Group 304
Fake Job Posting Prediction

First Name	Last Name	Email	Student ID
Divya	Poojari	dpoojari@hawk.iit.edu	A20446668
Tanmay	Patil	tpatil3@hawk.iit.edu	A20439531

Table of Contents

1. Introduction 2

2. Data 2

3. Research Problems 3

4. Solutions 3

5. Experiments and Results 4

5.1. Methods and Process 4

5.2. Evaluations and Results 7

5.3. Findings 18

6. Conclusions and Limitations 18

6.1. Conclusions and Future work 18

6.2. Limitations 18

1. Introduction

Fake job posts are everywhere, wasting time for dozens or hundreds of aspiring applicants. A single fake job posting can lead to a deceitful crowd gaining a wealth of information. Some of them lead to nothing more than adding email addresses to the spam distribution lists and some of the worst cases involve actual thieves trying to cull sensitive information for deviant activities.

Through this project, we are trying to build classification models that can help predict such fraudulent job postings.

2. Data

We have chosen the dataset provided by Kaggle to predict Fake Job Description Prediction based on various parameters such as job_id, title, location, department,salary_range, company_profile, etc.

The Dataset has been taken from the given link below:

https://www.kaggle.com/shivamb/real-or-fake-fake-jobposting-prediction

The Employment Scam Aegean Dataset (EMSCAD) is a publicly available dataset containing 17,880 real-life job ads that aims at providing a clear picture of the Employment Scam problem to the research community and can act as a valuable testbed for scientists working in the field.

EMSCAD records were manually annotated and classified into two categories. More specifically, the dataset contains 17,014 legitimate and 866 fraudulent job ads published between 2012 to 2014.

3. Research Problems

Below are the problems we are looking forward to resolve:

1. Identification of key traits or features like words/phrases/entities of job descriptions which are fraudulent in nature.

2. Usage of text and metadata features and predict which job descriptions are real and fake.

Summary: With the help of exploratory data analysis and classification models, we would be identifying interesting insights and run a contextual embedding model from this dataset.

4. Solutions

Solution 01: We would be using two-sample hypothesis testing to prove if the proportion of fake jobs in the two dependent variables are equal or different.

Solution 02: We would be using ML algorithms like KNN algorithm, SVM Model and Logistic Regression Model to find significant variables, build classification models using the text features in the job description column to predict fake job postings and will then be compared to each other on the basis of its simulation run-time, accuracy, sensitivity and specificity.

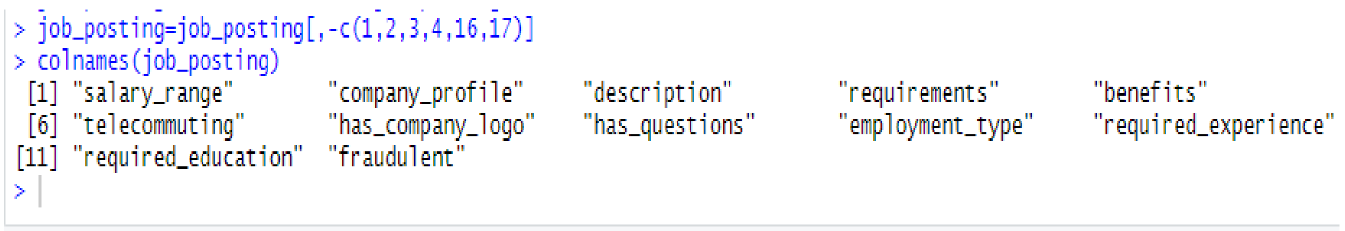
Solution 03: Various text pre-processing approaches to clean the text and search for the most frequent words and feature extraction functions are used to identify key traits of job postings which are fraudulent in nature and this analysis will set the fraudulent flag to 0(in case of genuine) and 1(in case of fraudulent) accordingly.

5. Experiments and Results

5.1 Methods and Processes

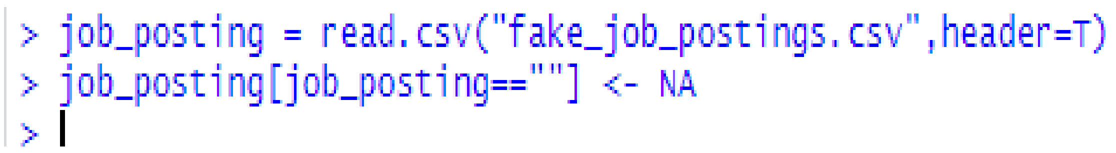
- **<u>Preprocessing of Data</u>**

  i. <u>Elimination of unused variable</u>



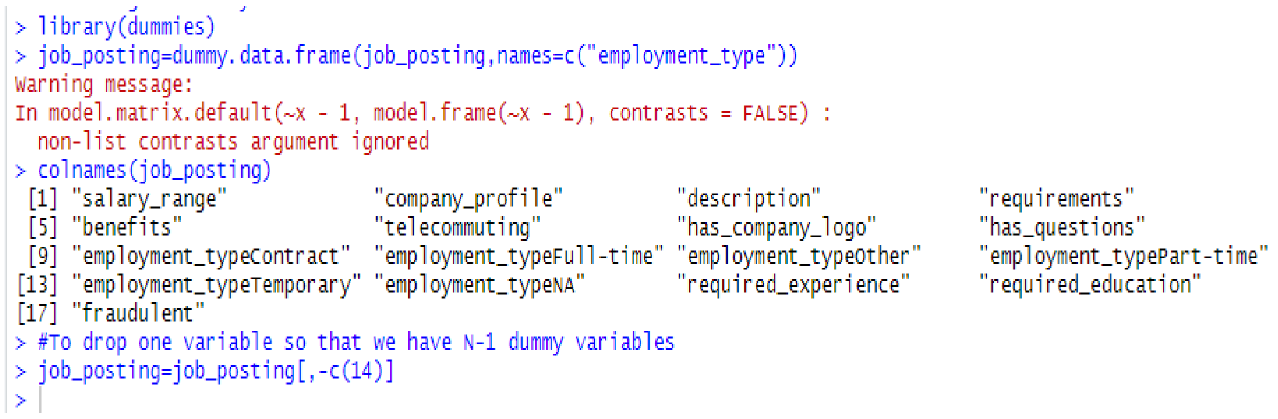
```
> job_posting=job_posting[,-c(1,2,3,4,16,17)]
> colnames(job_posting)
 [1] "salary_range"        "company_profile"     "description"         "requirements"        "benefits"
 [6] "telecommuting"       "has_company_logo"    "has_questions"       "employment_type"     "required_experience"
[11] "required_education"  "fraudulent"
>
```

  ii. <u>Replacing empty values with 'NA'</u>



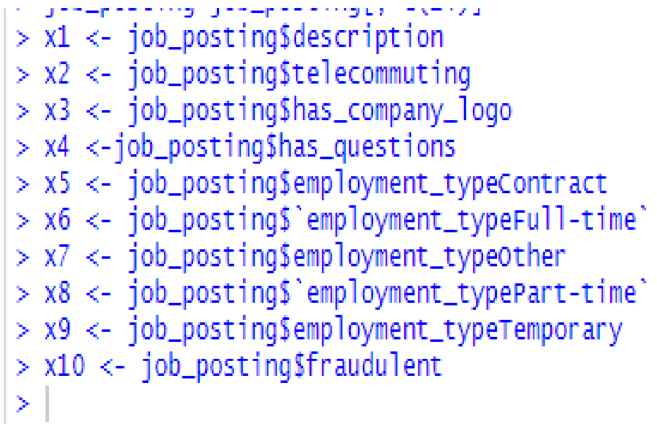
```
> job_posting = read.csv("fake_job_postings.csv",header=T)
> job_posting[job_posting==""] <- NA
>
```

  iii. <u>Converting nominal variable to binary variable and creating N-1 dummy variables</u>



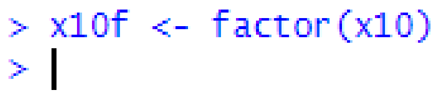
```
> library(dummies)
> job_posting=dummy.data.frame(job_posting,names=c("employment_type"))
Warning message:
In model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE) :
  non-list contrasts argument ignored
> colnames(job_posting)
 [1] "salary_range"                "company_profile"             "description"                 "requirements"
 [5] "benefits"                    "telecommuting"               "has_company_logo"            "has_questions"
 [9] "employment_typeContract"     "employment_typeFull-time"    "employment_typeOther"        "employment_typePart-time"
[13] "employment_typeTemporary"    "employment_typeNA"           "required_experience"         "required_education"
[17] "fraudulent"
> #To drop one variable so that we have N-1 dummy variables
> job_posting=job_posting[,-c(14)]
>
```

iv. Assigning variables



```
> x1 <- job_posting$description
> x2 <- job_posting$telecommuting
> x3 <- job_posting$has_company_logo
> x4 <-job_posting$has_questions
> x5 <- job_posting$employment_typeContract
> x6 <- job_posting$`employment_typeFull-time`
> x7 <- job_posting$employment_typeOther
> x8 <- job_posting$`employment_typePart-time`
> x9 <- job_posting$employment_typeTemporary
> x10 <- job_posting$fraudulent
> 
```

v. R reads labels as nominal variables. To convert it into numeric, we use the factor function.



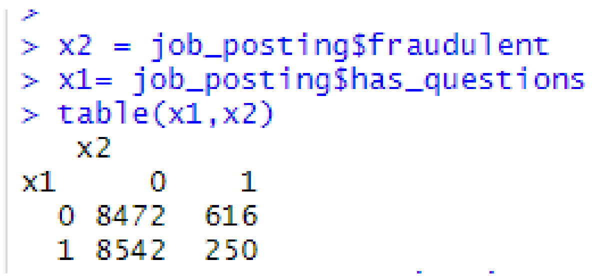
```
> x10f <- factor(x10)
> 
```

- **Two Sample Hypothesis Testing**

As per the comments, the hypothesis has been altered using correct method.

We have binary data which is useful to calculate proportions and percentages. In this case, we have considered the variable 'has_questions' and 'fraudulent'.

First we have used the table function to calculate the expected proportions of fraudulent job postings in the variable 'has_questions'.



```
> 
> x2 = job_posting$fraudulent
> x1= job_posting$has_questions
> table(x1,x2)
   x2
x1     0    1
  0 8472  616
  1 8542  250
```

From the snapshot, below are the inferences.

1. There are 8472 job postings for which ‘has_questions’=0 and are genuine.
2. There are 8542 job postings for which ‘has_questions’=1 and are genuine.
3. There are 616 job postings for which ‘has_questions’=0 and are fake.
4. There are 250 job postings for which ‘has_questions’=1 and are fake.

The total job postings for which the job postings are fake are 616+250= 866.

Proportion of expected job postings that are fraudulent= 866/17880 = 0.048

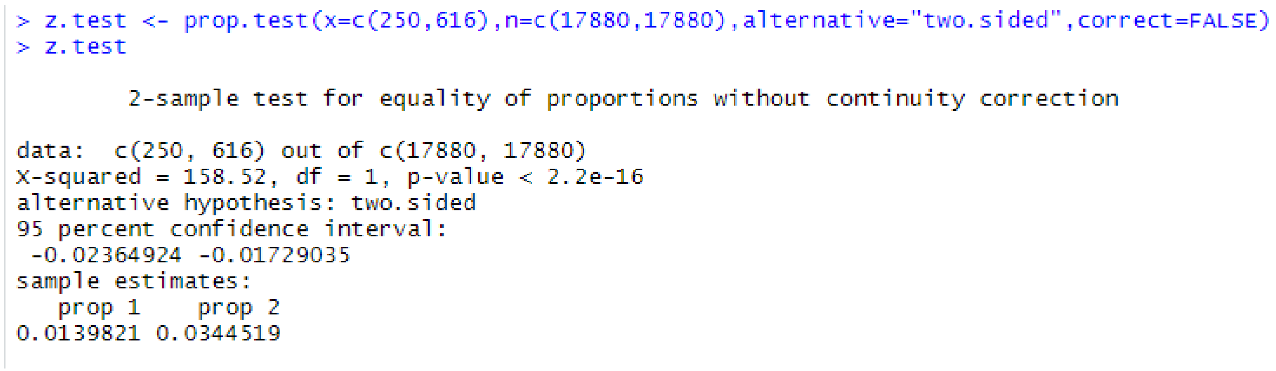
We can say that at 95% confidence interval, below are our assumptions:

Null Hypothesis H_0: The proportion of job postings with ‘has_questions’=1 and are fake = 0.048 (Have used proportion instead of mean because it is binary data)

Alternate Hypothesis H_a: The proportion of job postings with ‘has_questions’=1 and are fake is not equal to 0.048.

The line of code used for conducting z.test are as below:

z.test <- prop.test(x=c(250,616),n=c(17880,17880),alternative="two.sided",correct=FALSE)


```
> z.test <- prop.test(x=c(250,616),n=c(17880,17880),alternative="two.sided",correct=FALSE)
> z.test

	2-sample test for equality of proportions without continuity correction

data:  c(250, 616) out of c(17880, 17880)
X-squared = 158.52, df = 1, p-value < 2.2e-16
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.02364924 -0.01729035
sample estimates:
   prop 1    prop 2 
0.0139821 0.0344519 
```

We can see that the p-value < 2.2e-16.

Hence, it can be said that at a 95% confidence interval, since p-value < 0.05, we can reject the null hypothesis and the proportion of job postings with ‘has_questions’=1 and are fake is not equal to 0.048.

We have used the below classification models:

1. KNN Model

2. Logistic Regression Model

3. SVM model

For KNN and Logistic Regression model, we will be using hold-out evaluation where the dataset will be divided into training dataset(90% of data) and validation dataset(10% of data). Entire classification modelling will be done on the training dataset and will be later used on the validation dataset to predict its accuracy. But for SVM, we have used 10-fold cross validation method.

Once the best models have been selected for each method, we will compare its accuracy with each other and determine the best classification model with the highest accuracy.

5.2 Evaluation and Results

- **KNN Model**

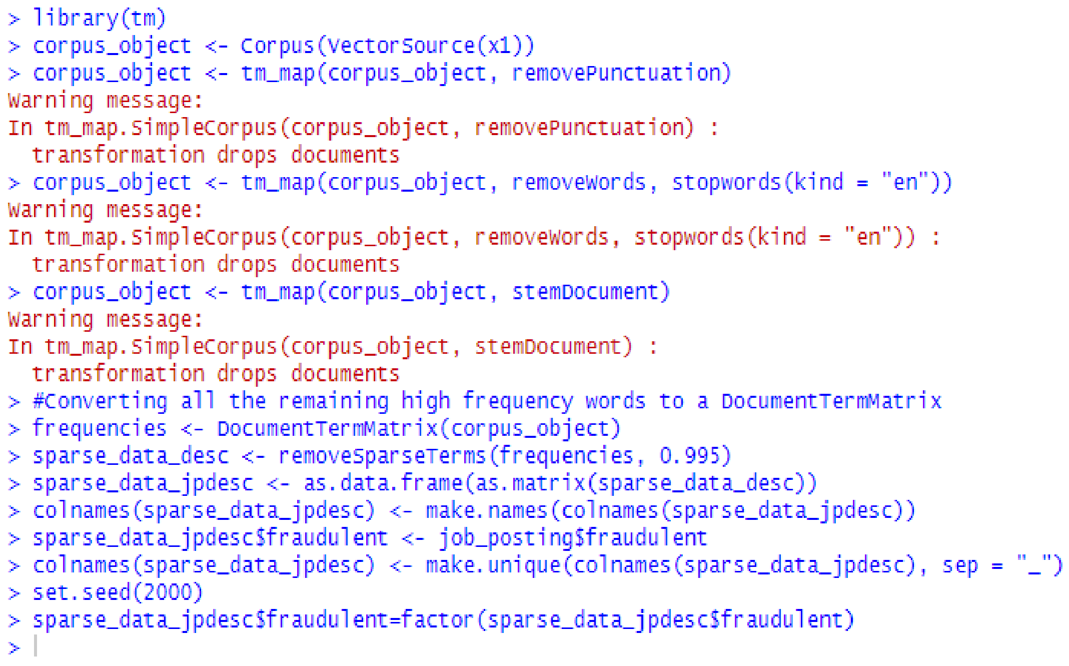
K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure i.e. distance functions.

We have used columns ‘description’ and ‘fraudulent’ for building the KNN model.

To use the ‘description’ model, we have first performed pre-processing of the column. Steps are below:

1. Creating a Corpus object.
2. Removed invalid characters.
3. Converted to Document Term Matrix
4. Removed sparse values i.e. words that occur less frequently compared to the threshold.
5. Removed duplicate columns and then created a data frame.

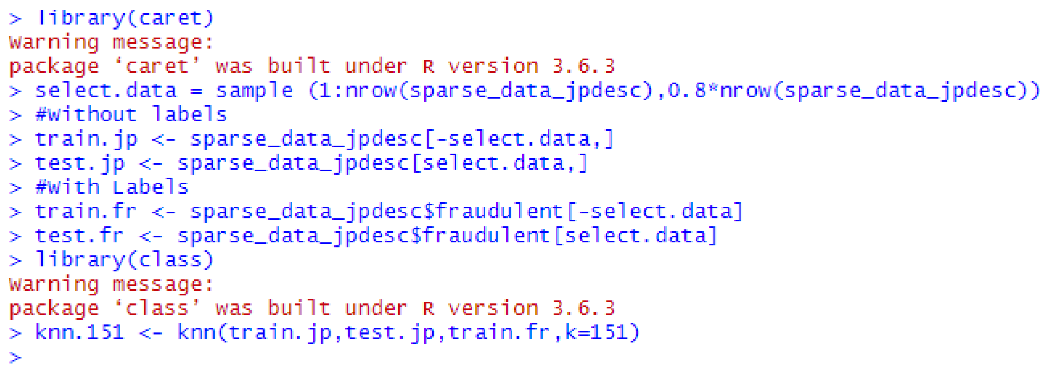
   Snapshot of the above steps are as below:



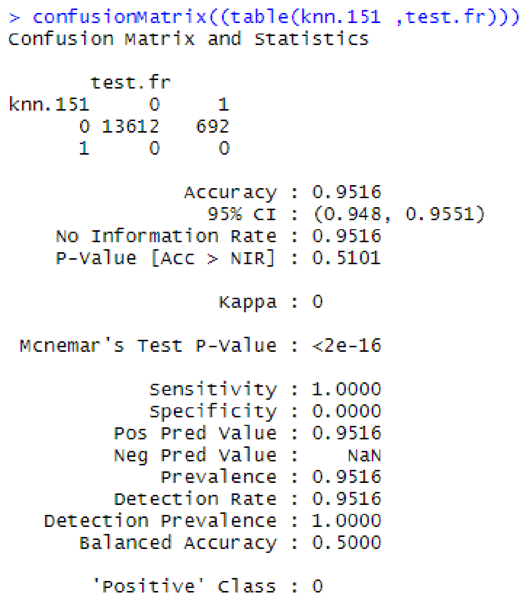
```
> library(tm)
> corpus_object <- Corpus(VectorSource(x1))
> corpus_object <- tm_map(corpus_object, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(corpus_object, removePunctuation) :
  transformation drops documents
> corpus_object <- tm_map(corpus_object, removeWords, stopwords(kind = "en"))
Warning message:
In tm_map.SimpleCorpus(corpus_object, removeWords, stopwords(kind = "en")) :
  transformation drops documents
> corpus_object <- tm_map(corpus_object, stemDocument)
Warning message:
In tm_map.SimpleCorpus(corpus_object, stemDocument) :
  transformation drops documents
> #Converting all the remaining high frequency words to a DocumentTermMatrix
> frequencies <- DocumentTermMatrix(corpus_object)
> sparse_data_desc <- removeSparseTerms(frequencies, 0.995)
> sparse_data_jpdesc <- as.data.frame(as.matrix(sparse_data_desc))
> colnames(sparse_data_jpdesc) <- make.names(colnames(sparse_data_jpdesc))
> sparse_data_jpdesc$fraudulent <- job_posting$fraudulent
> colnames(sparse_data_jpdesc) <- make.unique(colnames(sparse_data_jpdesc), sep = "_")
> set.seed(2000)
> sparse_data_jpdesc$fraudulent=factor(sparse_data_jpdesc$fraudulent)
> 
```

We then created training and validation datasets with and without labels after shuffling the data.

First, we have calculated the KNN for k=151 as well as its accuracy on the validation dataset. Snapshot of the output is as below:



```
> library(caret)
Warning message:
package ‘caret’ was built under R version 3.6.3
> select.data = sample (1:nrow(sparse_data_jpdesc),0.8*nrow(sparse_data_jpdesc))
> #without labels
> train.jp <- sparse_data_jpdesc[-select.data,]
> test.jp <- sparse_data_jpdesc[select.data,]
> #with Labels
> train.fr <- sparse_data_jpdesc$fraudulent[-select.data]
> test.fr <- sparse_data_jpdesc$fraudulent[select.data]
> library(class)
Warning message:
package ‘class’ was built under R version 3.6.3
> knn.151 <- knn(train.jp,test.jp,train.fr,k=151)
> 
```

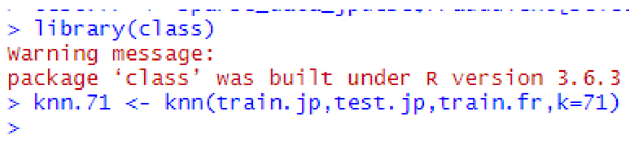


```
> confusionMatrix((table(knn.151 ,test.fr)))
Confusion Matrix and Statistics

       test.fr
knn.151     0     1
      0 13612   692
      1     0     0

               Accuracy : 0.9516
                 95% CI : (0.948, 0.9551)
    No Information Rate : 0.9516
    P-Value [Acc > NIR] : 0.5101

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9516
         Neg Pred Value :    NaN
             Prevalence : 0.9516
         Detection Rate : 0.9516
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

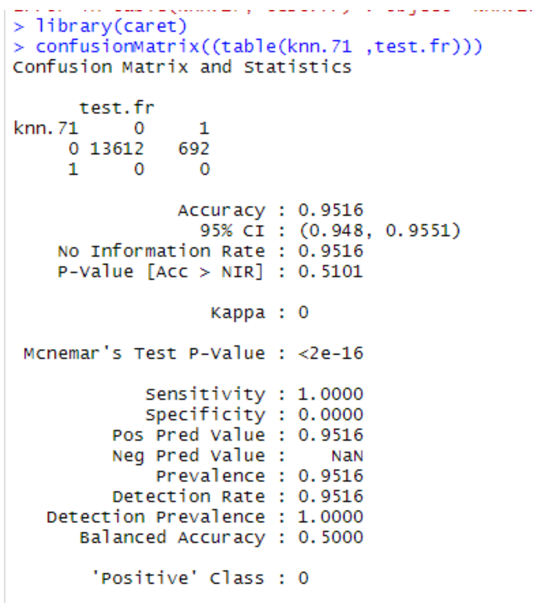
The accuracy of the KNN model for k=151 is 95.16%.

Apart from that, we have calculated KNN for 2 more values of k.

Snapshot of output for k=71



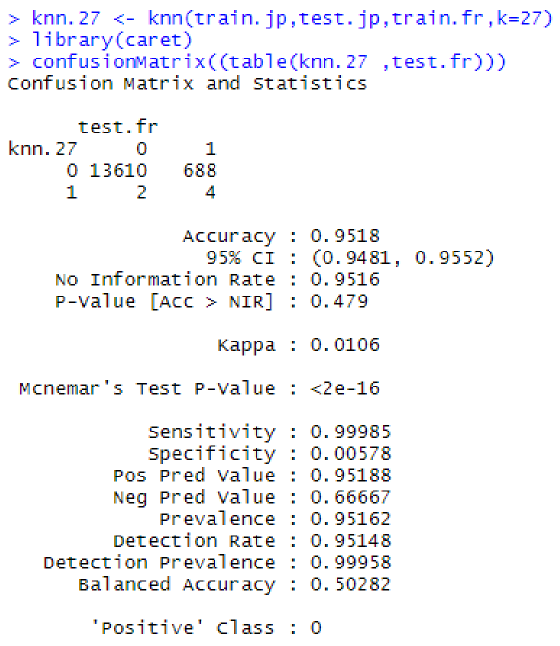
```
> library(class)
Warning message:
package ‘class’ was built under R version 3.6.3
> knn.71 <- knn(train.jp,test.jp,train.fr,k=71)
>
```



```
> library(caret)
> confusionMatrix((table(knn.71 ,test.fr)))
Confusion Matrix and Statistics

      test.fr
knn.71     0     1
     0 13612   692
     1     0     0

               Accuracy : 0.9516
                 95% CI : (0.948, 0.9551)
    No Information Rate : 0.9516
    P-Value [Acc > NIR] : 0.5101

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9516
         Neg Pred Value :    NaN
             Prevalence : 0.9516
         Detection Rate : 0.9516
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0
```

The accuracy of the KNN model for k=71 is 95.16%

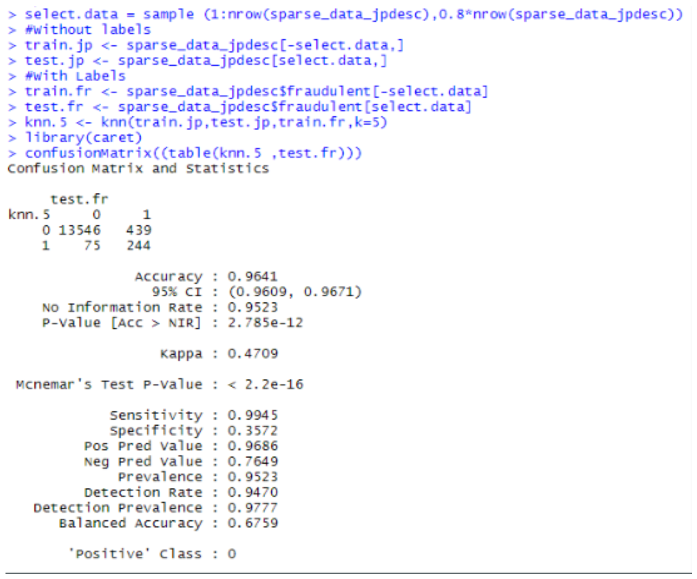
Snapshot of output for k=27



```
> knn.27 <- knn(train.jp,test.jp,train.fr,k=27)
> library(caret)
> confusionMatrix((table(knn.27 ,test.fr)))
Confusion Matrix and Statistics

      test.fr
knn.27     0     1
     0 13610   688
     1     2     4

               Accuracy : 0.9518
                 95% CI : (0.9481, 0.9552)
    No Information Rate : 0.9516
    P-Value [Acc > NIR] : 0.479

                  Kappa : 0.0106

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.99985
            Specificity : 0.00578
         Pos Pred Value : 0.95188
         Neg Pred Value : 0.66667
             Prevalence : 0.95162
         Detection Rate : 0.95148
   Detection Prevalence : 0.99958
      Balanced Accuracy : 0.50282

       'Positive' Class : 0
```

The accuracy of the KNN model for k=27 is 95.18%.

Snapshot of KNN model for k=5



```
> select.data = sample (1:nrow(sparse_data_jpdesc),0.8*nrow(sparse_data_jpdesc))
> #without labels
> train.jp <- sparse_data_jpdesc[-select.data,]
> test.jp <- sparse_data_jpdesc[select.data,]
> #with Labels
> train.fr <- sparse_data_jpdesc$fraudulent[-select.data]
> test.fr <- sparse_data_jpdesc$fraudulent[select.data]
> knn.5 <- knn(train.jp,test.jp,train.fr,k=5)
> library(caret)
> confusionMatrix((table(knn.5 ,test.fr)))
Confusion Matrix and Statistics

     test.fr
knn.5     0     1
    0 13546   439
    1    75   244

               Accuracy : 0.9641
                 95% CI : (0.9609, 0.9671)
    No Information Rate : 0.9523
    P-Value [Acc > NIR] : 2.785e-12

                  Kappa : 0.4709

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9945
            Specificity : 0.3572
         Pos Pred Value : 0.9686
         Neg Pred Value : 0.7649
             Prevalence : 0.9523
         Detection Rate : 0.9470
   Detection Prevalence : 0.9777
      Balanced Accuracy : 0.6759

       'Positive' Class : 0
```

The accuracy of KNN Model for k=5 is 96.41%

From the above results, we have achieved highest accuracy for k=5 which is 96.41%. Hence, we will proceed with this model (knn.5).

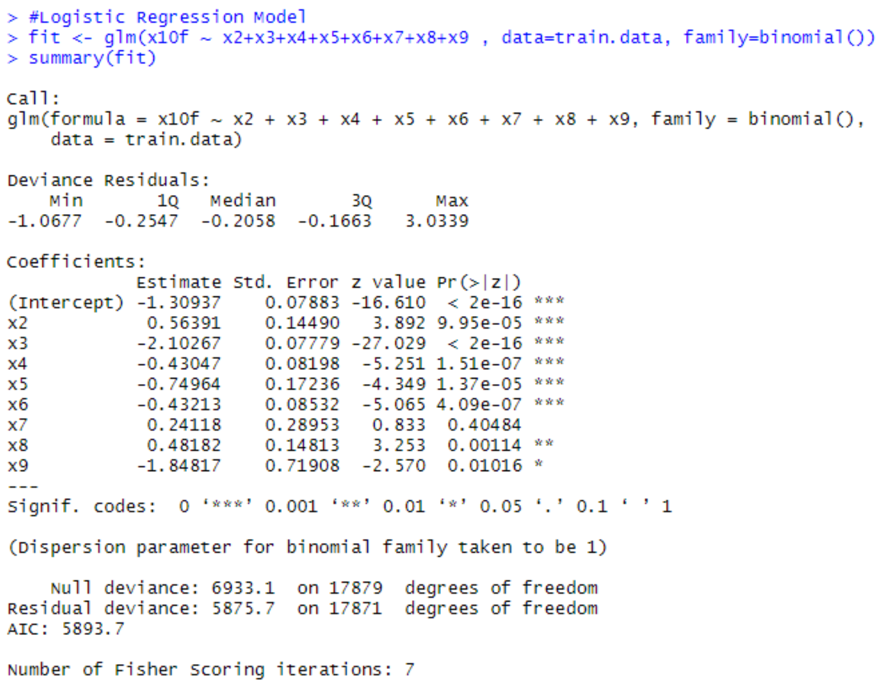
As per the comments, we have tried KNN for larger values, we got the same accuracy for all of them except for lower value k=5 and for values of k greater than 500, we got the error as 'too many equidistant points'.

Also, for the normalization part, as we have used column 'description' which is descriptive column, we have used function stemDocument which removes common words in English and acts as a normalization process also termed as Porter's stemming algorithm.

- **<u>Logistic Regression Model</u>**

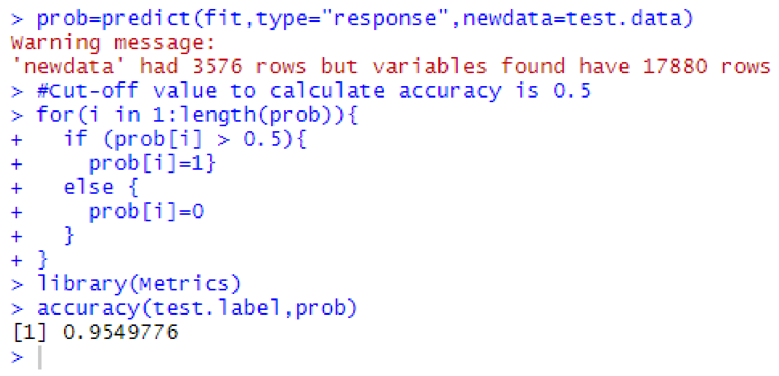
For the Logistic Regression Model, we have used the columns 'has_employment_type' , 'telecommuting', 'has_company_logo', 'has_questions'

We have already converted the nominal variable 'has_employment_type' into a binary variable with N-1 dummy variables. We have then created training and validation datasets , and created a fit model using all the variables.



```
> #Logistic Regression Model
> fit <- glm(x10f ~ x2+x3+x4+x5+x6+x7+x8+x9 , data=train.data, family=binomial())
> summary(fit)

Call:
glm(formula = x10f ~ x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9, family = binomial(),
    data = train.data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.0677  -0.2547  -0.2058  -0.1663   3.0339

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.30937    0.07883 -16.610  < 2e-16 ***
x2           0.56391    0.14490   3.892 9.95e-05 ***
x3          -2.10267    0.07779 -27.029  < 2e-16 ***
x4          -0.43047    0.08198  -5.251 1.51e-07 ***
x5          -0.74964    0.17236  -4.349 1.37e-05 ***
x6          -0.43213    0.08532  -5.065 4.09e-07 ***
x7           0.24118    0.28953   0.833  0.40484
x8           0.48182    0.14813   3.253  0.00114 **
x9          -1.84817    0.71908  -2.570  0.01016 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6933.1  on 17879  degrees of freedom
Residual deviance: 5875.7  on 17871  degrees of freedom
AIC: 5893.7

Number of Fisher Scoring iterations: 7
```

Later, we calculated accuracy of the fit model.

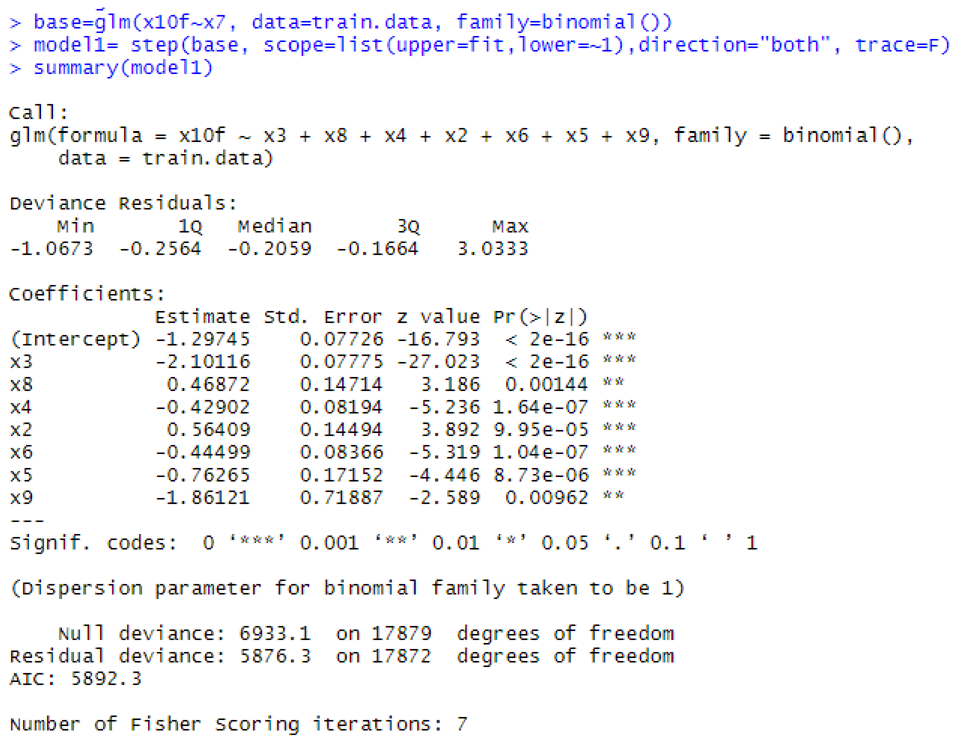


```
> prob=predict(fit,type="response",newdata=test.data)
Warning message:
'newdata' had 3576 rows but variables found have 17880 rows
> #Cut-off value to calculate accuracy is 0.5
> for(i in 1:length(prob)){
+   if (prob[i] > 0.5){
+     prob[i]=1}
+   else {
+     prob[i]=0
+   }
+ }
> library(Metrics)
> accuracy(test.label,prob)
[1] 0.9549776
> 
```

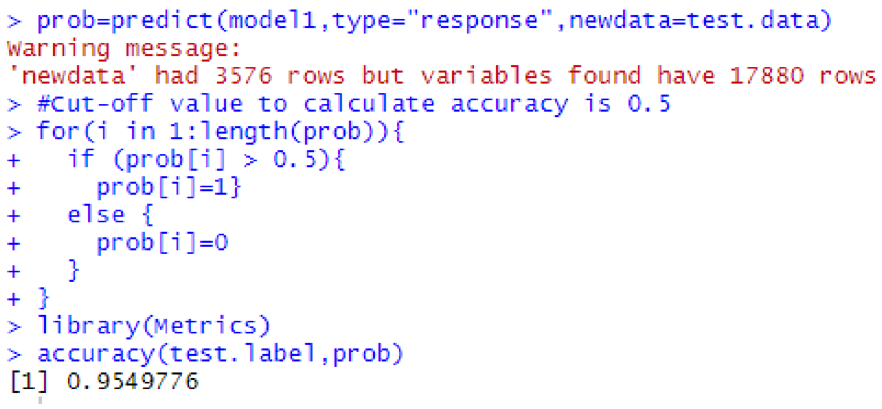
We can see from the above snapshot that **the accuracy of the fit model is 95.497%**

We have also used forward selection and backward elimination technique for creating a logistic regression model.

1. Below is the snapshot for forward selection technique with the base variable as x7.



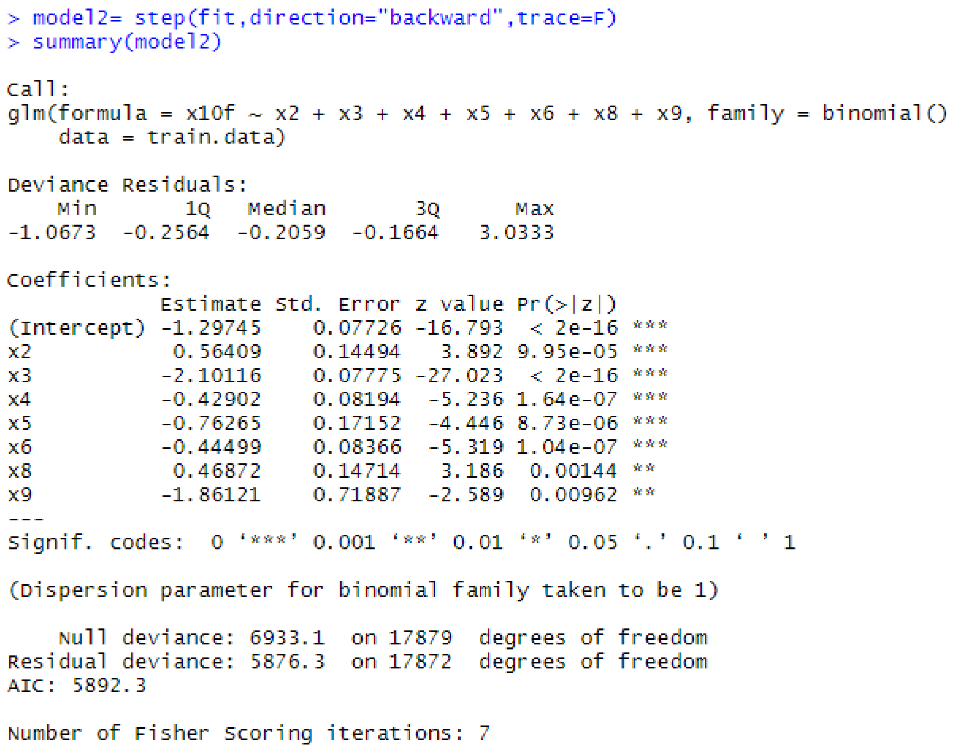
```
> base=glm(x10f~x7, data=train.data, family=binomial())
> model1= step(base, scope=list(upper=fit,lower=~1),direction="both", trace=F)
> summary(model1)

Call:
glm(formula = x10f ~ x3 + x8 + x4 + x2 + x6 + x5 + x9, family = binomial(),
    data = train.data)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.0673  -0.2564  -0.2059  -0.1664   3.0333  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.29745    0.07726 -16.793  < 2e-16 ***
x3          -2.10116    0.07775 -27.023  < 2e-16 ***
x8           0.46872    0.14714   3.186  0.00144 ** 
x4          -0.42902    0.08194  -5.236 1.64e-07 ***
x2           0.56409    0.14494   3.892 9.95e-05 ***
x6          -0.44499    0.08366  -5.319 1.04e-07 ***
x5          -0.76265    0.17152  -4.446 8.73e-06 ***
x9          -1.86121    0.71887  -2.589  0.00962 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6933.1  on 17879  degrees of freedom
Residual deviance: 5876.3  on 17872  degrees of freedom
AIC: 5892.3

Number of Fisher Scoring iterations: 7
```



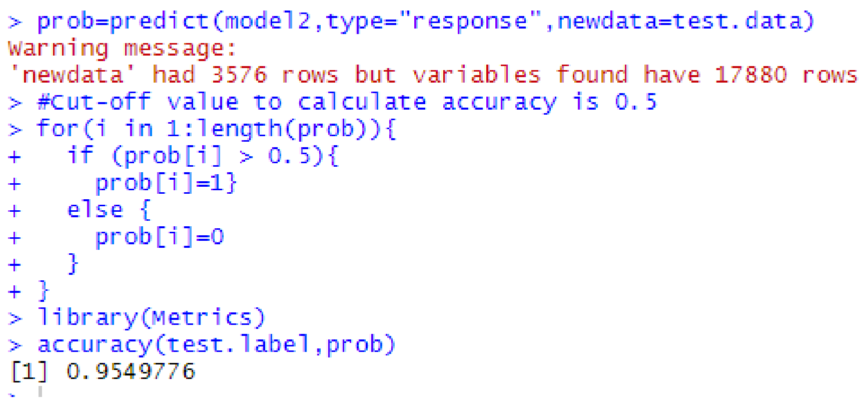
```
> prob=predict(model1,type="response",newdata=test.data)
Warning message:
'newdata' had 3576 rows but variables found have 17880 rows
> #Cut-off value to calculate accuracy is 0.5
> for(i in 1:length(prob)){
+   if (prob[i] > 0.5){
+     prob[i]=1}
+   else {
+     prob[i]=0
+   }
+ }
> library(Metrics)
> accuracy(test.label,prob)
[1] 0.9549776
```

We can see that the variable x7 has been eliminated and the **accuracy of the model using forward selection technique is 95.476%**

2. Below is the snapshot for the Backward elimination Technique and its accuracy.



```
> model2= step(fit,direction="backward",trace=F)
> summary(model2)

Call:
glm(formula = x10f ~ x2 + x3 + x4 + x5 + x6 + x8 + x9, family = binomial()
    data = train.data)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-1.0673  -0.2564  -0.2059  -0.1664   3.0333  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.29745    0.07726 -16.793  < 2e-16 ***
x2           0.56409    0.14494   3.892 9.95e-05 ***
x3          -2.10116    0.07775 -27.023  < 2e-16 ***
x4          -0.42902    0.08194  -5.236 1.64e-07 ***
x5          -0.76265    0.17152  -4.446 8.73e-06 ***
x6          -0.44499    0.08366  -5.319 1.04e-07 ***
x8           0.46872    0.14714   3.186  0.00144 ** 
x9          -1.86121    0.71887  -2.589  0.00962 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6933.1  on 17879  degrees of freedom
Residual deviance: 5876.3  on 17872  degrees of freedom
AIC: 5892.3

Number of Fisher Scoring iterations: 7
```



```
> prob=predict(model2,type="response",newdata=test.data)
Warning message:
'newdata' had 3576 rows but variables found have 17880 rows
> #Cut-off value to calculate accuracy is 0.5
> for(i in 1:length(prob)){
+   if (prob[i] > 0.5){
+     prob[i]=1}
+   else {
+     prob[i]=0
+   }
+ }
> library(Metrics)
> accuracy(test.label,prob)
[1] 0.9549776
```

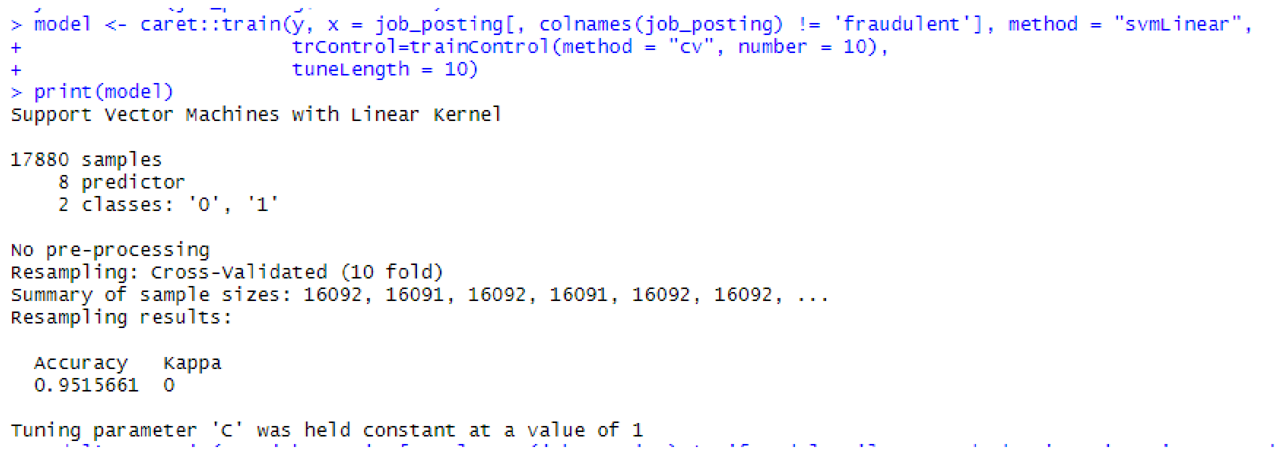
We can see that the variable x7 has been eliminated and **the accuracy of the model using Backward elimination technique is 95.497%.**

Thus, it has been demonstrated that the accuracy of all three models: fit, model1 and model2 are the same. **Hence, we can use any one of them.**

- **SVM Model**

SVM is a supervised machine learning algorithm which uses classification algorithms for two-group classification problems. Columns used for SVM Model are: ‘has_employment_type’ , ‘telecommuting’, ‘has_company_logo’, ‘has_questions’ with all the data preprocessing.

Below is the snapshot of the SVM model and its accuracy.



```
> model <- caret::train(y, x = job_posting[, colnames(job_posting) != 'fraudulent'], method = "svmLinear",
+                       trControl=trainControl(method = "cv", number = 10),
+                       tuneLength = 10)
> print(model)
Support Vector Machines with Linear Kernel

17880 samples
    8 predictor
    2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 16092, 16091, 16092, 16091, 16092, 16092, ...
Resampling results:

  Accuracy   Kappa
  0.9515661  0

Tuning parameter 'C' was held constant at a value of 1
```

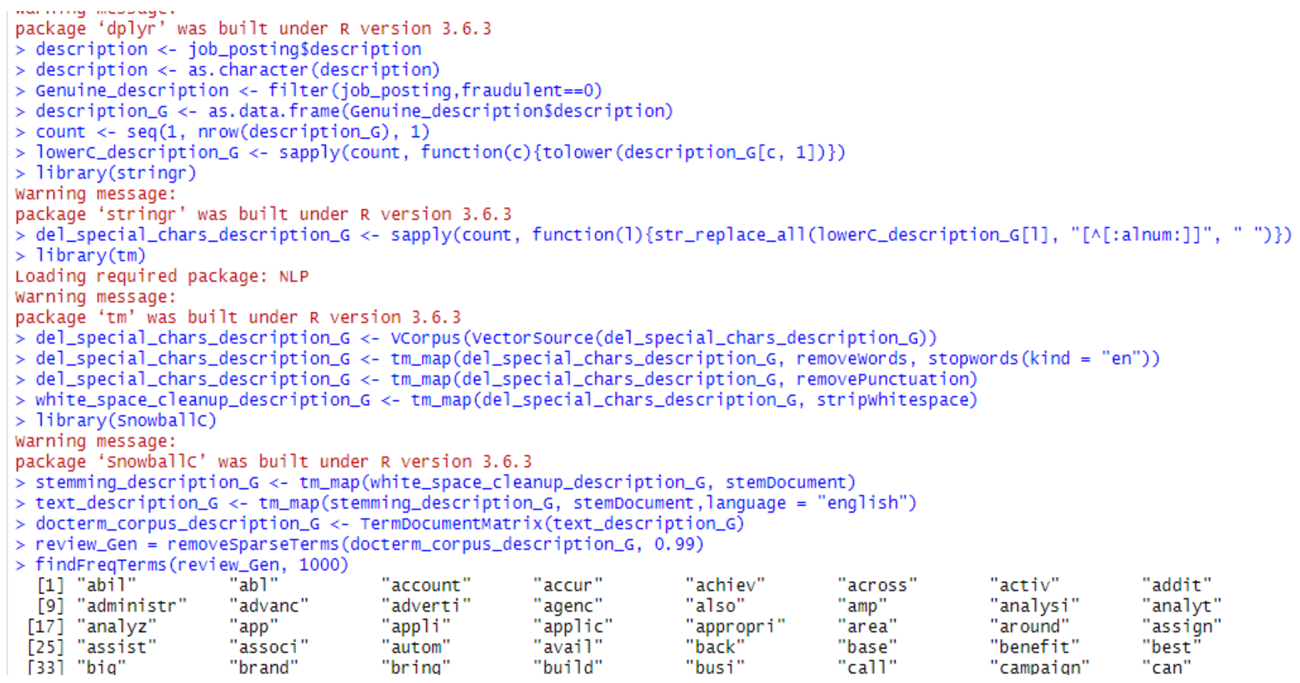
We can see that **the accuracy of the SVM model is 95.15%.**

As per the comment, we have not used pred_train and test.label together to calculate accuracy, but have used the train function instead.

- **Feature Extraction**

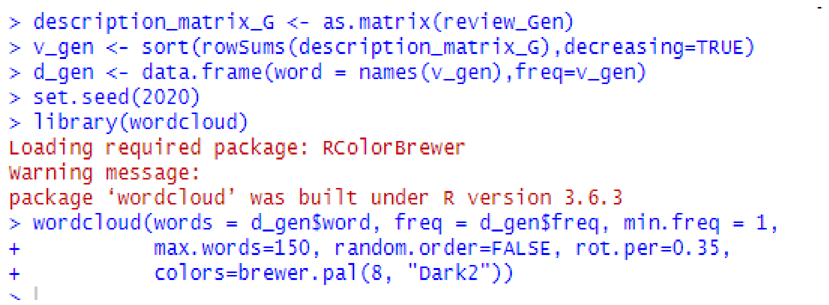
We are very well aware that the prediction whether the job posting is genuine or fraudulent mainly depends on its description. Hence, we have used column ‘description’ for some feature extraction techniques. Some of the data preprocessing that the column had to undergo are as below:

1. Converted to char
2. Converted to dataframe
3. Converted all the characters to lowercase
4. Replaced all the non-alphanumeric characters.
5. Created a corpus object from which all the punctuations, commoners, stopwords and white spaces were removed and a TermDocumentMatrix was created eliminating the sparse terms occurring below a threshold.
6. Created a matrix of words occurring with high frequency.



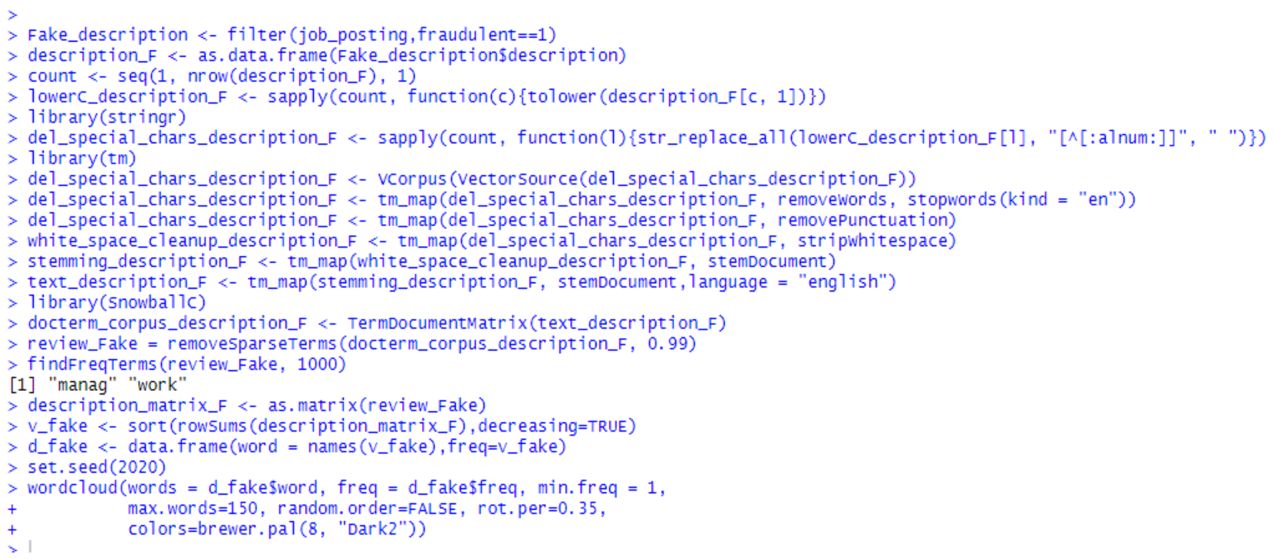
```
package ‘dplyr’ was built under R version 3.6.3
> description <- job_posting$description
> description <- as.character(description)
> Genuine_description <- filter(job_posting,fraudulent==0)
> description_G <- as.data.frame(Genuine_description$description)
> count <- seq(1, nrow(description_G), 1)
> lowerC_description_G <- sapply(count, function(c){tolower(description_G[c, 1])})
> library(stringr)
Warning message:
package ‘stringr’ was built under R version 3.6.3
> del_special_chars_description_G <- sapply(count, function(l){str_replace_all(lowerC_description_G[l], "[^[:alnum:]]", " ")})
> library(tm)
Loading required package: NLP
Warning message:
package ‘tm’ was built under R version 3.6.3
> del_special_chars_description_G <- VCorpus(VectorSource(del_special_chars_description_G))
> del_special_chars_description_G <- tm_map(del_special_chars_description_G, removeWords, stopwords(kind = "en"))
> del_special_chars_description_G <- tm_map(del_special_chars_description_G, removePunctuation)
> white_space_cleanup_description_G <- tm_map(del_special_chars_description_G, stripWhitespace)
> library(SnowballC)
Warning message:
package ‘SnowballC’ was built under R version 3.6.3
> stemming_description_G <- tm_map(white_space_cleanup_description_G, stemDocument)
> text_description_G <- tm_map(stemming_description_G, stemDocument,language = "english")
> docterm_corpus_description_G <- TermDocumentMatrix(text_description_G)
> review_Gen = removeSparseTerms(docterm_corpus_description_G, 0.99)
> findFreqTerms(review_Gen, 1000)
  [1] "abil"        "abl"         "account"     "accur"       "achiev"      "across"      "activ"       "addit"
  [9] "administr"   "advanc"      "adverti"     "agenc"       "also"        "amp"         "analysi"     "analyt"
 [17] "analyz"      "app"         "appli"       "applic"      "appropri"    "area"        "around"      "assign"
 [25] "assist"      "associ"      "autom"       "avail"       "back"        "base"        "benefit"     "best"
 [33] "big"         "brand"       "bring"       "build"       "busi"        "call"        "campaign"    "can"
```

We have then used this matrix to create a wordcloud using the words occurring with high frequency first in the genuine job postings and then in the fake job postings.



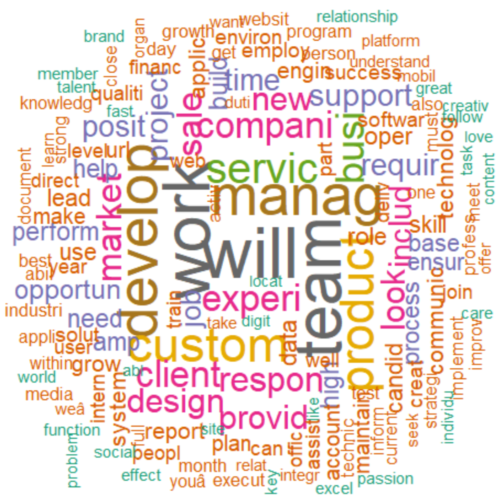
```
> description_matrix_G <- as.matrix(review_Gen)
> v_gen <- sort(rowSums(description_matrix_G),decreasing=TRUE)
> d_gen <- data.frame(word = names(v_gen),freq=v_gen)
> set.seed(2020)
> library(wordcloud)
Loading required package: RColorBrewer
Warning message:
package ‘wordcloud’ was built under R version 3.6.3
> wordcloud(words = d_gen$word, freq = d_gen$freq, min.freq = 1,
+           max.words=150, random.order=FALSE, rot.per=0.35,
+           colors=brewer.pal(8, "Dark2"))
> |
```

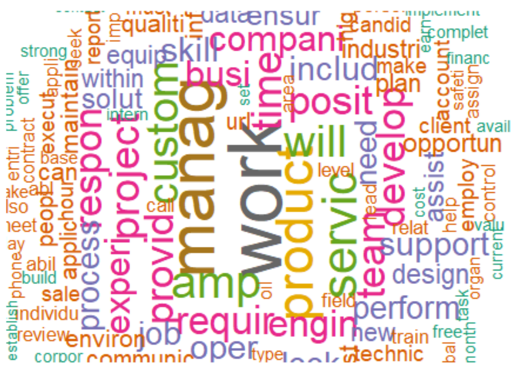
Below is the snapshot of the code executed on the fake job posting.



```
>
> Fake_description <- filter(job_posting,fraudulent==1)
> description_F <- as.data.frame(Fake_description$description)
> count <- seq(1, nrow(description_F), 1)
> lowerC_description_F <- sapply(count, function(c){tolower(description_F[c, 1])})
> library(stringr)
> del_special_chars_description_F <- sapply(count, function(l){str_replace_all(lowerC_description_F[l], "[^[:alnum:]]", " ")})
> library(tm)
> del_special_chars_description_F <- VCorpus(VectorSource(del_special_chars_description_F))
> del_special_chars_description_F <- tm_map(del_special_chars_description_F, removeWords, stopwords(kind = "en"))
> del_special_chars_description_F <- tm_map(del_special_chars_description_F, removePunctuation)
> white_space_cleanup_description_F <- tm_map(del_special_chars_description_F, stripWhitespace)
> stemming_description_F <- tm_map(white_space_cleanup_description_F, stemDocument)
> text_description_F <- tm_map(stemming_description_F, stemDocument,language = "english")
> library(SnowballC)
> docterm_corpus_description_F <- TermDocumentMatrix(text_description_F)
> review_Fake = removeSparseTerms(docterm_corpus_description_F, 0.99)
> findFreqTerms(review_Fake, 1000)
[1] "manag" "work"
> description_matrix_F <- as.matrix(review_Fake)
> v_fake <- sort(rowSums(description_matrix_F),decreasing=TRUE)
> d_fake <- data.frame(word = names(v_fake),freq=v_fake)
> set.seed(2020)
> wordcloud(words = d_fake$word, freq = d_fake$freq, min.freq = 1,
+           max.words=150, random.order=FALSE, rot.per=0.35,
+           colors=brewer.pal(8, "Dark2"))
> |
```

Below is the snapshot of the word cloud output for genuine and fake job postings.



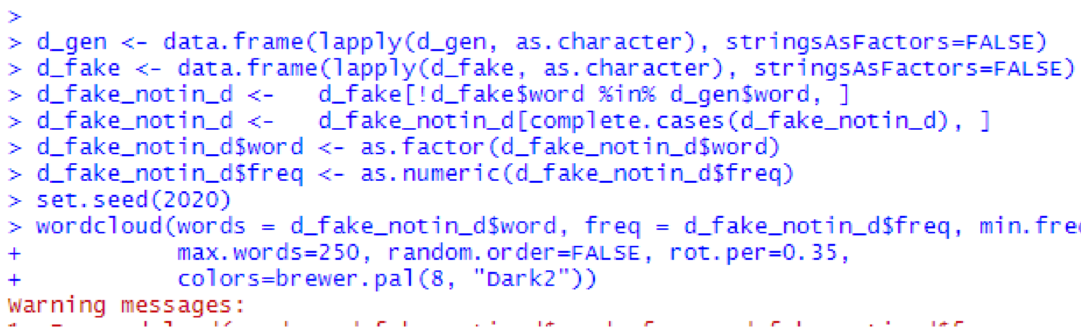


Genuine Job Posting **Fake Job Posting**

Reading the word cloud is quite easy. Larger the words, higher is its frequency. In the word cloud of the genuine job posting, the words, work, will, team, manag servic, develop have occurred in high frequency whereas the words manag, work, product, will, amp, custom have occurred in high frequency in the fake job postings. Note that some words are incomplete. This is due to the pre-processing of the data that has been performed before.

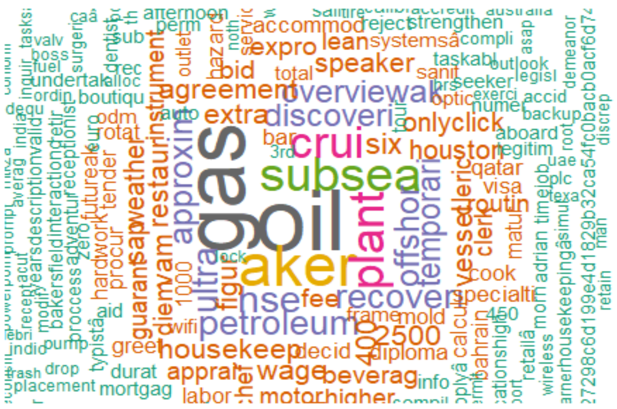
Our next step would be to find the words that are present only in the fraudulent job postings and not in the genuine job postings.

Below is the snapshot of the code used for the same purpose.



```
>
> d_gen <- data.frame(lapply(d_gen, as.character), stringsAsFactors=FALSE)
> d_fake <- data.frame(lapply(d_fake, as.character), stringsAsFactors=FALSE)
> d_fake_notin_d <-   d_fake[!d_fake$word %in% d_gen$word, ]
> d_fake_notin_d <-   d_fake_notin_d[complete.cases(d_fake_notin_d), ]
> d_fake_notin_d$word <- as.factor(d_fake_notin_d$word)
> d_fake_notin_d$freq <- as.numeric(d_fake_notin_d$freq)
> set.seed(2020)
> wordcloud(words = d_fake_notin_d$word, freq = d_fake_notin_d$freq, min.freq = 5,
+           max.words=250, random.order=FALSE, rot.per=0.35,
+           colors=brewer.pal(8, "Dark2"))
Warning messages:
```

The word cloud of the words that are present in fake job postings but not in the genuine job postings is given below:



From the above output, we can say that the words gas, oil, crui, aker, subsea, plant, offshor, temporari are present in the description of the fake job postings but not in genuine job postings.

5.3 Findings

Below are the findings from section 5.2,

a. The KNN model has the highest accuracy of 96.41%

b. The words gas, oil, crui, aker, subsea, plant, offshor, temporari are present in the description of the fake job postings but not in genuine job posting.

6. Conclusion and Limitations

6.1 Conclusion and Future Work

1. We have created classification models using 3 modelling algorithms: KNN model, Logistic Regression Model and SVM Model with KNN model having the highest accuracy of 96.41%.

2. We have also performed feature extraction using the Term Document Matrix method.

Future work includes implementation of more models which could lead to better comparison of all factors along with accuracy i.e. its RMSE value, specificity, run-time, and sensitivity.

6.2 Limitations

In this project, we were not able to use all the columns due to the type of the data present in it.

In case we had a more precise dataset, we could be using the same dimensions (set of columns) for all the models implemented and there definitely would have been an improvement in the modelling and comparison of accuracy levels.