Predicting the Outcome of H-1B Visa Applications

CS229 Term Project Final Report

Beliz Gunel
bgunel@stanford.edu

Onur Cezmi Mutlu
cezmi@stanford.edu

1 Introduction

In our project, we aim to predict the outcome of H-1B visa applications that are filed by many high-skilled foreign nationals every year. We framed the problem as a classification problem and applied Naive Bayes, Logistic Regression, SVM and Neural Network in order to output a predicted case status of the application. The input to our algorithm is the attributes of the applicant which will be further explained in the following parts.

H-1B is a type of non-immigrant visa in the United States that allows foreign nationals to work in occupations that require specialized knowledge and a bachelor's degree or higher in the specific specialty [1]. This visa requires the applicant to have a job offer from an employer in the US before they can file an application to the US immigration service (USCIS). USCIS grants 85,000 H-1B visas every year, even though the number of applicants far exceed that number [2]. The selection process is claimed to be based on a lottery, hence how the attributes of the applicants affect the final outcome is unclear. We believe that this prediction algorithm could be a useful resource both for the future H-1B visa applicants and the employers who are considering to sponsor them.

2 Related Work

Although this area currently doesn't seem to be well-studied, we were able to find two reports that are relevant. Both of the reports used the same dataset we had from Kaggle. First report conducted a detailed data analysis and visualization for H-1B application distribution based on different input features such as location, salary, year and job type [5]. Although they had a prediction algorithm based on K-means clustering and decision trees, they provided prediction accuracies for only a small subset of job types instead of an average one. Overall, this report gave us good insight on the distribution of our data. Second report used AdaBoost, Random Forest, Logistic Regression and Naive Bayes to predict the case status [4]. We found some of their pre-processing steps for features clever and implemented them ourselves accordingly before feeding the features into our own machine learning algorithms. These pre-processing steps such as creating new features for success rate and number of applications and converting some features into one-hot-k representation will be further explained in the Dataset and Features section.

3 Dataset and Features

Our dataset is from Kaggle listed under the name "H-1B Visa Petitions 2011-2016 dataset" [6], and it initially included about 3 million data points. It contained 7 features and 1 label which can be examined in Figure 1.

CASE_STATUS	EMPLOYER_NAME	SOC_NAME	JOB_TITLE	FULL_TIME_POSITION	PREVAILING_WAGE	YEAR	WORKSITE
DENIED	ARAPURA INC	GENERAL AND OPERATIONS MANAGERS	OPERATIONS MANAGER	N	60000.0	2016.0	SOUTHPORT, CONNECTICUT
CERTIFIED	CITIUSTECH INC	COMPUTER SYSTEMS ANALYSTS	COMPUTER PROGRAMMER ANALYST	Y	89086.0	2016.0	ORANGE, CALIFORNIA

Figure 1: Two data points from the unprocessed dataset

We processed some of the existing features, created new features that we thought could be useful for prediction and discarded some features using the library Pandas [8]. In particular, we noticed that the JOB_TITLE feature represents highly redundant information with the SOC_NAME feature, therefore we discarded JOB_TITLE from the beginning. Also, we transformed both SOC_NAME and COMPANY_NAME features into the corresponding forms of success rate and total number of applications. Finally, we normalized all the features such that they had zero mean and unit variance. Our final list of features is further described in the following:

CASE_STATUS: We excluded the cases 'CERTIFIED-WITHDRAWN' and 'WITHDRAWN', since 'WITHDRAWN' decisions are either made by the petitioning employer or the applicant, therefore not predictive of USCIS's future behavior. We labeled 'CERTIFIED' cases as 1 and 'DENIED' cases as 0.
FULL_TIME_POSITION: Positions are given in "Full Time Position = Y; Part Time Position = N" format. We converted them to "Full Time Position = 1; Part Time Position = 0" format.
YEAR: Year in which application was filed. We converted the data into one-hot-k representation.
PREVAILING_WAGE: Prevailing wage is the average wage paid to employees with similar qualifications in the intended area of employment. We discarded the outlier terms and used the rest of the data as it was.
APPS_PER_EMPLOYER: We created a feature for the number of H-1B applications per employer, and discarded data points that are petitioned by an employer that has less than 4 applications. Although this processing step undesirably gets rid of applications filed by small companies, it significantly helps with cleaning up the misspelled company names.
COMPANY_SUCCESS_RATE: We created a feature for the success rate per employer.
APPS_PER_SOC: SOC stands for Standard Occupational Classification System, which is a federal occupational classification system. We created a feature for the number of H-1B applications per SOC type, and discarded data points with SOC types that appear less than 4 times in the data. This processing step undesirably gets rid of applications with uncommon jobs, but helps with cleaning up misspelled SOCs. Some additional processing for spelling mistakes was also conducted.
SOC_SUCCESS_RATE: We created a feature for the success rate per SOC type.
WORKSITE: Data is given in the "City, State" format. We only included "State" and converted the data into one-hot-k representation.

After the pre-processing steps described above, we split the training, dev and test sets 60:20:20. Training set had a total of 1.2 million examples. Due to the inherent bias in our dataset towards the "CERTIFIED" label, we created two versions of dev and test sets in order to make sense of the error analysis later on. First version of dev and test sets were both unbalanced, each consisting of 400K examples. More specifically, around 90% of the examples had a "CERTIFIED" label, mimicking the nature of the original dataset. Second version of dev and test sets were manually balanced by sampling "CERTIFIED" labeled examples roughly equal to the number of "DENIED" labeled examples. In the end, balanced dev and test sets had 100K examples each.

4 Methods

Our dataset is large, and it contains correlated features given the output. Also, the nature of the prior distribution of the features is unknown. Therefore, we thought Logistic Regression, SVM and Neural Network would be the most feasible techniques for our application. We also tried Naive Bayes due to its simplicity of implementation. Below, we provide brief descriptions for each of our techniques. Material is partially taken from the lecture notes of Prof. Ng. [7].

- **Naive Bayes:** Naive Bayes assumes all features are conditionally independent given labels. It calculates $p(x|y=1)$, $p(x|y=0)$ and p(y) by taking their maximum likelihood estimates in the joint likelihood of the data. While making a prediction, it calculates (1) both for p(y=1) and p(y=0) using the Bayes rule and compares the two.

$$p(y=1|x) = \frac{\prod_{i=1}^{m} p(x_i|y=1)p(y=1)}{\prod_{i=1}^{m} p(x_i|y=1)p(y=1) + \prod_{i=1}^{m} p(x_i|y=0)p(y=0)} \tag{1}$$

- **Logistic Regression:** Logistic regression uses the sigmoid function as its hypothesis as defined in (2) to make sure $h_\theta(x) \in (0,1)$. It makes the assumptions in (3) and finds the θ that maximizes the log likelihood of the data, $l(\theta)$, which is defined in (4).

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}} \tag{2}$$

$$\begin{aligned} P(y=1|x;\theta) &= h_\theta(x) \\ P(y=0|x;\theta) &= 1 - h_\theta(x) \end{aligned} \tag{3}$$

$$l(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(1-x^{(i)}) \tag{4}$$

- **Support Vector Machine:** Support Vector Machine tries to find a hyperplane that separates two different classes such that the distance from the closest data point to that hyperplane is maximized. The equivalent optimization problem can be framed as in (5).

$$\begin{aligned} &\min_{\gamma,w,b} && \tfrac{1}{2}\|w\|^2 \\ &\text{s.t.} && y^{(i)}(w^T x^{(i)} + b) \geq 1, \ i = 1, ..., m \end{aligned} \tag{5}$$

- **Neural Network:** Neural Networks (NNs) are complex multi-layer structures that consist of units called "neurons". Each neuron introduces a non-linearity with a chosen activation function $g(x)$ after weighing its inputs, and propagates the information to the next layer. At the output layer, a prediction $\hat{y}$ is made. The cost function NN tries to minimize is given in (6) and the vectorized version of forward propagation for a NN with input, one hidden and output layer is shown in (7). Finally, the parameters, W and b, in each layer l are updated as in (8) using back-propagation and some form of gradient descent. In our neural network implementation, we had 64 neurons at the input layer, 500 neurons at the hidden layer and 1 neuron at the output layer. We used sigmoid as our activation function.

$$\mathcal{L}(\hat{y}, y) = -\Big[(1-y)\log(1-\hat{y}) + y\log\hat{y}\Big] \tag{6}$$

$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= g(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ \hat{y} &= g(Z^{[2]}) \end{aligned} \tag{7}$$

$$\begin{aligned} W^{[l]} &= W^{[l]} - \alpha\frac{\partial\mathcal{L}}{\partial W^{[l]}} \\ b^{[l]} &= b^{[l]} - \alpha\frac{\partial\mathcal{L}}{\partial b^{[l]}} \end{aligned} \tag{8}$$

Experiments & Results

We used the SGDClassifier from the scikit-learn library [9] to implement Naive Bayes, Logistic Regression and soft-margin linear SVM. We preferred stochastic gradient descent over batch gradient descent due to the size of the dataset. After observing a big variance between training and test accuracy, we performed different types of regularization such as $\ell 1$, $\ell 2$ and ElasticNet (combination of $\ell 1$ and $\ell 2$) on the model to increase the test accuracy. In order to tune the hyperparameters, we initially split the training, dev and test sets 60:20:20 as part of the hold-out cross-validation process. Then, we retrained the best-performing models on 80% of the data and tested on the remaining 20% both for balanced and unbalanced test sets.

We adjusted the parameters like learning rate, number of iterations and regularization weight based on trial/error. Larger constant learning rates ($\alpha = 0.1 - 0.5$) seemed to perform better compared to the time-decaying learning rates, which might be attributed to the size of the dataset causing the initial point to be far from the optimum. Regularization weight was set to 0.0001 and $\ell 1$ to $\ell 2$ ratio was set to 0.15 for the ElasticNet regularization after trying several different values. Max number of iterations was set to 50.

For the Neural Network, we used our custom script. Due to the size of the dataset, we only used mini-batch gradient descent. We set the learning rate to 0.1 and the mini-batch size to 2000, which were both experimentally found to be good numbers. For the Neural Network with $\ell 2$ regularization, we set the regularization weight to 0.0005, again based on trial/error. We listed the top 5 performing methods, their training and test accuracies both on balanced and

unbalanced test sets, and finally precision and recall on the test sets in Table 1. We consistently have high recalls over different models, which shows that our algorithm doesn't have many false negatives. However, our precision metric is relatively lower across different methods, which demonstrates high number of false positives.

Table 1: Experimental Results

	Balanced				Unbalanced			
	Tr. Acc.	Test Acc.	Prc.	Rcl.	Tr. Acc.	Test Acc.	Prc.	Rcl.
Naive Bayes	94%	72%	73%	96.7%	94%	94%	98.6%	95.2%
Log. reg w/$\ell 1$	98%	74%	72%	99.9%	98%	98%	98.5%	99.2%
Linear SVM w/ElasticNet	98%	78%	75%	99.4%	98%	97%	98.1%	100%
Neural Net.	98%	76%	75%	99.9%	98%	96%	98.2%	100%
Neural Net. w/$\ell 2$	98%	82%	81%	99.9%	98%	97%	98.5%	100%

Tr. Acc = Training Accuracy, Test Acc. = Test Accuracy, Prc. = Precision, Rcl. = Recall
(Precision and Recall results are from the experiments with the test data)

Our results showed that the most predictive features are EMPLOYER_SUCCESS_RATE and PREVAILING_WAGE. One can infer from these results that the chance of being certified increases with the amount of wage and how successful your sponsor was in the previous H1-B applications. Also, we believe some features were irrelevant to the output, considering $\ell 1$ regularization produced very sparse feature vectors, and $\ell 2$ regularization assigned some features very small weights.

Overall, Naive Bayes performed better than we expected, considering applicant attributes are intuitively not independent given the outcome of their H-1B applications. Linear models with regularization did reasonably well, both on balanced and unbalanced test sets. However, Neural network with $\ell 2$ regularization outperformed all the other models.

Conclusion & Future Work

In the end, it is indeed possible to predict the outcome of H-1B visa applications based on the attributes of the applicant using machine learning. Out of the models we tried, Neural Network with $\ell 2$ regularization outperformed all the other models with 98% training accuracy and 82% test accuracy on the balanced test data. That's likely because neural networks are inherently better at explaining the complexities in the data. Overall, regularized models performed better by decreasing the variance between training and test accuracy through bounding the size of theta or the weight vector.

If we had more time and computational resources, there are several directions we could take to improve our prediction algorithm. First of all, we would try Neural Network with $\ell 1$ regularization, since we believe some features are actually irrelevant to the output as previously discussed. Also, we could adjust the depth of the neural network, number of neurons at each layer and possibly the network architecture. In addition, we could convert more features such as SOC_NAME into one-hot-k representation to achieve better accuracy. Finally, we could create more informative features such as Standard Industrial Classification codes of the companies through web crawling instead of using the given EMPLOYER_NAME and SOC_NAME features directly.

Contributions

During the pre-processing of the data, we divided the list of features into two, conducted processing steps separately and combined our results in the original data. For the modelling and experimentation part, we worked together while trying different algorithms, different regularization techniques and different hyper-parameters for the SGDClassifier. In addition, we decided to build our own neural network instead of using the off-the-shelf libraries and built the structure working together.

References

1. "H-1B Fiscal Year (FY) 2018 Cap Season," USCIS. [Online]. Available: https://www.uscis.gov/working-united-states/temporary-workers/h-1b-specialty-occupations-and-fashion-models/h-1b-fiscal-year-fy-2018-cap-season. [Accessed: 20-Oct-2017].

2. "High-skilled visa applications hit record high," CNNMoney. [Online]. Available: http://money.cnn.com/2016/04/12/technology/h1b-cap-visa-fy-2017/index.html. [Accessed: 20-Oct-2017].

3. "Using Text Analysis To Predict H-1B Wages," The Official Blog of BigML.com, 01-Oct-2013. [Online]. Available: https://blog.bigml.com/2013/10/01/using-text-analysis-to-predict-h1-b-wages/ . [Accessed: 20-Oct-2017].

4. "Predicting Case Status of H-1B Visa Petitions." [Online]. Available: https://cseweb.ucsd.edu/classes/wi17/cse258-a/reports/a054.pdf.

5. "H-1B Visa Data Analysis and Prediction by using K-means Clustering and Decision Tree Algorithms." [Online]. Available: https://github.com/Jinglin-LI/H1B-Visa-Prediction-by-Machine-Learning-Algorithm/blob/master/H1B\%20Prediction\%20Research\%20Report.pdf.

6. H-1B Visa Petitions 2011-2016 — Kaggle. [Online]. Available: https://www.kaggle.com/nsharan/h-1b-visa/data. [Accessed: 20-Oct-2017].

7. A. Ng, "CS229 Lecture Notes" [Online]. Available: http://cs229.stanford.edu/notes/. [Accessed: 14-Dec-2017].

8. Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

9. Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011)

10. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011)

11. Fernando Pérez and Brian E. Granger. IPython: A System for Interactive Scientific Computing, Computing in Science & Engineering, 9, 21-29 (2007)