Machine Learning Final Project

STSCI 4740

*Jeffrey Wu, Marielle Jurist, Orvill Delatorre, Ryan Gale*

***Introduction***

Machine learning is the application of algorithms and statistical models to train a computer to progressively improve performance on a task.  Lying at the intersection of computer science, statistics, and mathematical optimization, machine learning applications include autonomous vehicles, stock trading algorithms, and speech recognition.  Supervised learning techniques are a subset of algorithms that use a known result of a computation to help “train” a statistical model to make predictions. We apply supervised learning to the google job skills dataset to determine which responsibilities and qualifications best describe the variation in possible job categories.  We then predict the job category that most closely corresponds to a given set of qualifications.

***Approach***

In order to parse out usable features from the unstructured data provided we had to use a series of natural language processing (NLP) and text matching processing in Python to clean the data  in the “Responsibilities”, “Minimum Qualifications”, and “Preferred Qualifications” columns. If we were to use the raw data as is, each observation for the three fields would be unique. It would then be impossible to train our models using supervised learning because the data is not numeric. We used a document term matrix approach for the *Responsibilities* field. Since the *Preferred Qualifications* and *Minimum Qualifications* fields have more similarities in language and key terms, we used string matching to manually parse out binary fields like “Currently Enrolled”, “Bachelors”, “Masters”, “Phd”, etc. Below we describe our steps in detail.

**I*. Punctuation***

Because the punctuation is not relevant and it does not provide significant insight in our models we eliminated it using the str.replace() function. When dealing with hyphens we replaced with a space and in contractions we just deleted the single quote and condensed the word. All other punctuation was deleted without replacement. We also trimmed any unnecessary whitespace and had to replace /n (new line characters) with a space.

**II*. Stemming***

In order to avoid counting similar words as two different tokens e.g. manage and managing, we used the Porter Stemming algorithm from the nltk package. This will stem words to their original root i.e. manag, and make identifying key features easier.

**III. *Vectorizer***

Within the Python package sklearn there is the CountVectorizer method (scikit-learn.org). The CountVectorizer method converts a collection of text documents to a matrix of token counts. In this case the “documents” are the individual “Responsibilities” entries, and the token counts are the unique words. By passing in certain parameters, the Vectorizer converts all letters to lowercase and ignores stop words. Stop words include *the*, *and*, *or*, *we* etc. that do not supply any significance to our models. We chose to token both 1, 2 and 3-grams where an n-gram is a sequence of n words. Another important parameter is the min and max frequency. For *Responsibility* we only kept the tokens that appear in at least 12 and max 727 descriptions. In total we extracted 1804 features from the 1227 descriptions. In the bag of words matrix, a 1 in indicates that the jth word or phrase appears in the listed responsibilities for the ith job posting. We added these new features by appending them to our original dataset as binary predictors.

**IV. *Text Matching***

A basic string search was used to extract minimum and preferred qualification features instead of vectorization. The objective is to identify whether an applicant matches requirements for a position. This means that certain qualifications must match in order for an applicant to be considered. Because the number of tokens in *Minimum Qualifications* and *Preferred Qualifications* is far fewer than the number of tokens identified in *Responsibilities* it is possible to simply list the tokens to be identified. The tokens for various degrees (e.g. “BA”, “BS”, “Bachelor”) were string matched and converted into binary features. Vectorization was not used in this case because a match rate is not required.

***Models***

**I. *Lasso***

The main motivation for implementing a Lasso method is that the document term matrix methods for preprocessing results in a sparse feature space with many variables. Lasso shrinks certain coefficients in the model to exactly zero, decreasing the degree of the model and thus decreasing the variance. This also leads to more interpretable models and it is easier to identify which predictors are important.

Because we are predicting multiple categories we had to implement a multinomial regression. After trying to run our model several times without success, we realized that 23 predictor categories was too many to run a multinomial regression. We then decided to group the job categories into broader buckets. The 4 buckets are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sales: 418** | **Tech: 270** | **Administrative: 247** | **Business: 292** |
| Sales Operations | Data Center Network | Administrative | Business Strategy |
| Sales and Account Management | Hardware Engineering | People Operations | Developer Relations |
| Partnerships | IT and Data Management | Program Management | Finance |
| Marketing and communications | Network Engineering | Product and Customer Support | Legal and Government Relations |
|  | Software Engineering |  | Manufacturing and Supply Chain |
|  | Technical Infrastructure |  |  |
|  | Technical Solutions |  |  |
|  | Technical Writing |  |  |
|  | UX and Design |  |  |

Even though this will increase the bias it will lead to a more parsimonious model. We found that the four bucket approach worked best for our Lasso model.

To code in R we used the glmnet library and the cv.glmnet function with alpha=1. We also used the built in cross validation function to pick the best lambda value from a grid of values. After we grabbed the best lambda value, we reran the lasso via logistic regression to classify the data into the four buckets.

In order to calculate test error we implemented 5-fold cross validation. We used a loop to run the glmnet function five different times, each with a new training set that leaves out a fifth of the data observations to set aside as the validation sets. We then got 5 different cross validated misclassification errors (using the predict function) and took the average. To calculate the misclassification error we used the table function - we took the sum of the diagonal over the total number of observations in the test set as the true positive rate. Then the misclassified observations are 1 - TPR.

**II. *Random Forest***

Because the Lasso may have over simplified the problem, we decided next to use a random forest because the model can handle a larger number of prediction classes. The reason we decided to run a Random Forest analysis is because it reduces the variance in our model in comparison to a bagged tree model. This is because the Random Forest algorithm decorrelates the different trees that are created by only choosing a split based on a random sample m of the total predictors available p. However, we should also expect that decorrelating the trees will cause our model to have a larger bias, as a result of the bias-variance tradeoff.

We started out running all 23 but then did another bucketing exercise because some of the job categories have as few as 3 instances. No matter how well our model performs, if those instances end up in the test data set (which is likely), the misclassification for these job categories are very high. We created two new groups called “Other tech” and “Other business” which included 33 and 23 observations respectively. The “Other tech” group includes the categories: Data Center and Network, IT and Data Management, Network Engineering, Technical Infrastructure, and Technical Writing. The “Other business” group includes the categories: Developer Relations and Manufacturing and Supply Chain.

To code the random forest in R we used the randomForest library and built in functions. The function call set the importance parameter to TRUE, in order to asses the importance of predictors. We set the number of trees to 50. We also passed in subset indices in order to set aside a test set and calculate cross validated misclassification error. Like our Lasso method, we implemented a 5-fold cross validation error; we looped five times to create 5 different random forests on the five different training data sets created using the k-fold approach. Again, we filled an array with five different CV errors (calculated from the tables the same way as in the lasso method) and took the average.

***Analysis***

**I. *Lasso Performance***

When running the random forest with 4 prediction categories, we got a misclassification error of 19.2%. Although this is a decent error rate, it should be noted that our model exhibits high bias because we shrank the number of categories. Hypothetically, had we been able to predict to all 23 job categories, our model would have outputted a much higher test error rate.

Important Predictors: achiev, adopt, advoc.new, analys, analyt, assign, background,

**II. *Random Forest Performance***

When running the random forest with 18 prediction categories we observed a misclassification error of 24.4%. Even though Random Forest yields a higher error than the Lasso model, it has less bias due to its higher number of response categories. We can therefore come closer to the goal of identifying the original job categories. For comparison purposes we also ran the tree based model on the 4 prediction categories, like we did in our lasso analysis, this gave us a misclassification error of 16.4%, which outperforms the Lasso (this however sacrifices lower test error and variance for greater bias). Perhaps in future implementations, we could rerun the model with different grouping methods and try to find the optimal number of prediction categories to minimize test error and finding the best balance in the bias-variance tradeoff.

**III. *Important Predictors***

Using the importance function we can get the importance values for each variable. In the 4 prediction category model the top 5 important features and their corresponding importance values for the Admin bucket are: googl (4.345122), growth (3.961638), provid (3.832381), staf (3.633662), and chang (3.421343).

For Business they are: financi (4.966718), polici (3.752126), complianc (3.624244), decis (3.317750), and risk (3.226771). For Tech they are: busi (4.314210), growth (4.224801), process (3.917872), strateg (3.835146), and campaign (3.772843). Finally, for Sales they are: polici (4.188306), design (4.172032), growth (3.952727), advertis (3.835194), and campaign (3.801661). Note, we are reporting type 1 importance which is the mean reduction in accuracy. In the 18 prediction category model, we found the top 20 most important features for all classes which were: financi (5.648554), custom, (5.388691), partner (5.354937), market (5.337132), work (5.018524), busi (5.001393), technic (4.840199), identifi (4.576206), legal (4.303590), product (4.297689), ux (4.183213), develop (4.152747), commun (4.148444), account (4.013735), present (3.959908), ensur (3.952135), content (3.944200), insight (3.928509), growth (3.869319), and design (3.855861)

**Conclusions**

It appears that our Random Forest analysis proved to be the better method for tackling our problem of identifying a specific job category given a job description. First, we can see that the test error rate for our Random Forest model is lower than that of our Lasso model. When predicting to 4 categories groups, our Random Forest model has a misclassification rate of 16.4% while our lasso model has a misclassification rate of 19.2%. Additionally, we were able to fit our Random Forest model to predict 18 category groups, which makes our model much closer to addressing our original goal of being able to predict all 23 categories. The misclassification rate for this model ended up being 24.4% which is not a large increase considering the fact that the tree model became much more complicated and target-specific.

Another reason we believe Random Forests to be the superior method for this analysis is because it is much easier to identify the important attribute and skills for a specific category, which is what we were charged with doing in the project prompt. Using the importance function, R can easily output which variables are most influential based on mean decrease in accuracy. Since our lasso model has so many predictors, it is difficult to determine which of the predictors that are included in our sparse model, which is still a lot, have the biggest influence on which category group our model will predict for a given job description. From what we found, most of the predictors that were nonzero were still relatively small i.e. absolute value less than 1.

***Appendix***

**I. text\_cleaning.ipynb**

import string

import pandas as pd

import numpy as np

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

"""

We use an n-gram vectorizer to collect significant words and phrases

from the job responsibilities section.

See scikit-learn.org/stable/modules/feature\_extraction.html and

scikit-learn.org/stable/modules/classes.html#text-feature-extraction-ref

for TfidfVectorizer parameters"""

data = pd.read\_csv('job\_skill\_short.csv', dtype = str)

# clean 15 empty lines, replace null entries with space

corpus = data['Responsibilities'].fillna(" ")

#replacing punctuation

corpus = corpus.str.replace('\'',' ')

corpus = corpus.str.replace("-",' ')

corpus = corpus.str.replace(",",'')

corpus = corpus.str.replace("\\n",'. ')

corpus = corpus.str.replace('[^\w\s]','.')

corpus = corpus.str.replace("\..",'.')

st = PorterStemmer()

corpus = corpus.apply(lambda x: " ".join([st.stem(word) for word in x.split()]))#stem words

""" tokenize all words 3 letters or longer.

Ignore 3-grams that occur in more than 60%

or in less than 1% of descriptions

"""

vectorizer = CountVectorizer(stop\_words = 'english', strip\_accents=ascii, ngram\_range=(1, 3), analyzer = 'word',max\_df=0.6, min\_df=.01)

bow = np.array(vectorizer.fit\_transform(corpus).toarray())

responsibility\_dtm = pd.DataFrame(bow, columns=vectorizer.get\_feature\_names())

# extract minimum degree qualifications

phd = []

babs = []

mstr = []

dct = []

mba = []

asc = []

for a in data['Minimum.Qualifications']:

if a is None:

a = ""

if (str(a).lower().find("phd") != -1):

phd.append(1)

else:

phd.append(0)

if (str(a).lower().find("ba/bs") != -1 or str(a).lower().find("bachelor") != -1):

babs.append(1)

else:

babs.append(0)

if (str(a).lower().find("ms") != -1 or str(a).lower().find("master") != -1):

mstr.append(1)

else:

mstr.append(0)

if (str(a).lower().find("juris") != -1):

dct.append(1)

else:

dct.append(0)

if (str(a).lower().find("mba") != -1):

mba.append(1)

else:

mba.append(0)

if (str(a).lower().find("aa") != -1 or str(a).lower().find("as") != -1):

asc.append(1)

else:

asc.append(0)

# append minimum degree qualifications to data set

minquals = {'min phd' : pd.Series(phd),

'min ba/bs' : pd.Series(babs),

'min master' : pd.Series(mstr),

'min juris' : pd.Series(dct),

'min mba' : pd.Series(mba),

'min associate' : pd.Series(asc)}

minqual\_dtm = pd.DataFrame(minquals)

# extract preferred degree qualifications

phd = []

babs = []

mstr = []

dct = []

mba = []

asc = []

for a in data['Preferred.Qualifications']:

if a is None:

a = ""

if (str(a).lower().find("phd") != -1):

phd.append(1)

else:

phd.append(0)

if (str(a).lower().find("ba/bs") != -1 or str(a).lower().find("bachelor") != -1):

babs.append(1)

else:

babs.append(0)

if (str(a).lower().find("ms") != -1 or str(a).lower().find("master") != -1):

mstr.append(1)

else:

mstr.append(0)

if (str(a).lower().find("juris") != -1):

dct.append(1)

else:

dct.append(0)

if (str(a).lower().find("mba") != -1):

mba.append(1)

else:

mba.append(0)

if (str(a).lower().find("aa") != -1 or str(a).lower().find("as") != -1):

asc.append(1)

else:

asc.append(0)

# append preferred degree qualifications to data set

prefquals = {'pref phd' : pd.Series(phd),

'pref ba/bs' : pd.Series(babs),

'pref master' : pd.Series(mstr),

'pref juris' : pd.Series(dct),

'pref mba' : pd.Series(mba),

'pref associate' : pd.Series(asc)}

prefqual\_dtm = pd.DataFrame(prefquals)

# append new features to original dataset

new\_data = pd.concat([data, responsibility\_dtm, minqual\_dtm, prefqual\_dtm], axis=1)

new\_data.to\_csv("dtm\_jobs.csv", header= True)

**II. lasso.R**

library("boot")

library("glmnet")

library("caret")

library("matrixStats")

setwd("~/Documents/Git/Job-Match-Prediction")

job\_data\_v2 = read.csv("dtm\_jobs\_v2.csv", header =TRUE)

set.seed(1)

#Randomly shuffle the data

shuffledData<-job\_data\_v2[sample(nrow(job\_data)),]

#Create 10 equally size folds

folds <- cut(seq(1,nrow(shuffledData)),breaks=5,labels=FALSE)

A = shuffledData[,12:1827]

X= as.matrix(A)

Y = as.vector(shuffledData$Cat1)

grid = 10^seq(10,-2,length = 100)

cv.error = rep(0,5)

#Perform 5 fold cross validation

for(i in 1:5){

#Segement your data by fold using the which() function

testIndexes <- which(folds==i,arr.ind=TRUE)

Xtest <- X[testIndexes, ]

Ytest <- Y[testIndexes]

Xtrain <- X[-testIndexes, ]

Ytrain <- Y[-testIndexes]

#lasso.mod = glmnet(X,Y,alpha = 1, lambda =grid, family = "multinomial")

cv.out=cv.glmnet(Xtrain,Ytrain,alpha=1,lambda=grid, family = "multinomial")

bestlam=cv.out$lambda.min

lasso.mod=glmnet(Xtrain,Ytrain,alpha=1,lambda=bestlam, family = "multinomial")

#coef(lasso.mod)

pred.lasso = inv.logit(predict.cv.glmnet(cv.out,s = bestlam, newx = Xtest, type = "link"))

pred = c(0,nrow(pred.lasso))

for(j in 1:nrow(pred.lasso)){

m= max(pred.lasso[j,,])

if(pred.lasso[j,1,] == m){pred[j]="Admin"}

else if(pred.lasso[j,2,] == m){pred[j]="Business"}

else if(pred.lasso[j,3,] == m){pred[j]="Sales"}

else{pred[j]="Tech"}

}

table= table(pred,Ytest)

print(table)

error = 1 - (sum(diag(table))/length(Ytest))

cv.error[i] = error

}

cv.error

mean\_error = mean(cv.error)

**III. tree.R**

library(tree)

library(randomForest)

setwd("~/Documents/Git/Job-Match-Prediction")

#job\_data = read.csv("dtm\_jobs.csv", header =TRUE)

job\_data\_v2 = read.csv("dtm\_jobs\_v2.csv", header =TRUE)

set.seed(1)

#Randomly shuffle the data

shuffledData<-job\_data\_v2[sample(nrow(job\_data)),]

#Create 10 equally size folds

X = shuffledData[,12:1827]

Y = as.vector(shuffledData$Cat1)

Y = as.factor(Y)

data = data.frame(Y,X)

cv.error = rep(0,5)

folds <- cut(seq(1,nrow(shuffledData)),breaks=5,labels=FALSE)

for(i in 1:5){

#Segement your data by fold using the which() function

testIndexes <- which(folds==i,arr.ind=TRUE)

train <- which(folds!=i,arr.ind=TRUE)

job.test = Y[testIndexes]

rf.job = randomForest(y=Y[train],x=X[train,], importance = TRUE, ntree=50)

rf.job

yhat.rf = predict(rf.job, newdata = X[testIndexes,])

table= table(yhat.rf,job.test)

print(table)

error = 1 - (sum(diag(table))/length(job.test))

cv.error[i] = error

}

mean(cv.error)

varUsed(rf.job,count=TRUE)