Predicting Caravan Potential Buyers based on Historical Consumer Data

Jigyasa Sachdeva

December 5, 2020

Jigyasa Sachdeva
University of Illinois at Chicago
Masters in Management Information Systems
Information and Decision Sciences Department
jsachd3@uic.edu

Abstract

Using existing data to classify unseen data into different categories is a classical supervised machine learning problem. I have used machine learning recipe for generalized linear models, regularization techniques and hyper parameter tuning to build Logistic Regression, Ridge Regression, K- Nearest Neighbors, Support Vector Machine and Random Forest models. The most optimized parameters of each model are respectively fitted on the test data to compare their evaluation metrics and identify the most suitable approach for predicting Caravan's insurance policy buyers.

1. Introduction

> There is only one winning strategy. It is to carefully define the target market and direct a superior offering to that target market.
> *Philip Kotler*

Target marketing is a very important scheme for businesses these days. Understanding characteristics of potential buyers, personalizing advertisements and managing customer relations is possible only when there is a smaller pool of customers. Caravan Insurance Policy sellers want to predict a pool of potential customers based on historical data in order to target them with better promotional strategies. This will be possible only when these potential buyers are identified correctly. This paper discusses a road map to tackle the data and the applications of supervised learning methodologies that help us meet Caravan's expectations. The scope of this project can be extended to other firms wishing to understand their customer pool.

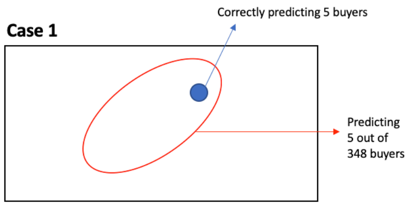
Marketing is a process by virtue of which a product's existence is spread to the general public. This process requires understanding potential customer's needs and wants from a product but may not always be convincing enough to convert their thought into a purchase. The buying cycle has three phases: awareness, consideration and purchase. At every phase, the proportion of people reduces drastically. There is a likelihood that only 5% of the people you target, may end up buying your product. Hence, this introduces imbalance in our customer data.

In order to deal with imbalance data, methodologies of undersampling, replicating oversampling, Synthetic minority oversampling and regularization methods were considered. For every model, different techniques worked out.

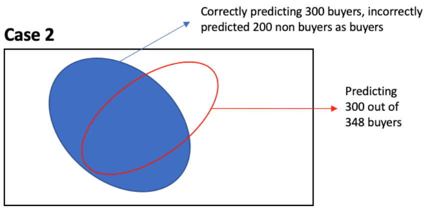
Considering the imbalance of classes, the following two statistics can be rendered important:

- Recall: This metric can be defined as the proportion of consumers correctly identified as insurance buyers out of total actual buyers.
- Precision: This metric can be defined as the correctness of our insurance buyers' prediction.

It is important to understand that recall is much more important than precision in this scenario. For example, Consider Case 1: out of a pool of 348 actual buyers, predicting 5 correct customers would yield us a 100% precision but a 1.44% recall. Alternatively, Case 2 depicts that extra prediction of customers isn't as harmful as we manage to target 86.2% of the customers.


Case 1

Precision = 100%, Recall = 1.44%


Case 2
Correctly predicting 300 buyers, incorrectly predicted 200 non buyers as buyers
Predicting 300 out of 348 buyers

Precision = 60%, Recall = 86.2%

Additionally, we cannot completely neglect precision because a model that maximizes only recall would predict all the customers as potential buyers, which is the opposite of target marketing. Since, F1 score is the harmonic mean between precision and recall and hence weighs them equally, I rather preferred choosing F-score as my evaluation criteria.

$$F_\beta = (1 + \beta^2) \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{(\beta^2 \cdot \mathrm{precision}) + \mathrm{recall}}.$$

I chose beta = 2, which implies that recall is two times more important than precision.[2]
Keeping in mind the above metric, all the models were respectively tuned on the training data to obtain the best parameters. A comparison of F scores of the final models on the test data will give us a better comparison of our prediction.

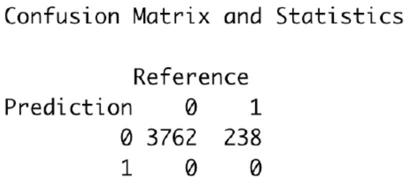
2. Related Work

Sampling techniques for balancing data have been thoroughly studied by statisticians.[4] Under sampling leads to data loss.[5] Hence, SMOTE was used for oversampling in the project (wherever necessary). For logistic regression modeling, lasso and ridge regression perform well with imbalanced data.[6] Additionally, Support Vector Machine algorithm have been used in past over SMOTE data[3] and has been proved to function well. This was attempted in the project and the results obtained were opposite. This can be explained by the fact that the newly added observations would not be support vectors and hence, not impact classification of unseen data. Scikit-learn documentations were very helpful to implement the learning in Python.[7] Similarly, medium articles depicting hyper parameter tuning have been very valuable to the code for optimizing results.[8]

3. Model

3.1. Mathematical modeling

Given the problem statement, a naïve model would predict majority class classification as follows:

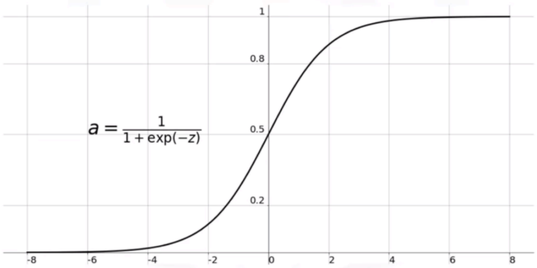


```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 3762  238
         1    0    0
```

Since majority class classification is the most undesirable situation for the specific problem, a basic logistic regression model was used as the baseline. Further models were built and compared with the baseline to improve the results.

3.1.1. Baseline Model

A logistic regression model is a part of the generalized linear model family. It can be used for classification by the use of the sigmoid function. The model provides probability in the range of [0,1]. This can be seen by the function seen below:



This function has been derived from the Machine Learning Recipe for Generalized Linear Model. The sufficient statistic based on exponential family function for logistic regression is log odds ratio. Since η is linearly related between parameters and observation, hence the following function can be established:

$$\text{Logistic regression: } \log\frac{\phi}{1-\phi} = \eta = \theta^T x \rightarrow \phi = \frac{1}{1+e^{-\eta}} = \frac{1}{1+e^{-\theta^T X}}$$

Predicted values is calculated as below:

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

3.1.2. Random Forest

Deviation between predicted and actual values can be calculated as the sum of bias, variance and error term:

$$\begin{aligned}\text{Error}(x) &= E\left(Y - \hat{f}(x)\right)^2 \\ &= \underbrace{\left(E\left(\hat{f}(x)\right) - f(x)\right)^2}_{\text{bias}^2} + \underbrace{E\left(\hat{f}(x) - E\left(\hat{f}(x)\right)\right)^2}_{\text{variance}} + \underbrace{\sigma_e^2}_{\text{error}}\end{aligned}$$

Decision trees are machine learning models which consider gini or information gain to identify the best split per node. After an optimal number of splits, decision trees classify observations into different categories. The final model is a set of if-then rules. Since, decision trees have a very low bias, it suffers from high variance. To reduce the chances of overfit due to high variance; random forest ensembles multiple decision trees and predicts majority classification based on important variables.

3.1.3. K- Nearest Neighbor

Proximity between two observations can be calculated mathematically as the inverse of distance between them. The metrics used for distance may be Manhattan or Euclidean. On the basis of 'k' nearest neighbors, the unseen data is classified on the basis of majority vote as follows:

$$\text{Predict } h(x') = arg\,\underset{y \in Y}{max} \left\{ \sum\nolimits_{i \in kNN(x')} 1_{[y^{(i)}=y]} \right\}$$

3.1.4. Support Vector Machine

Support Vector Machine attempts to solve constrained optimization problem by using Lagrange multiplier. In the process of simplification of loss function, kernel trick gets handy for computational purposes. The model can be regularized by softening the margin using cost parameter as follows:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{subject to } \; y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i, \;\; \xi_i \geq 0 \; \forall i$$

'C' in the above equation provides room for margin bending. It also acts as a balance between optimal weights and slack parameter. The regularization makes this model robust to complex classification problems.

3.1.5. Regularized logistic regression

Regularizers in logistic regression works wonders as much as in Support Vector Machine. Support Vector Machines and Logistic Regression are similar in this very way. The following equation explains how 'C' balances the trade-off between a high weight of parameters and loss function. This trade-off prevents overfitting and proves well for classification.

Regularized logistic regression

$$\min_{\theta} \; C\|\theta\|_2^2 - \sum_{i=1}^{m} \log\left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}$$

The following choice of penalization are the basis of three different regularization models: lasso, ridge and elastic net:

$$\|\theta\|_2^2 \text{ (Ridge regression)}, \|\theta\|_1 \text{ (Lasso regression)}, \; C_1\|\theta\|_2^2 + C_2\|\theta\|_1 \text{ (Elastic net)}$$

3.2. Pseudo code with algorithm

3.2.1. Grid Search Cross Validation Pseudo Code:

```
number of cross validations = c
for all hyper parameters:
    make all possible combination of each hyper parameter
    // For 3 parameters with n, k, l possibilities: total number of combinations = n x k x l
    for $i^{th}$ combination of hyper- parameters:
        iterate c times:
            test data = $c^{th}$ fold
            train data = all other folds
            run the model on train data with $i^{th}$ combination of hyper- parameters
            calculate f score on validation data
            c = c-1 // repeat until c is 1
        for c iterations: average f score
    choose best average f score
choose respective hyper parameters set as best hyper parameter set
fit the model with selected hyper parameters on train data
predict the model on test data
calculate f score
```

3.2.2. Random Search Cross Validation Pseudo Code:

```
number of cross validations = c
for all hyper parameters:
    select few random combinations of each hyper parameter //out of ‘n x k x l’ possibilities
    for $i^{th}$ combination of hyper- parameters:
        iterate c times:
            test data = cth fold
            train data = all other folds
            run the model on train data
            calculate f score on validation data
            c = c-1 // repeat until c=1
        for c iterations: average f score
    choose best average f score
choose respective hyper parameters set as best hyper parameter set
fit the model with selected hyper parameters on train data
predict the model on test data
calculate f score
```

3.2.3. k fold cross validation for tuning 1 hyper parameter:

```
for range of values of 1 hyperparameter (length = l):
    for k folds:
        validation data = kth fold
        train data = all other folds
            run the model with hyper parameter
            calculate f score on validation data
        k = k-1 // repeat until k=1
        average f score for given hyperparameter
choose highest average f score
choose respective hyper parameters set as best hyper parameter set
fit the model with selected hyper parameters on train data
predict the model on test data
calculate f score
```

3.2.4. Hold out cross validation

```
for range of values of 1 hyperparameter (length = l):
    random 70-30 split of train data into train and validation:
        fit model on train data using hyper parameter
        predict on validation data
        calculate f score on validation data
    choose best f score
choose respective best hyper parameter
fit the model with selected hyper parameter on train data
predict the model on test data
calculate f score
```

3.3 Code Snippets

3.3.1. K Nearest Neighbor

Hold out cross validation for tuning:

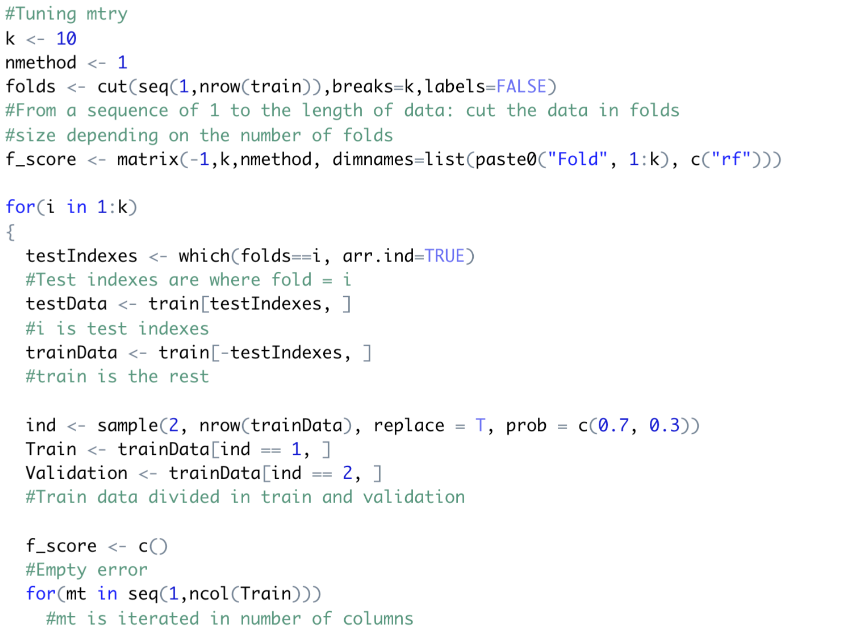
- k (Number of neighbors)

```python
#Optimal number of neighbors = k
optimal_k = 0
#Optimal score
optimal_score = 0
#Number of neighbors
sequence =  list(range(1, 39, 2))

#Optimizing F1
for k in sequence:
    #Fitting the model
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train, y_train)
    #Predicting the model on validation data
    knn_y_pred = knn_model.predict(X_val)

    #F1 score
    f1 = metrics.f1_score(y_val, knn_y_pred)
    #Checking if f1 score is greater than previous optimal score, updating k
    if f1 > optimal_score:
        optimal_k = k
        optimal_score = f1

knn_model = KNeighborsClassifier(n_neighbors=optimal_k)
knn_model.fit(X_train, y_train)
best_knn_pred = knn_model.predict(X_val)
print(confusion_matrix(y_val, best_knn_pred))
```

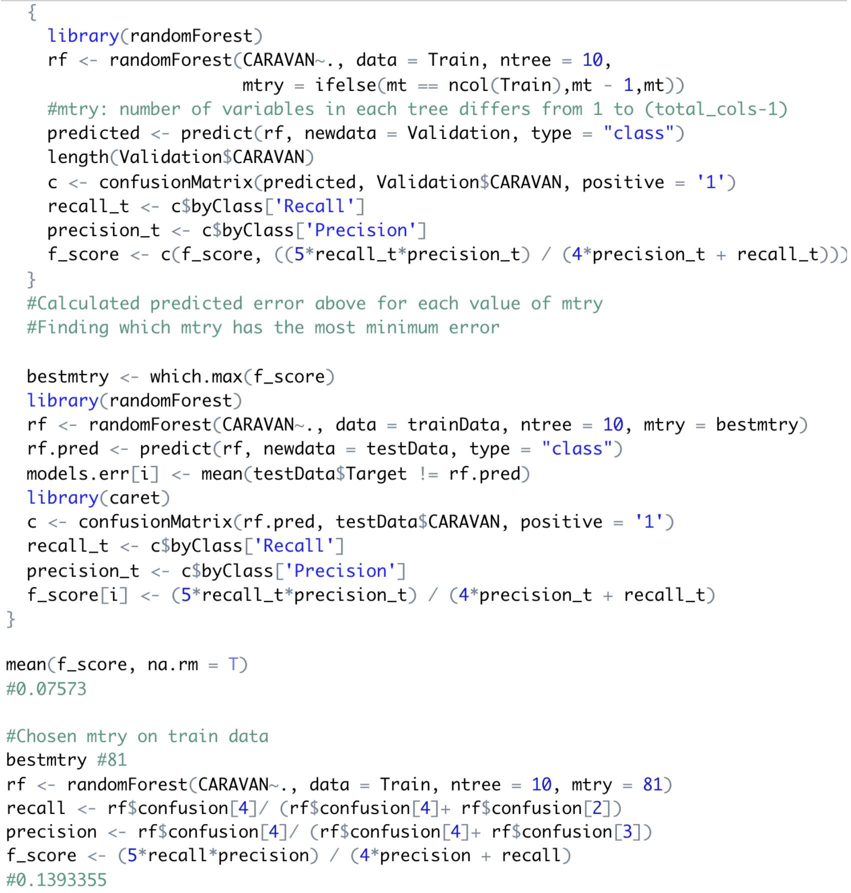
3.3.2. Random Forest

10-fold cross validation for tuning:

- mtry (number of features):



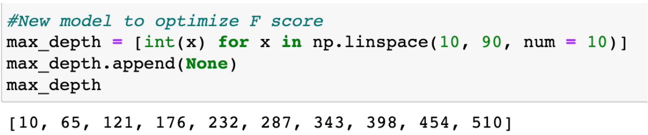
```
#Tuning mtry
k <- 10
nmethod <- 1
folds <- cut(seq(1,nrow(train)),breaks=k,labels=FALSE)
#From a sequence of 1 to the length of data: cut the data in folds
#size depending on the number of folds
f_score <- matrix(-1,k,nmethod, dimnames=list(paste0("Fold", 1:k), c("rf")))

for(i in 1:k)
{
  testIndexes <- which(folds==i, arr.ind=TRUE)
  #Test indexes are where fold = i
  testData <- train[testIndexes, ]
  #i is test indexes
  trainData <- train[-testIndexes, ]
  #train is the rest

  ind <- sample(2, nrow(trainData), replace = T, prob = c(0.7, 0.3))
  Train <- trainData[ind == 1, ]
  Validation <- trainData[ind == 2, ]
  #Train data divided in train and validation

  f_score <- c()
  #Empty error
  for(mt in seq(1,ncol(Train)))
    #mt is iterated in number of columns
```



```
  {
    library(randomForest)
    rf <- randomForest(CARAVAN~., data = Train, ntree = 10,
                       mtry = ifelse(mt == ncol(Train),mt - 1,mt))
    #mtry: number of variables in each tree differs from 1 to (total_cols-1)
    predicted <- predict(rf, newdata = Validation, type = "class")
    length(Validation$CARAVAN)
    c <- confusionMatrix(predicted, Validation$CARAVAN, positive = '1')
    recall_t <- c$byClass['Recall']
    precision_t <- c$byClass['Precision']
    f_score <- c(f_score, ((5*recall_t*precision_t) / (4*precision_t + recall_t)))
  }
  #Calculated predicted error above for each value of mtry
  #Finding which mtry has the most minimum error

  bestmtry <- which.max(f_score)
  library(randomForest)
  rf <- randomForest(CARAVAN~., data = trainData, ntree = 10, mtry = bestmtry)
  rf.pred <- predict(rf, newdata = testData, type = "class")
  models.err[i] <- mean(testData$Target != rf.pred)
  library(caret)
  c <- confusionMatrix(rf.pred, testData$CARAVAN, positive = '1')
  recall_t <- c$byClass['Recall']
  precision_t <- c$byClass['Precision']
  f_score[i] <- (5*recall_t*precision_t) / (4*precision_t + recall_t)
}

mean(f_score, na.rm = T)
#0.07573

#Chosen mtry on train data
bestmtry #81
rf <- randomForest(CARAVAN~., data = Train, ntree = 10, mtry = 81)
recall <- rf$confusion[4]/ (rf$confusion[4]+ rf$confusion[2])
precision <- rf$confusion[4]/ (rf$confusion[4]+ rf$confusion[3])
f_score <- (5*recall*precision) / (4*precision + recall)
#0.1393355
```

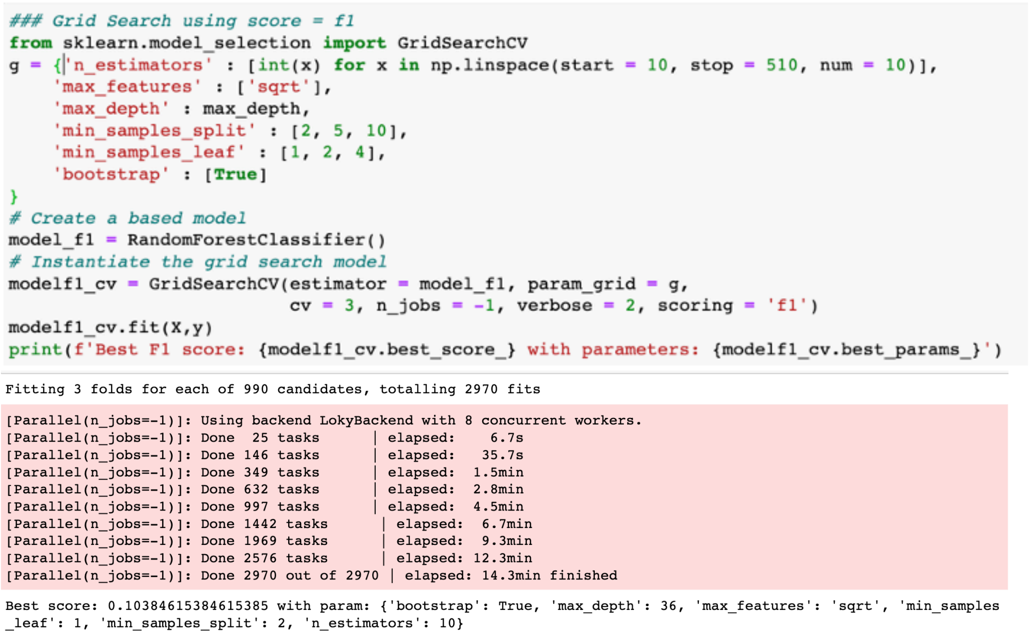
Grid search cross validation for tuning:

- n_estimators (number of trees),
- max_depth (maximum depth of tree),
- min_samples_split (minimum observations required to split),
- min_samples_leaf (minimum observations required in the leaf node)



```
#New model to optimize F score
max_depth = [int(x) for x in np.linspace(10, 90, num = 10)]
max_depth.append(None)
max_depth
```

```
[10, 65, 121, 176, 232, 287, 343, 398, 454, 510]
```



```
### Grid Search using score = f1
from sklearn.model_selection import GridSearchCV
g = {'n_estimators' : [int(x) for x in np.linspace(start = 10, stop = 510, num = 10)],
     'max_features' : ['sqrt'],
     'max_depth' : max_depth,
     'min_samples_split' : [2, 5, 10],
     'min_samples_leaf' : [1, 2, 4],
     'bootstrap' : [True]
}
# Create a based model
model_f1 = RandomForestClassifier()
# Instantiate the grid search model
modelf1_cv = GridSearchCV(estimator = model_f1, param_grid = g,
                          cv = 3, n_jobs = -1, verbose = 2, scoring = 'f1')
modelf1_cv.fit(X,y)
print(f'Best F1 score: {modelf1_cv.best_score_} with parameters: {modelf1_cv.best_params_}')
```

```
Fitting 3 folds for each of 990 candidates, totalling 2970 fits
```

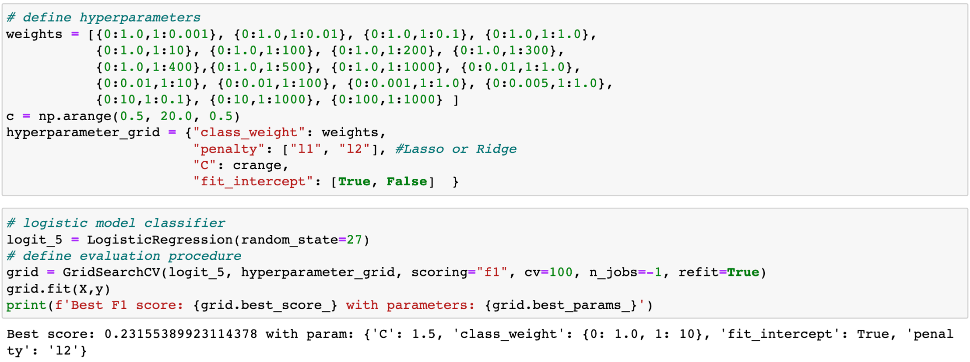
```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    6.7s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   35.7s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 997 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done 1442 tasks      | elapsed:  6.7min
[Parallel(n_jobs=-1)]: Done 1969 tasks      | elapsed:  9.3min
[Parallel(n_jobs=-1)]: Done 2576 tasks      | elapsed: 12.3min
[Parallel(n_jobs=-1)]: Done 2970 out of 2970 | elapsed: 14.3min finished
```

```
Best score: 0.10384615384615385 with param: {'bootstrap': True, 'max_depth': 36, 'max_features': 'sqrt', 'min_samples
_leaf': 1, 'min_samples_split': 2, 'n_estimators': 10}
```

3.3.3. Logistic Regression

Grid search cross validation for tuning:

- class_weight (from a range of values)
- C (cost = inverse of regularization strength)
- penalty (lasso or ridge)
- fit_intercept (true or false depending on better fit)



```
# define hyperparameters
weights = [{0:1.0,1:0.001}, {0:1.0,1:0.01}, {0:1.0,1:0.1}, {0:1.0,1:1.0},
           {0:1.0,1:10}, {0:1.0,1:100}, {0:1.0,1:200}, {0:1.0,1:300},
           {0:1.0,1:400},{0:1.0,1:500}, {0:1.0,1:1000}, {0:0.01,1:1.0},
           {0:0.01,1:10}, {0:0.01,1:100}, {0:0.001,1:1.0}, {0:0.005,1:1.0},
           {0:10,1:0.1}, {0:10,1:1000}, {0:100,1:1000} ]
c = np.arange(0.5, 20.0, 0.5)
hyperparameter_grid = {"class_weight": weights,
                       "penalty": ["l1", "l2"], #Lasso or Ridge
                       "C": crange,
                       "fit_intercept": [True, False]  }
```

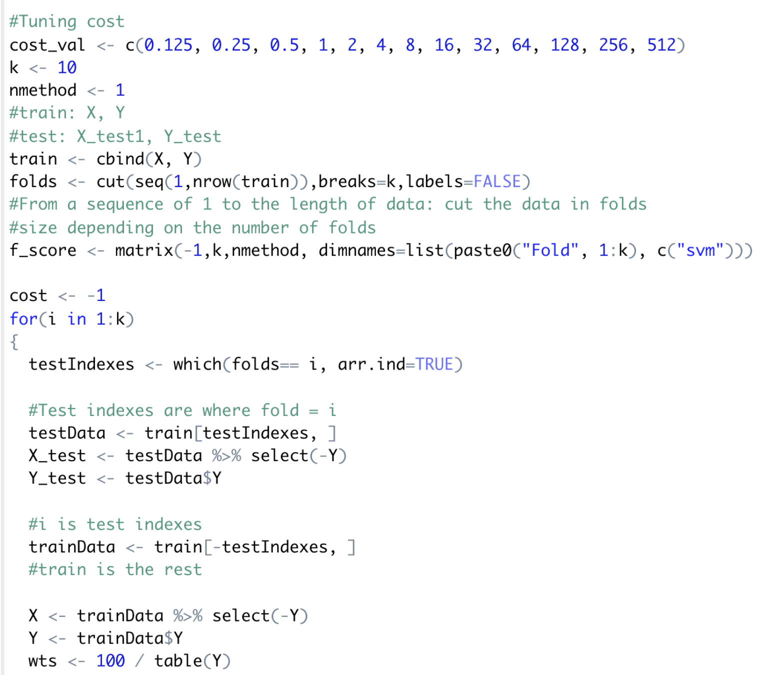
```
# logistic model classifier
logit_5 = LogisticRegression(random_state=27)
# define evaluation procedure
grid = GridSearchCV(logit_5, hyperparameter_grid, scoring="f1", cv=100, n_jobs=-1, refit=True)
grid.fit(X,y)
print(f'Best F1 score: {grid.best_score_} with parameters: {grid.best_params_}')
```

```
Best score: 0.23155389923114378 with param: {'C': 1.5, 'class_weight': {0: 1.0, 1: 10}, 'fit_intercept': True, 'penal
ty': 'l2'}
```

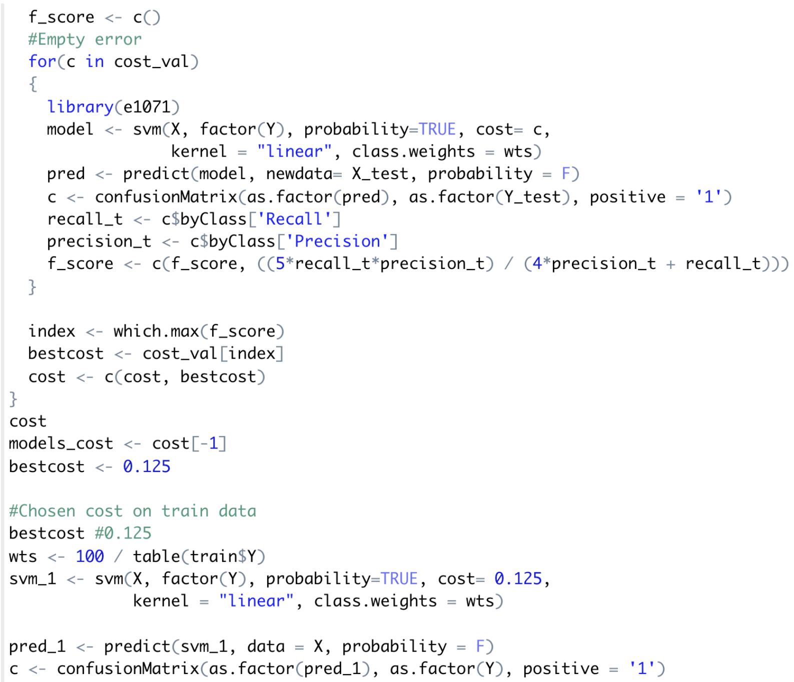
3.3.4. Support Vector Machine

10-fold cross validation to tune

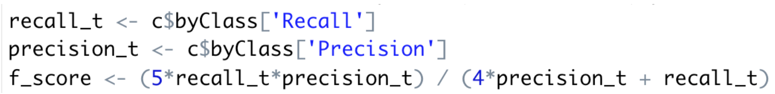
- cost parameter



```
#Tuning cost
cost_val <- c(0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512)
k <- 10
nmethod <- 1
#train: X, Y
#test: X_test1, Y_test
train <- cbind(X, Y)
folds <- cut(seq(1,nrow(train)),breaks=k,labels=FALSE)
#From a sequence of 1 to the length of data: cut the data in folds
#size depending on the number of folds
f_score <- matrix(-1,k,nmethod, dimnames=list(paste0("Fold", 1:k), c("svm")))

cost <- -1
for(i in 1:k)
{
  testIndexes <- which(folds== i, arr.ind=TRUE)

  #Test indexes are where fold = i
  testData <- train[testIndexes, ]
  X_test <- testData %>% select(-Y)
  Y_test <- testData$Y

  #i is test indexes
  trainData <- train[-testIndexes, ]
  #train is the rest

  X <- trainData %>% select(-Y)
  Y <- trainData$Y
  wts <- 100 / table(Y)
```



```
  f_score <- c()
  #Empty error
  for(c in cost_val)
  {
    library(e1071)
    model <- svm(X, factor(Y), probability=TRUE, cost= c,
                 kernel = "linear", class.weights = wts)
    pred <- predict(model, newdata= X_test, probability = F)
    c <- confusionMatrix(as.factor(pred), as.factor(Y_test), positive = '1')
    recall_t <- c$byClass['Recall']
    precision_t <- c$byClass['Precision']
    f_score <- c(f_score, ((5*recall_t*precision_t) / (4*precision_t + recall_t)))
  }

  index <- which.max(f_score)
  bestcost <- cost_val[index]
  cost <- c(cost, bestcost)
}
cost
models_cost <- cost[-1]
bestcost <- 0.125

#Chosen cost on train data
bestcost #0.125
wts <- 100 / table(train$Y)
svm_1 <- svm(X, factor(Y), probability=TRUE, cost= 0.125,
             kernel = "linear", class.weights = wts)

pred_1 <- predict(svm_1, data = X, probability = F)
c <- confusionMatrix(as.factor(pred_1), as.factor(Y), positive = '1')
```



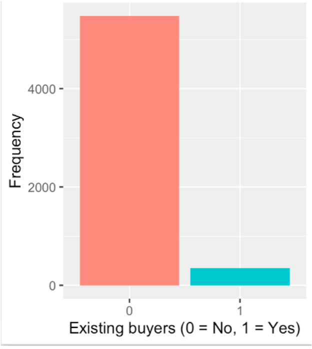
```
recall_t <- c$byClass['Recall']
precision_t <- c$byClass['Precision']
f_score <- (5*recall_t*precision_t) / (4*precision_t + recall_t)
```

4. Experiment Results

4.1. Data description

The data is taken from Kaggle's 2000 challenge[1]. This was a CoIL data mining challenge. The dataset is owned and supplied by Dutch data mining company Sentiment Machine Research and is a real world business problem. The problem statement is to classify customers as potential buyers of the Caravan Insurance Policy based on customer data of an insurance firm.

The dataset comprises of customer's socio-demographic, product ownership and insurance statistics data. The dataset consists of 9822 observations which represent customers and their characteristics. The characteristics are explained by 87 variables which includes socio- demographic, product ownership and insurance statistics. Variables starting with 'M' are socio-demographic related, starting with 'P' are product ownership related and starting with 'A' are insurance statistics related. A variable 'ORIGIN' determines the train and test observations which are 5822 and 4000 respectively. The target variable is CARAVAN, a 0/1 binary classifier, which tells us whether the customer buys Caravan insurance policy or not. The is a classification problem which predicts either 0: a customer would not buy Caravan's Insurance Policy or 1: a customer would buy Caravan's Insurance Policy.



The frequency distribution of the target variable can be seen above. As discussed, this is a highly imbalanced target with about buyers constituting 5.4% of the entire customer pool.

4.2. Experimental Process

Before application of models, exploratory data analysis was performed to identify features of interest and their contribution to the target variable. The target variable is a binary variable. The independent variables are either categorical (binary/ multiple levels) or numeric. For numeric independent variables, Analysis of Variance was performed and for categorical independent variables, chi- square testing is performed and tables are drawn.

ANOVA results- The following variables are significant:

- MGEMOV: Average household size for customers buying insurance is more
- AWAPART: Number of private third party insurance for customers buying insurance is more
- APERSAUT: Number of car policies bought by customers buying car insurance is more than the customers not buying
- ABROM: Number of mopped policies is less for customers buying Caravan insurance

- ALEVEN: Number of insurance policies bought by Caravan insurance buyers is generally more
- AGEZONG: Number of family accidents insurance policies by Caravan insurance buyers is generally more
- ABRAND: Number of fire policies by Caravan insurance buyers is more
- APLEIZER: Number of boat policies of Caravan insurance buyers is more
- ABYSTAND: Number of social security insurance policy of Caravan insurance buyers is more

Chi square test results- The following variables are significant:

- MOSHOOFD: Customer type
- MRELGE: Married
- MRELOV: Other relation
- MFALLEN: Singles
- MFWEKIND: Household with children
- MOPLHOOG: Higher level education
- MOPLMIDD: Medium level education
- MOPLLAAG: Lower level education
- MBERHOOG: High status
- MBERBOER: Farmer
- MBERMIDD: Middle management
- MBERARBG: Skilled laborer
- MBERARBO: Unskilled laborer
- MSKA: Social class A
- MSKB1: Social class B1
- MSKC: Social class C
- MSKD: Social class D
- MHHUUR: Rented house
- MHKOOP: Home owners
- MAUT1: 1 car
- MAUT0: 0 car

The evaluation criteria selection projected the aim as maximizing the F-score. Keeping the evaluation criteria in mind, the following models were fitted on the training data:

- On training data, a baseline model: logistic regression was fit. This model had a very high accuracy but an F Score of 0.015 on test data. this model was kept as the benchmark to make sure the further models perform better than this.

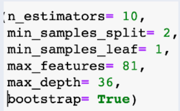
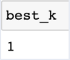
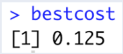
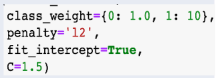
- Random Forest is an ensemble model which takes into accounts multiple decision trees fitted on the same train data with different number of features to reduce the variance of fit. The number of features was determined by 10-fold cross validation. The training data was further split into 10 folds. Each fold represented validation data and the other 9 folds represented the training data respectively in subsequent iterations. For each iteration, the best 'number of features' was the 'mtry' that proved to have the maximum F score on the validation data. The average of the number of features were determined as the optimal. Further, for identifying the optimal number of trees, maximum depth of the tree, minimum samples for splitting on a node and the minimum samples in the leaf node were optimized by grid search cross validation. All the optimized hyper parameters were combined to attain the best fit random forest. This model worked a lot better than the baseline.

- K- Nearest Neighbor re-iterates the training data every time a new test observation is encountered to classify the new test observation on the basis of majority votes of training examples. Since the original data was imbalanced, Synthetic Minority Oversampling Technique was performed on the data to reproduce synthetic observations belonging to class '1'. This data was scaled to assign equal weights to features. The hyper parameter 'k' which depicts the number of neighbors was tuned between a range of odd numbers between 1 to 39. The optimal 'k' was determined as '1' by hold-out cross validation. This 'k' was used to fit the model and predict results on the test data. This performed much better than random forest.

- Support vector Machine classifies observations to binary categories using the kernel trick. It takes one hot encoded data matrix for classification. The cost parameter allows the bending of the margin to classify misclassified points correctly on train data. This parameter was tuned using 10-fold cross validation. An optimal cost was attained and used to fit the model.

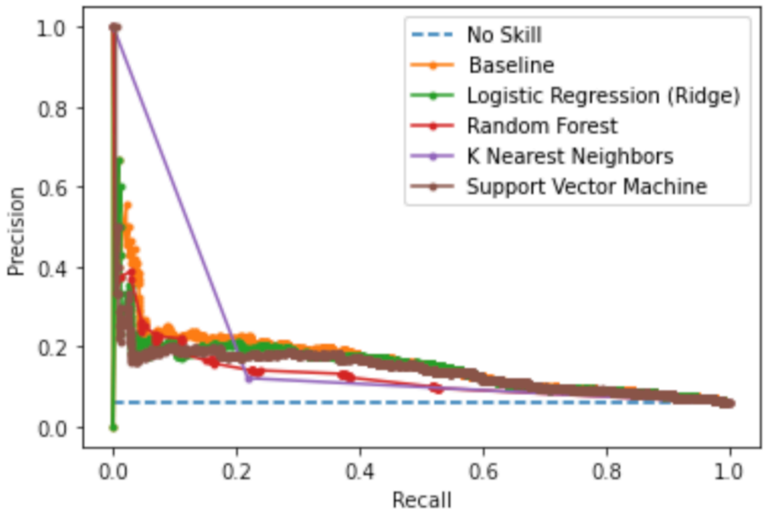
- Logistic regression classifies observations into different classes using generalized linear model family techniques. The sigmoid function is unique to the model which gives each observation a probability between 0 and 1. On the basis of an optimal cut-off, observations can be classified in either classes. Since classes are imbalanced, they were assigned class weights based on Grid Search Cross Validation. The other hyper parameters tuned in Grid Search CV were penalty, intercept fit and C (opposite of regularization strength). The penalty obtained was 'l2' which depicts a ridge regression fit on the train data. This model out performed all the other models.

The following table summarizes the techniques applied:

	Baseline (Logistic)	Random Forest	K Nearest Neighbors	Support Vector Machine	Logistic Regression (Ridge)
Pre-processing used in final model	-	-	Scaled data, applied SMOTE	One hot encoding	-
Technique used for hyper parameter tuning	-	10-fold cross validation and Grid Search CV	Hold out cross validation	10-fold cross validation	Grid Search CV
Hyper parameters tuned	-	Number of features, Number of trees, Maximum depth, Minimum samples required for splitting, Minimum samples required in leaf, Bootstrap	Number of neighbors	Cost	Class weights, Penalty (Ridge/Lasso), Intercept Fit, C (Inverse of regularization Strength)
Best parameters obtained	-	 (n_estimators= 10, min_samples_split= 2, min_samples_leaf= 1, max_features= 81, max_depth= 36, bootstrap= True)	 best_k 1	 > bestcost [1] 0.125	 class_weight={0: 1.0, 1: 10}, penalty='12', fit_intercept=True, C=1.5)

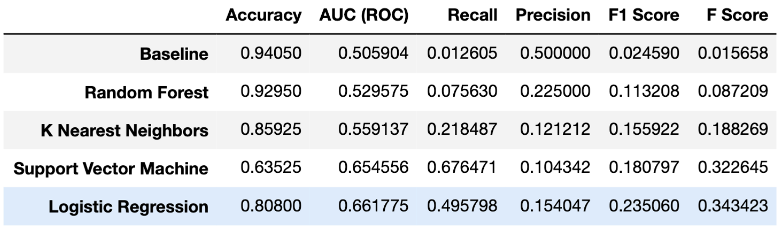
4.3. Outcomes

Precision Recall curves are the standard for comparing models on imbalanced data. Given, the class of interest as the positive class, the recall determines the proportion of identification and precision determines the proportion of correctness. PR-Curves compare models at different cut offs and the area under the curve gives us a sense of a better model overall. The following Precision Recall curve compares all the fitted models with each other and the baseline. It can be seen that majority models retrieve a very high precision with low recall at certain cut off points. As discussed, this is not a good fit as we are more interested in high recall than precision. At higher recall values, every model performs better than the baseline.



Since PR curves weighs Precision and Recall equally, we submit to F-score. The table depicts the results more clearly. Logistic Regression has the highest F Score, followed by SVM. This means that Logistic Regression is the most successful model in our entire pool to predict Caravan's Insurance Buyers.

Additionally, SVM has a better recall than Logistic Regression, as it predicts a higher majority to belong to class '1'. Either can be chosen as the best fit based on the problem statement.



	Accuracy	AUC (ROC)	Recall	Precision	F1 Score	F Score
Baseline	0.94050	0.505904	0.012605	0.500000	0.024590	0.015658
Random Forest	0.92950	0.529575	0.075630	0.225000	0.113208	0.087209
K Nearest Neighbors	0.85925	0.559137	0.218487	0.121212	0.155922	0.188269
Support Vector Machine	0.63525	0.654556	0.676471	0.104342	0.180797	0.322645
Logistic Regression	0.80800	0.661775	0.495798	0.154047	0.235060	0.343423

4.4. Analysis

The logistic regression model is better than naïve common classifier as it predicts some of the insurance buyers. The accuracy reduced by 1%, but the 95% confidence interval of accuracy is almost the same. Since this is an imbalanced class problem, sensitivity and positive predicted value are much more important than accuracy. The positive predicted value and sensitivity of the logistic regression has increased significantly. The baseline is a basic model and every model should perform better than the baseline. This goal can be considered to be fulfilled. Even though, Random Forest is the only ensemble model in the pool of models being considered; it performs the worst. The reason of this happening could be the higher bias in the decision tree for F score evaluation criteria. Random Forest only reduces the variance in multiple decision trees, there is a possibility that the bias for F1 score was high and hence lead to the fatality. K-Nearest Neighbors performs better than baseline and random forest probably because it identifies the nearest neighbor characteristic, which is identifying a similar customer. Two most similar customers may have similar buying psychology. Support Vector Machine and Logistic regression performs the best on the data because they use regularizers to fit better. There is a high scope of improvement in the project because understanding the psychology of buyers is very difficult through a small data of potential buyers. As more and more data accumulates, better prediction engines can be formulated.

5. Conclusion and Discussion

Dealing with imbalance data can be tricky but it can be seen that regularization technique works wonders and may not always require sampling techniques. This observation yields as the most important learning from the project. Besides, I experimented the following techniques for imbalance data:

- Assigning higher weights to classes of high importance result in better evaluation. The general practice used by statisticians: 100/ (target class distribution proportion) For example, the target variable is distributed as a 95-5 split. The majority class will be weighted 100/95, 1.05 and minority class will be weighted 100/5, i.e. 20. This practice can be used or further tuned using different weights and selecting the best weights using Grid Search Cross Validation.
- If you are using Synthetic Minority Oversampling Technique to balance your target variable, it may overfit your model. Cross validate the SMOTE fit model on train data before finalizing it as the best model.

For future scope and an ensemble model using tuned Adaptive Boosting, Support vector machine and ridge regression can be applied to improve prediction results on the current data. Additionally, as and when we have more information on buyers, the prediction engine will be more accurate on F-score.

6. References:

[1] https://www.kaggle.com/uciml/caravan-insurance-challenge

[2] https://en.wikipedia.org/wiki/F-score

[3] Rehan Akbani, Stephen Kwek, Nathalie Japkowicz. Applying Support Vector Machines to Imbalanced Datasets

[4] https://stackoverflow.com/questions/43877848/lasso-logistic-regression-suitable-for-imbalanced-data

[5] https://topepo.github.io/caret/subsampling-for-class-imbalances.html

[6] https://stats.stackexchange.com/questions/321970/imbalanced-data-smote-and-feature-selection

[7] https://scikit-learn.org/stable/modules/cross_validation.html

[8] https://medium.com/swlh/the-hyperparameter-cheat-sheet-770f1fed32ff