Machine Learning Engineer Nanodegree

Capstone Project

Min Se (Brian) Chang

Dec. 20, 2018

I. Definition

Project Overview

Engineers around the world dream of working in the United States, especially the bay area, where coolest advancements in technology happen every day. Many talented engineers from foreign countries work hard to get temporary visas to start their American dream. H-1B is a visa issued by the U.S. government to foreign nations in special occupations. Employers who are willing to sponsor a foreign national need to file a labor condition application (LCA) and get it approved in order to be considered for a H-1B lottery for each year.

Problem Statement

Given complex procedures and paperwork that are required, most companies delegate H-1B applications to an immigration attorney. Each application needs to adhere to a proper format and satisfy conditions required for each job category in order to be successfully certified. The whole process is usually achieved by law professionals communicating with the applicant and the employer multiple times. A denied application not only incurs additional cost for each individuals' time, but could also potentially lower the applicant's chance to be successfully certified for the same position. Hence, it is imperative that each filed application satisfies requirements specified by USCIS and is comparable to other applications from the same industry.

For my capstone project, I created three different supervised classifier models (Random Forest, XGBoost, and Support Vector Machine), evaluated performance of each of them, and decided the final model based on performance and feasibility. I started by figuring out the set of features that are common between data source from different years, resolving naming conflict, and discarding infeasible and least useful features. After cleaning up records with empty values, I tried initial implementation of all three models and realized all models severely overfit to a dominant class due to highly imbalanced training data. I random-sampled records from the dominant class to achieve about the same balance between two classes, applied dimensionality reduction (PCA)

to the data set, and implemented three models again with default parameters. All three models turned out to be successful, and parameter tuning was applied to each of them to further optimize performance. XGBoost classifier was selected as the final model due to its high performance and efficiency. The final model was tested with manipulated data with random noise to demonstrate it is robust enough and not significantly affected by noisy and unseen data.

The set of features is texts entered in different sections of a filed LCA. A successful model enables individuals filing an application to quickly check whether the information they enter in the application is strong enough to be considered by USCIS. This model does not replace an immigration attorney by any means, but could be used as a quick sanity check after a proper application is created.

Metrics

Since the purpose of this model is to evaluate likelihood of an unsuccessful outcome, a pessimistic model which returns slightly lower chance of getting certified is much more tolerable than an overly optimistic one. An application with a false negative could be improved to increase confidence level, but a false positive result is certainly not acceptable. For this reason, my model is a high-precision model, which is calculated as follows:

$$Precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

However, this does not mean we can completely ignore recall and create a skewed model. To give more weights to precision than recall, F-beta score with $\beta = 0.5$ was used to evaluate performance of each model.

$$F = \frac{1.25 * \text{Precision} * \text{Recall}}{0.25 * \text{Precision} + \text{Recall}}$$

II. Analysis

Data Exploration

United States Department of Labor discloses annual statistics for research and analysis purpose [1]. Since this is actual data that was used by DOL to decide whether or not each application is certified, it was used as source data for all models.

The disclosure data is broken down per year, each of which is an excel spreadsheet format. Each year's spreadsheet has slightly different column names; some of them are merely renamed fields from previous years, and some are totally new fields added in newer years. For this assignment, I concatenated data from year 2015 to 2018

because pre-2014 data had noticeably fewer number of columns. After selecting (renamed) columns that are present in all years' records, the concatenated data has 2462248 samples before going through clean-up and sampling stage. The concatenated dataset consists of the following 40 columns.

FIELD NAME	DESCRIPTION
Case Number	Unique identifier assigned to each application submitted
Case Status	Status associated with the last decision ("Certified," "Certified-Withdrawn," Denied," and "Withdrawn")
Case Submitted	Date and time the application was submitted.
Decision Date	Date on which the decision was recoded
Visa Class	A H-1B visa technically has three more subsets. Possible values include 'H-1B', 'H-1B1 Chile', 'H-1B1 Singapore', 'E-3 Australian'.
Employment Start Date	Start date of employment
Employment End Date	End date of employment
Employer Name	Name of employer submitting labor condition application.
Employer Address	Geographical information of the Employer requesting temporary labor certification
Employer City	
Employer State	
Employer Postal Code	
Employer Country	
Employer Province	
Employer Phone	Contact information of the Employer requesting temporary labor certification
Employer Phone Ext.	
Agent Attorney Name	Contact information of the attorney filing an H-1B application on behalf of the employer
Agent Attorney City	
Agent Attorney State	
Job Title	Title of the job
SOC Code	Occupational code associated with the job being requested for temporary labor condition, as classified by the Standard Occupational Classification (SOC) System.
NAICS Code	Industry code associated with the employer requesting permanent labor condition, as classified by the North American Industrial Classification System (NAICS)
Full Time Position	Boolean field (Y/N) indicating full-time and part-tome position
Prevailing Wage	Prevailing Wage for the job being requested for temporary labor condition (continuous integer values)
PW Unit of Pay	Unit of pay for prevailing wage (Hour, Week, Bi-Weekly, Month, Year).
PW Wage Level	Prevailing wage level ("I", "II", "III", "IV" or "N/A")
PW Wage Source	prevailing wage source ("OES", "CAB", "DBA", "SCA", "Other")
PW Wage Source Year	Year the Prevailing Wage Source was Issued
PW Wage Source Other	Detailed text field for "Other" for wage source
Wage Rate Of Pay From	Wage offered by the employer for the position (continuous float value)
Wage Rate Of Pay To	Maximum wage the employer is willing to pay for the position (continuous float value)

Wage Rate Of Pay	Unit of pay for wage offered by employer (Hour, Week, Bi-Weekly, Month, Year).
H-1B Dependent	Boolean field (Y/N) indicating whether the employer is hugely depending on H1B employees
Willful Violator	Boolean field (Y/N) indicating whether the employer committed a willful failure or a misrepresentation of a material fact in the past. A willful violator employer must provide additional evidences for any LCA it files and could be subject to random investigations
Worksite City	Geographical information of worksite
Worksite Country	
Worksite State	
Worksite Postal Code	

Table 1: List of Common Features in Disclosed Data from Year 2014 to 2018

Fields “Prevailing Wage”, “Wage Rate of Pay From”, and “Wage Rate of Pay To” are continuous floating-point numbers. All other fields are categorical fields with possible values as described above.

Exploratory Visualization

Below is initial observation of features from the dataset.

- While most categorical features (excluding geographical and contact information) have only a few values, the following features have more than 10 unique values:
    - “SOC Code” and NAICS Code have 1250 and 3508 unique values, respectively.
    - “Job Title” and “Employer Name” have 169572 and 148747 unique values, respectively, even after removing special characters and converting to uppercased strings.
- Three columns (“Wage Rate of Pay From”, “Wage Rate of Pay To”, and “Prevailing Wage”) representing wage are the only continuous features in the dataset. Given that feature “Wage Rate Of Pay To” an optional field, it is missing from 21% of samples and set to 0 from 59% of samples.

As shown in Figure 1, the dataset is significantly imbalanced in that only 1.47% of samples have label “DENIED”. A careful sampling technique is required in order to prevent the models from overfitting to dominant class.

```
status_stats = df.groupby('CASE_STATUS').size().to_frame()
status_stats['% Distribution'] = status_stats[0] / status_stats[0].sum() * 100.0
status_stats.index.rename("Case Status", inplace=True)

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

status_stats['% Distribution'].plot(kind='barh', colormap='tab10',
figsize=(12,3)).invert_yaxis()
plt.legend(loc="lower right")
plt.title('% Distribution of Samples with Each Class')
plt.xticks(np.arange(0,110,step=10))
plt.xlabel('% Distribution')
```

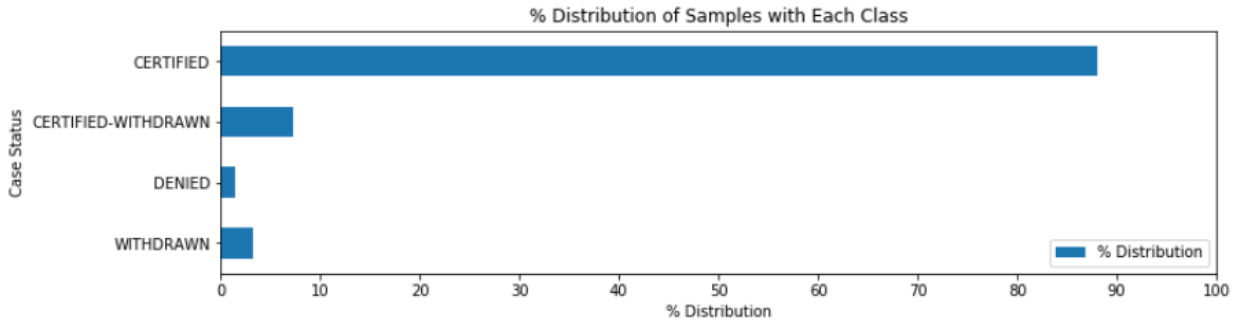

Figure 1: % Distribution of Samples with Each Class

Figure 2 shows distribution of top 30 occupations by count. SOC Codes 15-1131, 15-1121, 15-1131, and 15-1199 has the most number of applications, all of which is related to computer software development. Applications for occupations other than the top 30 account for only 15.3% of total applications, which means a majority of decisions made for LCAs are for the top 30 occupations.

Figure 3 is zoomed-in version of figure 2; this plot tells shows a very valuable information in terms of feature selection; occupations such as 10-1011 (chief executives), 10-1021 (general and operations managers), and 17-1011 (architects, except landscape and naval) show 100% denial rate, whereas occupations such as 17-2041 (statisticians) and 19-1042 (medical scientists), have near 0% denial rate. This clearly shows that SOC code could serve as a useful feature in separating out samples into two classes. Given that samples with denied case status is only 1.47% of total records, the number of samples for SOC codes with 100% denial rate is less than 0.1% of total. However, the high denial rate is not an artifact of under-sampling since there are more than 200 samples in all top-30 SOC codes.

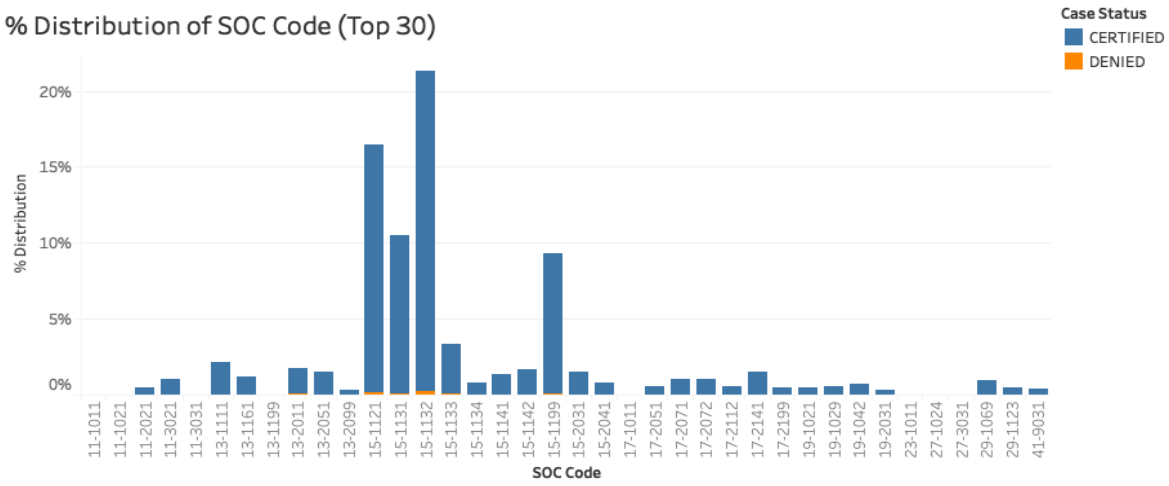

Figure 2: % Distribution of SOC Code for Certified and Denied Samples

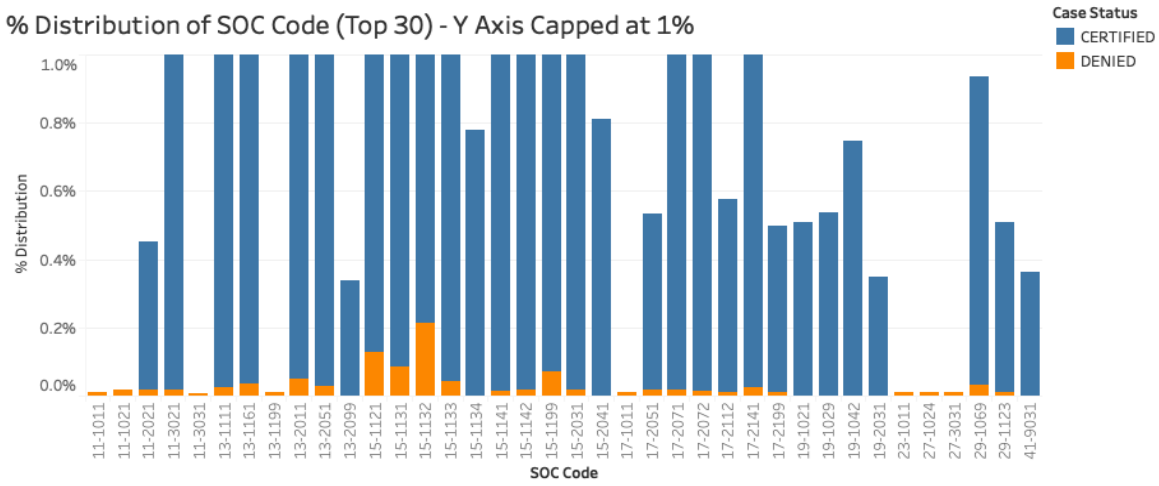

Figure 3: % Distribution of SOC Code for Certified and Denied Samples (Zoomed-in at 1%)

NAICS code also seems to show similar trends as SOC code. As shown in Figure 4, industry 541511 (custom computer programming services) alone accounts for 37.7% of total samples, followed by 541512 (computer systems design services) and 611310 (academies, college or university), 6.0% and 4.1%, respectively. Figure 5, enlarged version of Figure 4, shows that industries such as 522320 (financial transactions processing, reserve, and clearinghouse activities) and 551112 (offices of other holding companies) have near 0% denial rate. In contrast, industries including 541110 (offices of lawyers), 541310 (architectural services), and 621340 (offices of physical, occupational and speech therapists, and audiologists) have 100% denial rate with more than 189 samples. Hence, NAICS code definitely needs to be selected as one of the features that will be used to train and test the dataset.

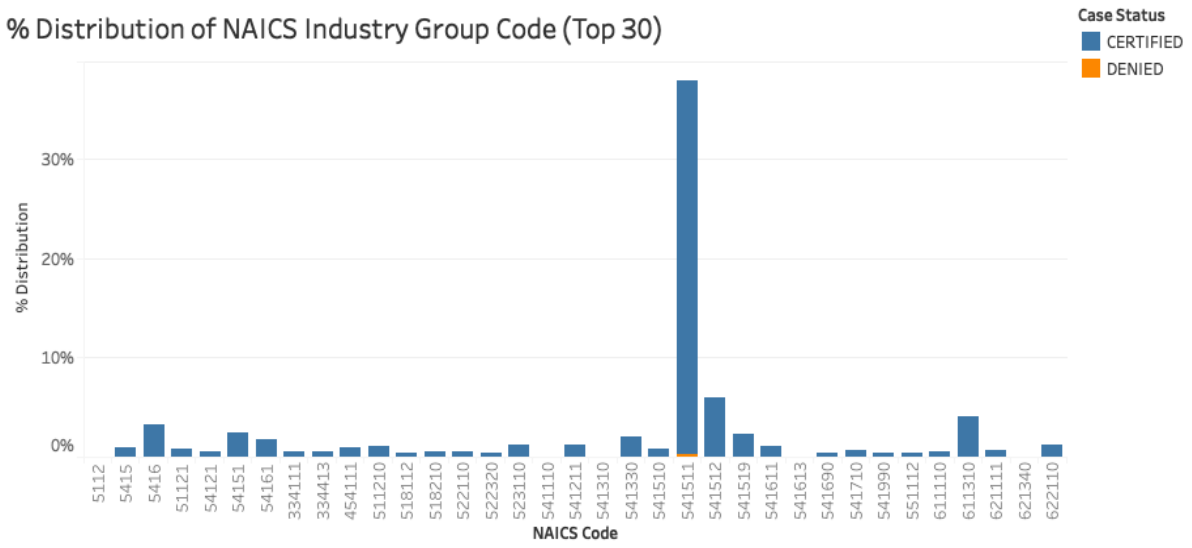

Figure 4: % Distribution of NAICS Code for Certified and Denied Samples

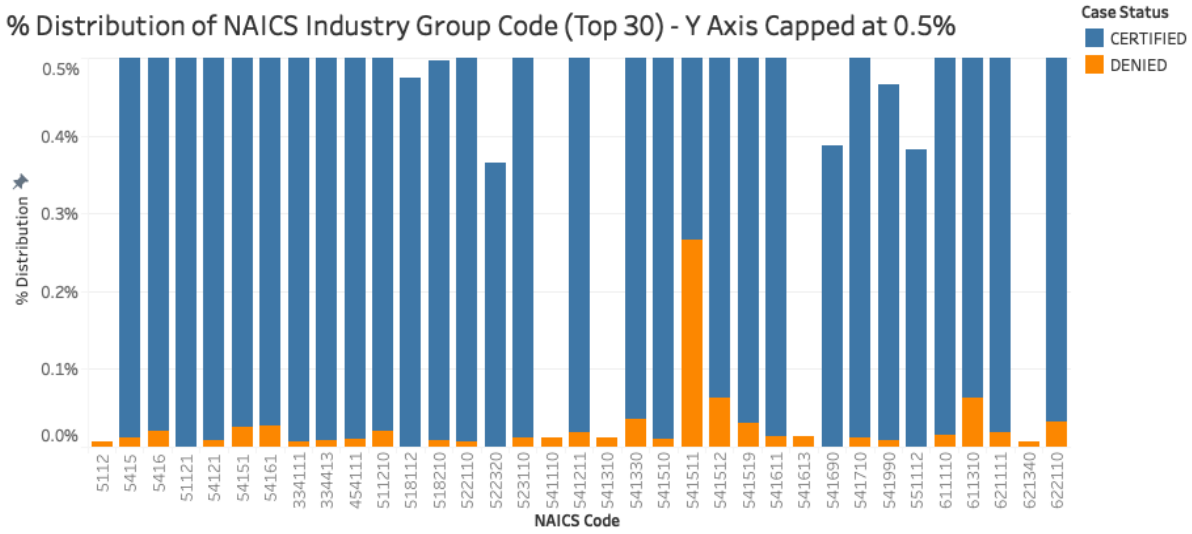

Figure 5: % Distribution of NAICS Code for Certified and Denied Samples (Zoomed-in at 0.5%)

Finally, Figure 6 shows that continuous variables “Prevailing Wage” and “Wage Rate Of Pay From” are both normally distributed and show very similar bell-shape. This is confirmed by calculating basic statistics using Pandas, after removing outliers as follows:

```
>>> df_raw[(df_raw['WAGE_RATE_OF_PAY_FROM']>df_raw['WAGE_RATE_OF_PAY_FROM'].quantile(0.05)) &
...         (df_raw['WAGE_RATE_OF_PAY_FROM']<df_raw['WAGE_RATE_OF_PAY_FROM'].quantile(0.95))]\
...          .WAGE_RATE_OF_PAY_FROM.describe()
count    2.120200e+06
mean     8.454033e+04
std      2.462791e+04
min      7.000910e+01
25%      6.696030e+04
50%      7.965402e+04
75%      9.889339e+04
max      1.526311e+05
>>> df_raw[(df_raw['PREVAILING_WAGE']>df_raw['PREVAILING_WAGE'].quantile(0.05)) &
...         (df_raw['PREVAILING_WAGE']<df_raw['PREVAILING_WAGE'].quantile(0.95))]\
...          .PREVAILING_WAGE.describe()
count    2.291004e+06
mean     7.498997e+04
std      2.250399e+04
min      4.224030e+01
25%      6.102075e+04
50%      7.311200e+04
75%      8.925259e+04
max      1.290178e+05
Name: PREVAILING_WAGE, dtype: float64
```

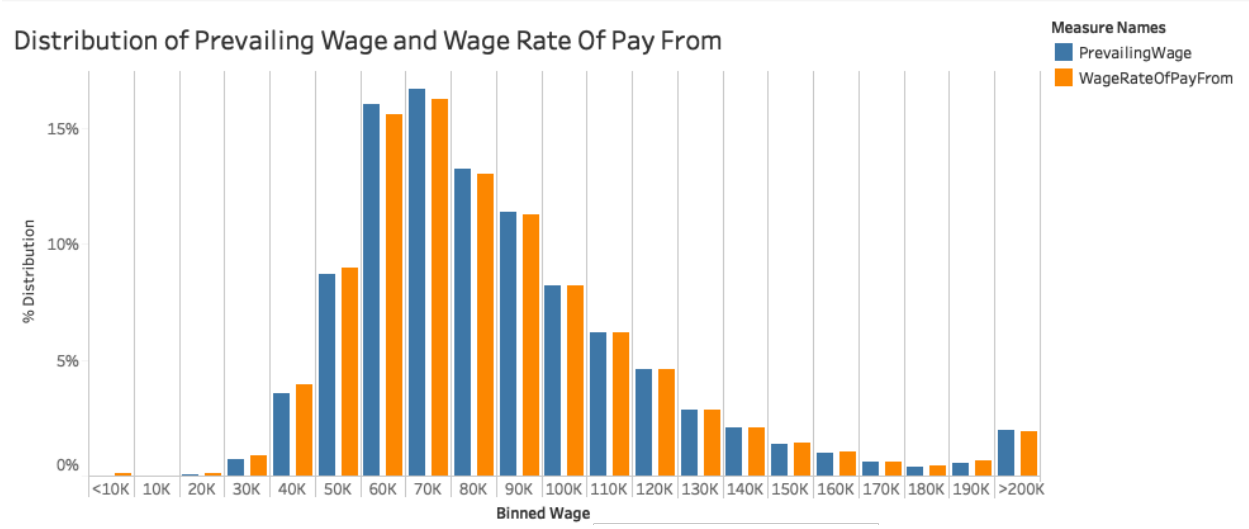

Figure 6: Distribution of Prevailing Wage and Wage Rate Of Pay From

This means it might not be necessary to use raw values from both of these features. In general, “Wage Rate Of Pay From” is similar or higher than “Prevailing Wage”, with the exception of range between 60K and 90K. This is within ±1σ (standard deviation), which means some samples around the mean have lower wages than prevailing wage but samples from most other quantiles have greater offered wage than prevailing wage. Given that offered wage needs to be higher than prevailing wage, we could deduce that a lot of denied applications are within this range.

Algorithms and Techniques

For this project, I used the following algorithms to predict outcome of each LCA application.

- XGBoost: eXtreme Gradient Boosting is known to push the limits of computing power for gradient boosting by utilizing multiple threads. It is believed to take significantly less time to train than other boosting models and efficient in decreasing chance of overfitting by performing L1 and L2 regularization. It creates a strong classifier based on many weak classifiers, where “weak” and “strong” refer to how closely correlated each learner is to the actual target. Since it adds models on top of each other iteratively, errors from previous “weaker” learners could be corrected by the “stronger” predictor in the next level. This process continues until the training data is accurately predicted by the model. Since we’re dealing with a data that has a lot of noise, it is definitely worth trying.
- Random Forest: By constructing multiple simple trees, random forest model is known to alleviate the tendency of overfitting for skewed data. It is also known to handle a large data set with very high dimensionality well. It is trained via bagging method, which randomly sub-samples the training data, fits a decision tree model to each smaller subset (using the best split point), and aggregates the predictions at the end. I trained and tested this model using the entire one-hot encoded dataset to compare the result with the other two models, both of which needed dimensionality reduction due to model complexity.
- Support Vector Machine: SVC with RBF kernel is known to perform very well when working on points that are not linearly separable. As shown in figure 7, it uses a kernel trick to transform the training set into a higher dimension and finds a separation boundary between classes (called a hyperplane). Hyperplanes are identified by locating support vectors (data points that would change the dividing hyperplane if removed) and their margins. Since linearity between features is not guaranteed, SVC is definitely worth trying despite very expensive computational power. Given the expensive computational cost, dimensionality of training and test set features was reduced by applying PCA with 400 features.

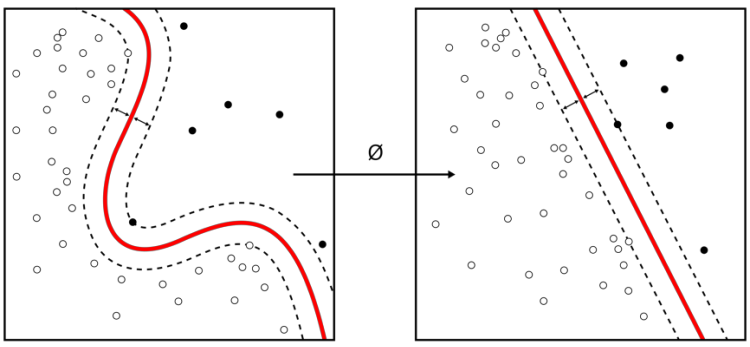

Figure 7: Transformation and Hyperplane of a Support Vector Machine [2]

Benchmark

The law requires that an employer sponsoring a foreign national employee pay at least the prevailing wage rate for the offered job [3]. Also, generally speaking, the purpose of hiring foreign workers is to accommodate special occupations that cannot be fulfilled by locals. For this project, two simplified calculations were used as benchmark.

- Naïve classifier which predicts a case will be denied if offered wage is less than prevailing wage.
- Decision Tree classifier which only the following features:
    - Whether offered wage is less than prevailing wage
    - Prevailing wage
    - SOC Code
    - Company name

Result of each model will be compared against these two models and ideally should perform better than them.

III. Methodology

Data Preprocessing

As discussed above, feature “Case Status” is the label which all models take and try to predict. Although there are four classes in total (“Certified”, “Certified-Withdrawn”, “Denied”, “Withdrawn”), there are actually only two true classes. “Certified-Withdrawn” was renamed to “Certified” since these are certified applications that were voluntarily withdrawn. Samples with label “Withdrawn” were excluded from the input data since they did not go through case decision process. After making these changes, there were 1479869 samples, 2.19% with label “Denied” and 97.81% with label “Certified”.

As described in Data Exploration section, only columns that are present in all years’ disclosed data were selected. Strangely enough, data from 2016 had only one column for wage, whose range was specified by concatenating two numbers with “-”. This needed to be broken down into columns “Wage Rate Of Pay From” and “Wage Rate Of Pay To”. After resolving inconsistent names (e.g. from PW_WAGE_SOURCE to PW_SOURCE and from NAIC_CODE to NAICS_CODE), the following processing was done:

Excluded Features

After carefully reviewing values of all features, the following were excluded from the dataset:

- Attorney name, attorney city, attorney state: Common reasons for LCA denial include filing incomplete LCA, failure to pay filing fees, and insufficient job description [4]. My initial assumption was that these trivial mistakes would significantly decrease if the application is handled by an immigration attorney. However, as seen from Figure 8, samples with both labels showed almost the same distribution of whether the attorney name is missing. This tells us that whether these fields are empty don't really make a difference in terms of case decision, so features "Agent Attorney City", "Agent Attorney Name", and "Agent Attorney State" are excluded.
- Case number, case submitted date, case decision date: As disclosed in DoL website, most applications got processed in a timely manner, and there wasn't much variation in terms of turnaround time. Case number is a unique identifier for each case, which doesn't help us in generalizing to a pattern.
- Employer's street address, city, state, phone, phone extension, postal code, province, country: Additional geographical and contact information of the employer are unique to each employer and LCA decision is not dependent on location.
- Case number, case submitted date, case decision date: As disclosed in DoL website, most applications got processed in a timely manner, and there wasn't much variation in terms of turnaround time. Case number is a unique identifier for each case, which doesn't help us in generalizing to a pattern.
- Employer's street address, city, state, phone, phone extension, postal code, province, country: Additional geographical and contact information of the employer are unique to each employer and LCA decision is not dependent on location.
- Worksite city, state, postal code: Similarly, additional geographical information for worksite doesn't add any value to LCA decision process.
- Prevailing wage source other: This is additional text field for "other" option for prevailing wage source. Since there are 10663 unique values, it's not feasible to one-hot encode this field.
- Prevailing wage source year: This is the year the prevailing wage source was issued. Not useful since applicants would pull the latest source for most applications.
- Employer name: There are 148747 unique employers even after performing basic cleanups (removing non-alphanumeric characters and casting to upper-case). It's not feasible to apply one-hot encoding to these many fields and applying label encoding would introduce incorrect rank between employers (e.g. name "EmployerX" is not any higher or lower than "EmployerY").
- Job title: SOC Code represents occupation of the position very well and this plain-text column is very prone to errors.

- Wage Rate Of Pay To: As shown in analysis section, this optional feature is missing from 21% of samples and set to 0 from 59% of samples.
- PW_WAGE_LEVEL: This represents level of prevailing wage (I, II, III, IV), but data for year 2016 is missing this column and it cannot be inferred from other years' data.
- PW Unit Of Pay: This is only useful to adjust non-yearly wages to yearly wages. Removed after converting all wages to yearly wages.
- Wage Rate Of Pay From: As seen from exploratory visualization section, it has very similar distribution as prevailing wage. After creating a new feature by subtracting offered wage and "Prevailing Wage" as discussed below, this raw value was removed from the dataset.

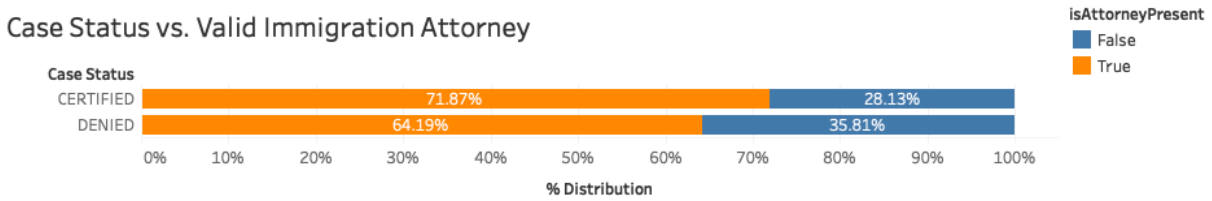

Figure 8: Case Status vs. Valid Immigration Attorney

Feature Engineering

As discussed above, instead of using raw values of "Wage Rate Of Pay From", a Boolean feature named "WageLowerThanPW" was created. It is calculated by taking a difference between "Wage Rate Of Pay From" and "Prevailing Wage" and assigning 1 if the difference is negative and 0 otherwise. Figure 9 shows that 94.5% of cases with this feature set to true are denied.

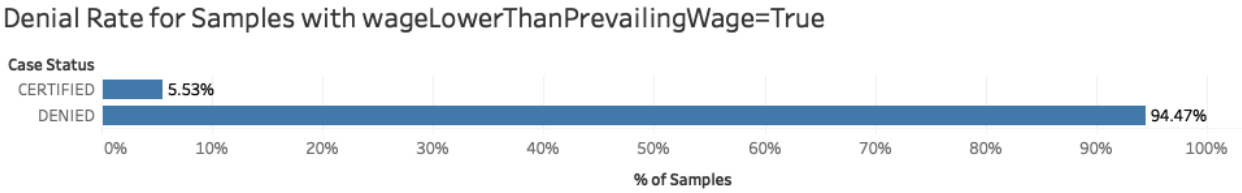

Figure 9: Denial Rate for Samples with wageLowerThanPrevailingWage=True

Feature Selection

Given there are significantly more samples with "Certified" label, any sample in this class that's missing any of selected features were excluded.

- Case status: As explained at the beginning of data preprocessing section, samples with "Withdrawn" status were excluded and class "Certified-Expired" was renamed to "Certified". This will be the label used for all models.
- SOC Code: It is a federal statistical standard code for classifying occupations by categories. A valid SOC code is "xx-xxxx", but there are quite a few values with

invalid values. Only values in format of “xx.xxxx”, “xxxxxx”, and “xxxxx” were corrected by replacing “.” with”-“ or adding it to second index, and the rest were assigned null. For samples with denied labels, value for “Job Title” was filled in for nulls.

- NAICS Code: If the values are present, most of them are either 5~6-digit integers or floating point numbers (integers followed by trailing “.0x” mostly due to incorrect format in Excel). These values are cased to float and then to int to make sure only digits are present. If this fails, null was filled in. For samples with denied labels, values for “Employer Name” was filled in for nulls.
- Visa class: For samples with denied labels that are missing these values, “H-1B” was filled in, which 99.7% samples have.
- Full-time position: For samples with denied label, “N” was filled in for nulls.
- H1B dependent: For samples with denied label, “N” was filled in for nulls.
- Prevailing wage: For consistency, wage for each year was corrected to yearly wage using “PW Unit of Pay”, assuming 40-hour/week shift for 4 weeks, 12 months. Also, all wages before 2018 were adjusted to today’s wage by applying by inflation rate. For samples with denied label, median value was filled in for nulls.
- PW_WAGE_SOURCE: prevailing wage source (OES, CAB, DBA, SCA, etc). For samples with denied label, “Other” was filled in.
- WILLFUL_VIOLATOR: For samples with denied label, “N” was assigned for nulls.
- Wage unit of pay: For samples with denied label, “Year” was filled in, which 95% of samples have.
- Wage lower than prevailing wage: The derived field discussed in feature engineering subsection. “False” is assigned for samples with denied label and missing wage.

All selected categorical features were one-hot encoded using `get_dummies` function from `pandas`. Although this significantly increases the number of features, it is the only option since label-encoding would introduce incorrect rank between categorical variables. Given that feature “Prevailing Wage” is widespread between minimum and maximum, this continuous feature was normalized using `MinMaxScaler` from `sklearn`.

Initial Implementation

Using the cleaned data, I initially implemented the aforementioned models with default parameters. As a first attempt, the dominant class (Certified) was used without any down-sampling, as I was under the impression that ensemble methods are able to handle imbalanced data efficiently. Instead, stratified sampling was used to retain ratio between the two classes when the dataset was split into training and test set.

```
# Load cleaned dataset generated by cleanup.py
df = pd.read_pickle('/Users/ml/Desktop/Udacity_ML_Capstone/data/H1B_15-18_FINAL.pickle')
# Scale continuous feature and one-hot encode categorical features
scaler = MinMaxScaler()
X = df.drop(columns='CASE_STATUS')
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['CASE_STATUS'])
X[['PREVAILING_WAGE']] = scaler.fit_transform(X[['PREVAILING_WAGE']])
X = pd.get_dummies(X)
# Split X and y with stratify option
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2)

# Train a Decision tree classifier and predict output
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
rfc_pred = rfc.predict(X_test)from sklearn.tree import DecisionTreeClassifier

# Train a XGBoost Classifier and predict output
from xgboost import XGBClassifier
xgb = XGBClassifier(random_state=99)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
```

For the initial modeling, no dimensionality reduction was used to see if it's feasible to use the entire encoded dataset. Training time turned out to be very long (10s of hours) for random forest and XGBoost models, but I was able to train and evaluate these models successfully. SVC was excluded from initial modeling since it's not feasible to train it using 5370 features. Below is results from the initial modeling:

Model	Precision	Recall	Accuracy	F-Beta score	Confusion Matrix
Random Forest	0.64	0.34	0.98	0.54	[[296410 1438] [4915 2540]]
XGBoost	0.91	0.27	0.98	0.62	[[297657 191] [5431 2024]]

Table 2: Initial Results from Random Forest and XGBoost Models with Default Parameters

At first, the accuracy and precision for all models may look very good, but their confusion matrices tell us that they are performing horribly. Although all models are correctly detecting samples with "Certified" label very well (over 90% accuracy), they are doing a very bad job classifying samples with "Denied" label (less than 35% accuracy). This is a classic example of overfitting, which is mainly caused by the high imbalance of dataset despite introducing stratified sampling and selecting ensemble models that are less prone to class imbalance. The unbelievably high accuracy is merely due to the fact that 97.8% of samples are in the first class.

The encoded dataset had 5370 features, due to a large number of unique values for categorical features “SOC Code” and “NAICS Code”. Given that feature importance from random forest and XGBoost classifier decays down after the first 20 features (please refer to Jupyter notebooks `rfc_initial.ipynb` and `xgboost_initial.ipynb` for details), a dimensionality reduction technique could be considered to make the model simpler. Not all features are contributing to models’ overall performance, and it is not feasible to perform parameter tuning using this dataset, not to mention training step for SVC model didn’t finish even after several days.

Refinement

In order to overcome the overfitting situation, samples with “Certified” label was randomly sampled at 2% to have about the same balance between two classes. Also, PCA was applied with 400 features for SVC to make it feasible to run on a conventional Mac Pro.

I also noticed that XGBoost algorithm utilized only one during training and testing steps on a 12-core Mac Pro. This was surprising, since XGBoost is known for very efficient parallelization (multi-threading). After doing lots of research, I found out this is because default Apple Clang compiler that ships with the OS does not support OpenMP, which is what XGBoost uses under the hood to parallelize jobs [5]. To enable multi-threading, I manually installed gcc7 and specified the compiler version in makefile and built the library manually. I also had to specify the number of threads (nthread parameter) when initializing the model in order to initiate parallelization, as the default parameter “-1” didn’t enable parallelization.

```
from xgboost import XGBClassifier
# Create 11 threads to utilize 11 cores
xgb = XGBClassifier(random_state=99, nthread=11)
```

After completing these steps, I confirmed that my computer was finally using 11 cores for training and testing by checking CPU usage in Activity Monitor (which said ~1100% CPU usage for Python 2.7). Random forest classifier, XGBoost classifier, and SVC were created using default parameters again and trained and tested using the new dataset. Results from the second attempt is as follows.

Model	Precision	Recall	Accuracy	F-Beta	Confusion Matrix
Random Forest	0.76	0.72	0.73	0.752	[[4927 1699] [2079 5383]]
XGBoost	0.78	0.67	0.72	0.754	[[5200 1426] [2465 4997]]
SVC (with PCA)	0.78	0.66	0.72	0.754	[[5237 1364] [2569 4918]]

Table 3: Results from Sampled Dataset and Default Parameters

The result for all three models are considerably better than the initial result. Not only are F-Beta score, precision, and recall more reasonable, the confusion matrix tells us that number of times each classifier predicts a class correctly is at least twice higher than incorrect predictions. From the initial analysis, the number of false-negatives were at least twice higher than the number of true-positives.

The following parameters are believed to play a great role in impacting performance of each model:

- Random Forest
    - n_estimators: This is the number of trees in the forest. It plays a crucial role in reducing risk of overfitting and making the predictions more stable and stronger.
    - max_depth: It controls how deep each tree can grow. To prevent an overfitting tree, it's very important to prune the tree properly, especially when dealing with noisy data.
    - max_features: Since it is the number of features to consider when looking for the best split, it's worth experimenting integer values of these instead of default 'auto' (so that we have some control over split decision).
- XGBoost
    - max_depth: Same as Random Forest, it is crucial to prune trees properly.
    - n_estimators: Same as Random Forest, this parameter controls the number of trees.
    - gamma: It acts as a regularization parameter, which controls minimum loss reduction required to make a split. It decides how conservative the algorithm could be when making a split.
    - reg_lambda: This is L2 regularization term on weights, which is effective in reducing chance of overfitting.
    - reg_alpha: This is L1 regularization term on weights, which not only helps with overfitting but also leads to feature selection.
    - subsample: Controls row subsampling of training dataset.
    - colsample_bytree: Controls subsample ratio of columns when constructing trees.
- Support Vector Classifier
    - C: It represents how much we're willing to avoid misclassifying each training sample; small margin for large C will have higher accuracy but may lead to overfitting, whereas large margin (for small C) may generalize better but could lead to lower accuracy.
    - gamma: For a RBF kernel, it controls how spread the decision region is (low gamma leads to very broad region, whereas very high gamma leads to "islands" of decision boundaries, which could result in overfitting).

In order to find a set of parameters that performs better than default parameters, I ran parameter tuning for all three models. Given the complexity of the models, randomized search with different iterations (depending on model complexity) was used instead of exhaustive grid search. Accuracy was used for scoring mechanism for simplicity, and PCA with 400 features was applied for SVC and XGBoost.

```
# Example for SVC
...
svc = SVC(kernel='rbf')
params={
    'C':[1,10,100,1000],
    'gamma':[1,0.1,0.001,0.0001],
}
random_search = RandomizedSearchCV(svc, param_distributions=params, scoring='accuracy',
    n_iter=n_iter_search, cv=5)
random_search.fit(X, y)

# Iterate through random_search.cv_results_ and get the one with the highest mean validation
#score (Please refer to Jupyter notebooks for RandomizedSearch).
```

Below is the set of parameters and the result they returned.

Model	Precision	Recall	Accuracy	F-Beta	Confusion Matrix
Random Forest (n_estimators=290, max_features=18, max_depth=80)	0.82	0.58	0.76	0.765	[[8787 913] [3070 4392]]
XGBoost (reg_alpha=0.06, colsample_bytree=0.8, n_estimators=1000, subsample=0.9, reg_lambda=0.07, max_depth=3, gamma=2)	0.80	0.68	0.74	0.771	[[5289 1303] [2385 5111]]
SVC (gamma=0.001, C=100)	0.79	0.68	0.74	0.768	[[5350 1302] [2406 5030]]

Table 4: Results from Sampled Dataset and Optimized Parameters

IV. Results

Model Evaluation and Validation

As shown from Table 4, all three models perform almost equally well in predicting case outcome based on other features. Among the three, XGBoost model was chosen as the final model because it had the highest F-beta score with $\beta = 0.5$, which is the target metric used for evaluation. Also, owing to its very efficient parallelization implementation, it takes much shorter to train than the other two classifier models.

Result from the final model has a good balance between precision, accuracy, and recall, which resulted in high F-beta score. The score is reasonable for an ensemble classifier with relatively small number of estimators (1000), basic pruning criteria, and relatively small regularization parameters. It is believed to be able to predict outcome of real-world LCA as long as the data entered in the application is realistic (what we would see from a real LCA).

The model's sensitivity was tested using three different ways

- Given the number of samples with "Denied" label was very limited, I actually ended up using all available ones for training and test set. Since samples with "Certified" label were sampled at 2%, I randomly sampled 10000 of the 98% of data that the model has never seen and tried predicting the case decision using the model that was trained with all the data it has previously seen. The model returned 80% accuracy for these sampled data with "Certified" label.
- As a previous H-1B employee, I tested the model with inputs from my LCA (assuming the employer, Apple Inc., is neither "H-1B Dependent" nor a "Willful Violator") and got a "Certified" outcome.
- To see if the model is robust enough, I manipulated the test set by multiplying "Prevailing Wage" by a random number between 0.95 and 1.05 to every sample. I also negated all Boolean fields with 5% probability (please refer to the code snippet below). When this new test data was tested on the trained model, performance degradation was negligible with the exception of precision (which decreased by 5%) as shown in table 4. F-beta score only decreased by 3%. The result still looks very reasonable despite the introduced noise to all data points, which proves the model generalizes reasonably well to unseen data and small changes in testing data doesn't affect the outcome completely.

```
random_constants = np.random.uniform(0.95,1.05,X_test.shape[0])
X_test['PREVAILING_WAGE'] = X_test['PREVAILING_WAGE'] * random_constants

choices = [True, False]
prob = [0.05, 0.95]
X_test['WAGE_LOWER_THAN_PW'] = X_test['WAGE_LOWER_THAN_PW'].apply(
    lambda x: not x if np.random.choice(choices, 1, p=prob)[0] else x)
# Toss unfair coins
flipped_indices = list(np.nonzero(np.random.choice(choices, X_test.shape[0], p=prob))[0])
if flipped_indices:
    X_test.ix[[0,1,2,3],'FULL_TIME_POSITION_N'] = \
        X_test.ix[flipped_indices,'FULL_TIME_POSITION_N'].apply(lambda x: 1-x)
    X_test.ix[flipped_indices,'FULL_TIME_POSITION_Y'] = \
        X_test.ix[flipped_indices,'FULL_TIME_POSITION_Y'].apply(lambda x: 1-x)
...
```

	Precision	Recall	Accuracy	F-Beta	Confusion Matrix
Scores	0.75	0.70	0.72	0.744	[[4961 1700] [2198 5229]]

Table 5: Impact of Randomly Manipulated Test Data on Final Model

Justification

Result from the final model definitely beats the naïve predictor benchmark. Even though the simple decision-tree benchmark did considerably better than I expected, the final model scored higher precision and F-beta score. It was also proven to generalize to unseen data pretty well and a slight change in test data did not affect the overall performance of the model greatly.

Model	Precision	Recall	Accuracy	F-Beta	Confusion Matrix
Naïve Predictor (Denied if Wage Lower than PW, Certified Otherwise)	0.996	0.185	0.64	0.531	[[9675 6] [6097 1384]]
Decision Tree (Based on 4 Features)	0.72	0.64	0.73	0.701	[[7896 1846] [2697 4723]]
XGBoost (Parameters from RandomizedSearchCV)	0.80	0.68	0.74	0.771	[[5289 1303] [2385 5111]]

Table 6: Results Comparison between Benchmark Models and Final Model

The final model demonstrated to work well with unseen data in model evaluation and validation section, which means it is generalizing to pattern of training data instead of overfitting to a specific environment. This solution has solved the problem described in problem statement section since it is expected to consistently predict case decision based on the set of entered text information, as long as the data is not from a completely different environment (for example, from a future year by which there was a significant H-1B policy change in wage and other eligibility requirements).

V. Conclusion

Free-Form Visualization

As discussed above, the final model predicts outcome of disclosed H-1B LCA from 2014 to 2018 with around 70% accuracy. The model is able to consistently predict the outcome regardless of which class the sample lies in, and definitely does better job than a random classifier. This is clearly shown in figure 10 shows ROC curve of the final model, with AUC of 0.74.

```
from sklearn.metrics import roc_curve, auc,recall_score,precision_score
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

FlasePositive, TruePositive, _ = roc_curve(y_test, xgb_pred)
roc_auc = auc(FlasePositive, TruePositive)
plt.figure(figsize=(8,8))
lw = 2
plt.plot(fpr, tpr, color='red',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False-Positive Rate')
plt.ylabel('True-Positive Rate')
plt.title('ROC curve')
plt.legend(loc="best")
plt.show()
```

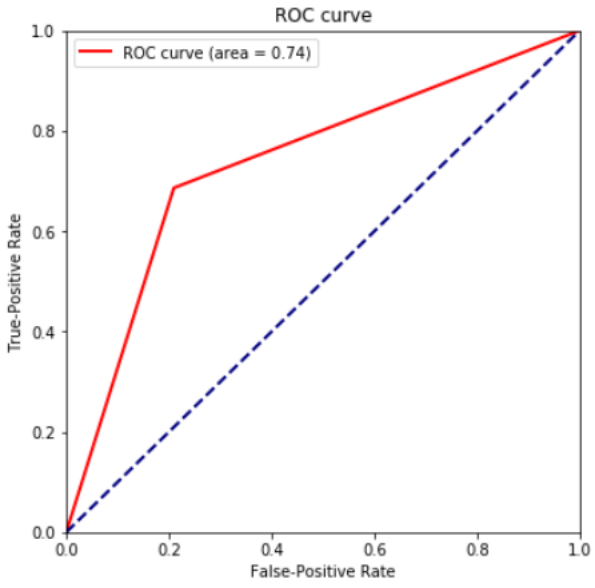

Figure 10: ROC Curve of the Final Model

Among the 11 selected (and created) features, “Visa Class”, “Wage Lower Than PW”, “Prevailing Wage”, and “H1B Dependent”, seems to be contributing most to the success of the model. Figure 11 shows top 25 feature importance scores from the trained final model. Please note this is for top 25 one-hot encoded features, which means only a few of the 11 selected features are shown in this figure.

```
importances = []
import operator
nonzero_features = {}
for i in range(len(xgb.feature_importances_)):
    if xgb.feature_importances_[i] > 0.0:
        nonzero_features[list(X_train.columns)[i]] = xgb.feature_importances_[i]
sorted_result = sorted(nonzero_features.items(), key=operator.itemgetter(1))
for i in range(len(sorted_result)):
    importances.append(sorted_result[-(i + 1)])
fi = pd.DataFrame(importances, columns=['Feature Name', 'Feature Importance'])
fi = fi.set_index("Feature Name")
fi.iloc[:25].plot(kind='barh', figsize=(6, 6)).invert_yaxis()
```

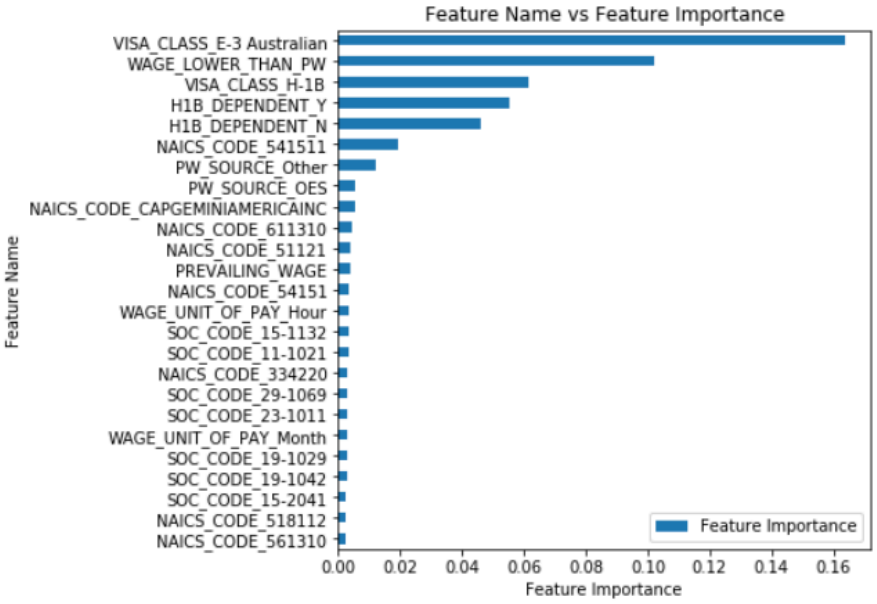

Figure 11: Top 25 Values from Feature Importance of Final Model

Although only a few values are shown on Figure 10, “NAICS Code” and “SOC Code” are top contributors to the success of the model as well. This is confirmed by checking top 60 feature importance value, in which 21 of them are “SOC Code” and 25 are “NAICS Code”.

Reflection

In summary, I created and evaluated a supervised classifier which predicts H-1B LCA case decision based on the texts entered in the application. I started by defining problem statement and decided to use F-beta score with $\beta = 0.5$ as an evaluation

metric given the nature of the problem. I explored official H-1B LCA disclosure data, identified which features are present in all years' data, and explored the data by making use of a few helpful plots to find distribution and trend of each feature. Then I listed algorithms to use as initial modeling, defined benchmark, and implemented the initial version of three models, all of which were unsuccessful due to highly imbalanced data. After balancing out samples from both classes, I evaluated results from all three models, tuned parameters for more accurate result, and picked the final model based on the evaluation score and feasibility.

The most interesting part of this project was that models with default parameters produced very good result. Finding suboptimal parameters using randomized search did increase overall score to some degree, but not to the point where the new set of parameters make the default parameters obsolete. Another interesting part was that the decision tree benchmark, which only used four features, did surprisingly well in generalizing to a pattern and predicting classes. I initially thought this model would be far less accurate than ensemble models (we're comparing a "tree" against a "forest"), but the difference was not as significant as I thought. This could be because the size of training set was not as big as I would have liked, in which case ensemble methods will definitely do a much better job preventing overfitting to training set.

The most difficult aspect of this project was finalizing the set of features to use and which technique to clean up human input data. Each Excel spreadsheet contained huge amount of human errors such as having trailing ".00" for an integer, using an invalid character (".." instead of "-" for SOC Code), not to mention a lot of missing cells. Especially since the number of samples for "Denied" label was very small, careful techniques needed to be considered and used to make the disclosure data usable as the classifier model. Also, because using more features is a tradeoff between potentially better result and noticeable increase in computing resource and time, I had to go through multiple iterations to finalize on the set of features (e.g. making use of feature importance values after initial modeling).

Improvement

There are lots of rooms for improvement including the following.

- LCA disclosure data for years 2017 and 2018 have significantly more number of categorical features that are missing from previous years (such as *CHANGE_PREVIOUS_EMPLOYMENT, CHANGE_EMPLOYER, or AMENDED_PETITION*). Since the goal of this project is to predict outcome of new applications, if the model could be generalized well with smaller volume of data, it is worth experimenting using data for years 2017 and 2018 only and see if these additional features are highly correlated to case decision.
- Given the limited resources and execution time, only a few parameters were used for parameter tuning with very sparse and limited range. Also, randomized search with a fixed number of iterations was used instead of exhaustive grid

search. Grid search with more parameters would lead to higher accuracy or F-beta score.

- Randomized search for Support Vector Classifier was done for only 7 iterations and with dimensionality reduction. Since SVC is expected to perform well for non-linear data (and given that the SVC model was on par with the best XGBoost model), it is definitely worth tuning more parameters with exhaustive search. Based on my experience, the performance bottleneck was CPU usage since scikit-learn implementation of SVC utilizes only one CPU core. To make it more feasible, I could try this on a more powerful computer (with higher clock speed than 2.7GHz and abundant L1/L2 cache) or use proprietary cloud services such as Amazon Web Services. More importantly, if a distributed cluster system could be used, I would use a library with parallelized system support, such as Spark mllib instead of scikit-learn.

References

[1] https://www.foreignlaborcert.doleta.gov/performancedata.cfm
[2] https://en.wikipedia.org/wiki/Support_vector_machine
[3] https://www.alllaw.com/articles/nolo/us-immigration/why-labor-certification-denied-employment-visas.html
[4] https://www.bridge.us/blog/top-h1b-visa-mistakes-that-employers-make
[5] https://xgboost.readthedocs.io/en/latest/build.html#building-on-osx