_WAGE, SOC\_NAME, lon, lat | 0.1  Hinge  10  L1 | NA | TRUE | 0.96965 | 0.97; 0.00; 0.94 | 1.00; 0.00; 0.97 | 0.98; 0.00; 0.95 |

## 5.3 Interpreting Full Model Results

As you can see above, most of the models have very high accuracy. The only model with less than 96.96% accuracy was the model that did not fit an intercept. The fact that they all have a very high accuracy score should not be that surprising and when taken in context is not very impressive. Of the more than 2,790,000 applications 96.968% of all applications are certified. My best model only has an accuracy score of 96.967% which means that it performs slightly worse than if we randomly flipped a coin to determine whether an application would be certified or denied. My best model does a very good job at predicting certified applications, but it is not perfect. It is also very bad at predicting denied applications, even though it is the only model to predict a denied application. A precision score of 1.0 means that all the applications that were classified as denied should have been classified as such, but a recall sore of 0.00 means that there were so many false-negative applications that essentially 0% were classified correctly.

I wanted a way to visualize my data to see if I could determine why the model was having such a hard time predicting denied applications. I used T-distributed stochastic neighbor embedding (t-SNE) to map my high dimensional data to two dimensions so that I could plot it and look at how applications should be classified. Figure 5 provides a plot in two dimensions of my data after a t-SNE transformation. Looking at this t-SNE plot, I can understand why the model is having a hard time predicting denied applications. There does not seem to be any clear pattern of where denied applications fall on the plot. In fact, they tend to overlap with the certified applications almost perfectly.

Figure 5: t-SNE Plot of 10,000 Application Records



The code used to generate this t-SNE plot can be found here: <https://github.com/Liptoni/Springboard/blob/master/H1B_Capstone/python/analysis_tsne.py>

# 6. Discussion

Obviously having a model that doesn’t predict the outcome of an event better than a coinflip is not ideal. With that said, I don’t think that we’ve come away from this process with nothing. The fact that nearly 97% of more than 2,790,000 LCAs between 2011 and 2016 were certified is telling. It means that there might be some outside factor that I am not privy to that narrows the scope of applicants before the LCA process. It may also mean that the certification of the LCA process might merely rely on completeness of the application. Remember that I am only looking at records that were complete. I am also only using a subset of all the possible fields on the application. My original dataset does not include every field from the LCA. It is possible that the denied applications that I am seeing, while complete for the records that I do have, are missing some information for features that I do not have access to. This could explain why the denied applications map so perfectly with the certified applications on the t-SNE plot as we are using only the features available to me.

If this were my business, my next steps would be to do more research into who decides whether LCAs are approved or denied and how they make those decisions. Many policy decisions like this have some transparency. Rarely do government agencies make decisions without clearly documenting why those decisions are made, especially in circumstances regarding immigration and potential racial bias. If I were going to give advice to a client today, I would advise them to make sure that every field in their application was completely filled out. I would suggest that they make sure that the position they are being hired for, or hiring for, has clear analogs in other companies in the area so that there is a direct comparison to make for the application. Finally, I would inform them that while most applications are certified, about 3% are denied and that they should prepare for that unlikely eventuality and not just assume that their application will absolutely be certified.