# Prediction of Insurance Policy Customers

# by Machine Learning Techniques

Xiner Zhou

UNI: xz2315

xz2315@columbia.edu

M.A. in Statistics, Columbia University

Table of Contents

**Part I: Preliminary Analysis3**

Intrduction3

Data Description3

Exploratory Data Analysis 3

Measure of Performance4

Feature Selection and Feature Importance6

**Part II: Machine Learning Methods8**

Naïve Bayes Classifier8

Bayesian Classification 8

Why Naïve Bayes and Its Assumptions 9

Naïve Bayes Model 9

Parameter Estimation 10

2-Fold Cross Validation and Model Selection 10

NB with Top-10 Features on Test Dataset 12

Support Vector Machine (SVM)13

Maximum Margin Motivation 13

Soft-Margin SVM 13

Non-linear SVM with Kernel 14

Full-fledged SVM 14

2-Fold Cross Validation and Model Selection 15

RBF Kernel with Gamma=0.1 & Cost=1000 on Test Dataset 18

Classification Tree18

Un-pruned Tree18

Feature Importance by Tree 19

Overfitting and Pruning 19

Model Selection 20

Best Classification Tree on Test Dataset 21

A Simple Tree for Interpretation Task 22

Boosting: Ensemble Method22

Pseudocode for AdaBoost23

AdaBoost using Decision Stump as Weak Learner 23

2-Fold Cross Validation and Model Selection 23

AdaBoost with 500 Decision Stumps on Test Dataset 25

**Part III: Conclusion26**

**Part IV: Appendix27**

A: Data Dictionary27

B: R Code29

**Abstract:** Ability to predict potential customers is crucial for insurance company or any other companies to market campaign. Machine learning algorithms have been successfully shown to generate high predictive accuracy on real-world data. In this project, four different supervised learning algorithms are discussed and applied to predict potential caravan insurance policy buyers on the insurance company (TIC) benchmark dataset: Naïve Bayes, SVM, Tree, and AdaBoost. Instead of looking at accuracy of prediction, another set of statistics is used: sensitivity, specificity, precision, and F1-score. Results show that Naïve Bayes generates superior predictive performance than the others in terms of sensitivity and F1-score.

**Keywords:** Machine learning, supervised learning, Naïve Bayes, SVM, Tree, AdaBoost, Sensitivity, Specificity, Precision, F1-score

**Part I: Preliminary Analysis**

1. Introduction

Direct mailings to a company’s potential customers—“junk mail” to many—can be a very effective way to market a product or service. Much of this junk mail is really of no interest to the majority of the people that receive it. Most of it ends up thrown away, not only wasting the money that the company spent on it, but also filling up landfill waste sites.

However, if the company had a better understanding of who their potential customers were, they would know more accurately who to send it to, so some of this waste and expense could be reduced. Therefore, the CoIL Challenge 2000 data mining competition provided The Insurance Company (TIC) Benchmark Dataset, and asked the question:

Can you predict who would be interested in buying a caravan insurance policy and give an explanation why?

2. Data Description

The Insurance Company (TIC) Benchmark Dataset was provided by the COIL Challenge 2000 which was organized by the Computational Intelligence and Learning (CoIL) cluster, a cooperation between four EU funded research networks. The problem is representative of an important class of real world learning problems: noisy, correlated, redundant and high dimensional data with a weak relationship between predictors and target. The homepage of TIC Benchmark is: http://www.liacs.nl/~putten/library/cc2000/

Data files:

(1) TICDATA2000.txt:

Dataset to train prediction models and build a description (5822 customer records). Each record consists of 86 attributes, containing sociodemographic data (attribute 1-43) and insurance policy ownership (attributes 44-86).The sociodemographic data is derived from zip codes. All customers living in areas with the same zip code have the same sociodemographic attributes. Attribute 86, "CARAVAN:Number of mobile home policies" (i.e. whether or not having the caravan insurance policy), is the target variable. The detailed meaning of the attributes and attribute values is given in Appendix A.

(2) TICEVAL2000.txt:

Dataset for test or prediction (4000 customer records). It has the same format as TICDATA2000.txt, only the target is missing.

(3) TICTGTS2000.txt:

Targets for the test set.

3. Exploratory Data Analysis

(1) Characteristic 1: Categorical Variables 🡪 Method of Feature Selection

Since all the 86 variables are indicators of levels or number of insurance policy ownership, they are all categorical. For example, the 4th column (sociodemographic data) representing “Average Age” has 6 levels; the 47th column (insurance policy ownership data) representing “Car Policy Contribution” has levels 0-8.

The characteristic that all of the variables are categorical inspires the way to do feature selection, as explained in the *section “Feature Selection”.*



(2) Characteristic 2: Only 6% Positive Response 🡪 Measures of Performance



In the supervised learning setting, the classification can be formulated as: let Y=1 if having caravan insurance policy and Y=0 if not, the goal is to classify observations into two classes Y=0 or Y=1. From the barplots above, we see that on both training and test sets, the positive response (Y=1) is an “rare event”, that is, only 6% of the sample belongs to class 1. This characteristic makes the “accuracy” of classification non-optimal for measuring performance of different models, as discussed in the *section “Measure of Performance”.*

4. Measure of Performance

Given that only 6% of customers on both training and test sets actually owns the caravan insurance policy, regular 0-1 loss classification “accuracy” or “error rate” is not appropriate measure of performance for different classification models. A “naïve” model that simply predicts no one will buy has a high classification accuracy of 94% but is useless.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | True Class | |  |
| 1=Positive | 0=Negative |
| Predicted  Class | 1  =Positive | True Positive | False Positive  ( Type I error) | Precision Rate=TP/(TP+FP) |
| 0  =Negative | False Negative  ( Type II error ) | True Negative |  |
|  | | Sensitivity=TP/(TP+FN) | Specificity=TN/(TN+FP) |

**Sensitivity**, **Specificity**, and **Precision Rate**, are statistical measures of the performance of a *binary classification test*, also known in statistics as “*classification function*”.

* **Sensitivity** ( also called the *true positive rate* ):

measures the proportion of actual positive which are correctly identified as positive.

Sensitivity relates to the test’s ability to identify positive results. For example, a sensitivity of 100% means that the test recognizes all actual positives. A test with a high sensitivity has a low type II error.

* **Specificity** ( also called the *true negative rate* ):

measures the proportion of negatives which are correctly identified as negative

Specificity relates to the test’s ability to identify negative results. A test with a high specificity has a low type I error.

* **Precision Rate**:

the proportion of positive results that are true positive.

We prefer the tests with high sensitivity, specificity, and precision. But there are intrinsic relationship among them and there is a trade-off. For example, a classifier simply predicts no one will buy caravan policy has specificity=100%, but sensitivity=0%; while another classifier predicts everyone will buy has sensitivity=100%, but sensitivity=0% and precision=6%. Both of the extreme cases are not ideal. Thus, we introduce the **F1-score**.

**F1-Score**

In statistics, the F1-score is a measure of a test’s accuracy. It considers both the precision rate and the sensitivity to compute the score. The F1-score can be interpreted as a weighted average of the precision rate and sensitivity, where an F1-score reaches its best value at 1 and worst score at 0.

The F1-score is often used in machine learning for measuring classification performance. Hence, we will use the F1-score to measure model performance and do model selection.

5. Feature Selection and Feature Importance

The TIC dataset contains 85 features, possibly highly correlated. To find a reliable subset of features for predictive model is the first important task.

*Initial Guess*: Among the 85 features, containing sociodemographic data (feature 1-43) and insurance product ownership (feature 44-85). Some people would think the sociodemographic feature unimportant, since they’re derived from zip codes and all customers living in the same zip code have the same sociodemographic features, thus can’t reflect individual information. But afterwards we’ll see some of them are really informative and others are truly uninformative. Since the target variable is the caravan insurance policy ownership, we would expect the features relating to car important.

**Information Gain**

The usual statistics *Pearson’s correlation coefficient* is a measure of linear relationship for continuous variables, thus inappropriate for our case. We’ll use “*information gain*” as a measure of predictive power for individual feature.

* Entropy:

is a measure of the amount of uncertainty.

When H(Y)=0, Y is perfectly classified, i.e. all points are of the same class. The higher the H(Y), the more uncertainty about Y.

* Information Gain IG(X):

is a measure of the difference in entropy from before to after Y is split on X, i.e. how much uncertainty in Y was reduced after split on X.

If X and Y are independent, then knowing X does not provide any additional information about Y, i.e. H(Y|X)=H(Y) 🡪 IG(X)=0; If X and Y are identical, then knowing X perfectly determines Y, i.e. H(Y|X)=0 🡪 IG(X)=H(Y). Thus, information gain is a measure of the inherent dependence of X and Y, the larger the IG(X) the greater predictive power of X.

### TIC Data Feature Ranking by Information Gain



The top-10 strongest features are:

1. Car policy contribution

Expected insurance ownership relating to car

Other insurance product ownership

Sociodemographic variable

1. Fire Policy contribution
2. Number of Car policies
3. Customer subtype
4. Customer main type
5. Average income
6. Purchasing power class
7. Income < 30
8. Lower level education
9. Private third party insurance contribution

The most and third-most important variables are truly the same as expected which are indicators relating to car; there are also some features relating to personal wealth, it’s reasonable since they are related to purchasing power; but the fire policy contribution, lower level education, and private third party insurance contribution, are somewhat surprising here.

The question of feature selection is how many features should be included in our predictive models, and it is an issue of model complexity and will be choosen according to difference types of classification model.

**Part II: Machine Learning Methods**

1. NAÏVE BAYES CLASSIFIER

### Bayesian Classification

In the supervised learning problem, let Y=1 if having caravan insurance policy and Y=0 if not, let denote the ith feature. In the risk framework, the best classifier is the one which minimizes the empirical risk. The risk of a classifier under 0-1 loss is defined as:

Minimizing R(f) is equivalent to minimize for every x, by monotonicity of the integral. Instead, minimizing:

Thus, the Bayes-optimal classifier under 0-1 loss is:

That is, the Bayes-optimal classification rule is “maximizing posterior probability”. But what is posterior probability ?

Bayes Equation:

The rest problem is, how to model class-conditional distribution ?

* Parametric method: Gaussian discriminant Analysis (LDA/QDA)
* Nonparametric method: Naïve Bayes Classification

### Why Naïve Bayes and Its Assumption

Although the Gaussian discriminant analysis is a popular classification model, it’s not appropriate for two reasons:

* GDA assumes features are continuous and normally distributed; but in our TIC case, features are all categorical;
* When features space is high-dimensional, it’s hard to estimate Gaussian covariance matrix.

While Naïve Bayes makes a strong but elegant assumption.

NB assumption:

Features are conditional independent given class labels.

### Naïve Bayes Model

Naïve Bayes Classifier

We need to model the class-conditional probability distribution for each feature. Since all the features are categorical, the multinomial distribution is an ideal first choice. **We can think of the TIC data sampled following the Bayes model intuitively as two steps:**

1. **Randomly sample a customer from a Bernoulli process with probability , Y=1 if having caravan policy and Y=0 if not;**
2. **Then the customer decide each of his/her features from a multinomial**

**distribution:**

Then, we can get the full Naïve Bayes Model for our problem:

Naïve Bayes Model

where:

### Parameter Estimation

We’re given a training set, we use *Maximum Likelihood Estimation* (MLE) to train parameters.

Maximum Likelihood Estimators:

where: is the j-th value of the p-th feature

is the “phantom parameter”, which is the number of values

of the p-th feature

### 2-Fold Cross Validation and Model Selection

**Tuning Parameter**: number of features

**Purpose of Cross Validation**: The training process optimizes the tuning parameters to make the model fit the training data as well as possible. If we then take an independent dataset from the same distribution as the training dataset, it will generally turn out the model does not fit as well as it fits the training set. This is called “*Overfitting*”. Cross-validation is a way to trade-off between overfitting and underfitting.

**Why 2-Fold Cross Validation**:

The TIC training dataset contains 5822 observations but only 348 positive instances which is really small. So we can’t make the number of folds too large. The 2-fold CV would be optimal. We simply split the training dataset into two equal size folds, and train on fold1 and test on fold2, followed by train on fold2 and test on fold1. This has the advantage that our **training and validation sets are both large enough, and each data points is used for both training and testing.**

The TIC Training Dataset

|  |  |
| --- | --- |
| fold 1 | fold 2 |

**2-Fold CV Results on TIC Training Set**

* Both training and test accuracy are very high ( 80%-92%). Interesting, the accuracy decays when number of features in the model increases.



* The green line shows that specificity are always very large, i.e., NB classifier always has good ability to identify class 0;
* The red line shows that sensitivity are generally small, i.e., NB classifier's ability to identify class 1 increases when the number of features included in the model increases;
* The purple line shows that precision decays with the number of features.



* **The F1-score reaches its maximum at number of features=10. Hence, the optimal number of feature in the NB model should be 10.**

### NB with Top-10 Features on Test Dataset

F1= 0.2060811

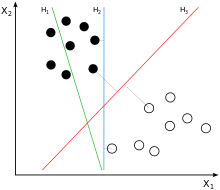
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | True Class | |  |
| 1=Positive | 0=Negative |
| Predicted  Class | 1  =Positive | 61 | 293 | Precision=17.23% |
| 0  =Negative | 177 | 3469 |  |
|  | | Sensitivity= 25.63% | Specificity= 92.21% |

* **The Naive Bayes model able to identify 25.63% of customers who actually having caravan insurance policy, 92.21% of those who actually don't have, and about 17.23% of those who are predicted to have caravan policy are actually policy owners.**

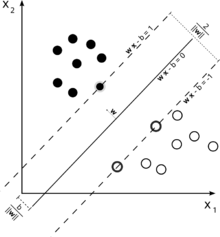
2. Support Vector Machine (SVM)

In machine learning, support vector machines (SVMs) are supervised learning models used for binary classification. It's a generalization of perceptron which only works for linearly separable problems, thus SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. The TIC data is apparently linear un-separable, SVMs are good candidate classifiers.

### Maximum Margin Motivation



For linear separable data, there’re multiple hyper-planes that might classify the data. But they have different predictive performance on another independent data. The maximum margin idea is, in order to achieve good generalization (low prediction error), place the hyper-plane in the “middle” between two classes; more precisely, choose hyperplane such that distances to the closest point in each class is maximal. This distance is called the “**margin**”. Since without distribution assumption, best guess is symmetric.



The SVM classifier is obtained by solving the optimization problem.

**SVM classifier**:

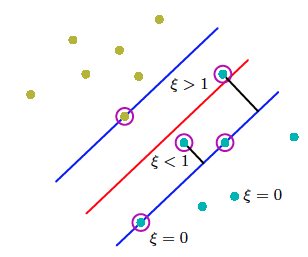
### Soft-Margin SVM

* Motivation 1: non-separable data

SVMs with maximum margin are linear classifiers without further modification, they can’t be trained on a non-separable data. (The TIC data is non-separable)

* Motivation 2: robustness

Suppose we have two training samples and train an SVM on each, if locations of support vectors vary significantly between samples, SVM classifiers vary significantly, thus SVM are not “robust”.

The idea of soft-margin SVM is to introduce “**slack variables** ”, and permit training data to cross the margin or even of the hyperplane, but impose cost which increases the further beyond the margin.

**Soft-margin SVM classifier**:

where: **Cost parameter C** is a tuning parameter, which specifies the cost of allowing a point on the wrong side.

### Non-Linear SVM with Kernel

The SVMs discussed so far uses the scalar product as a measure of similarity and distance between two points, since the scalar product is linear, the SVM is a linear classifier, i.e. the decision boundary is a hyperplane. But most empirical data can be separated more accurately by a non-linear classifier. By using kernel functions, we can make the decision boundary non-linear.

A function is called a “**Kernel**” on ,

if there is some function such that

i.e. **k is a kernel is it can be interpreted as a scalar product on some other space** .

If we substitute the kernel function for in all equations, we implicitly train a linear SVM on the higher-dimensional space on which a linear SVM works well, but back to the original space, the decision boundary is non-linear.

**Soft-margin SVM with kernel**:

where: **Cost parameter C** is a tuning parameter, which specifies the cost of allowing a point on the wrong side.

### Full-Fledged SVM

**Ingredient** **Purpose**

Maximum margin Good generalization property

Slack variables Overlapping classes, Robustness

Kernel Non-linear decision boundary

### 2-Fold Cross Validation and Model Selection

The effectiveness of SVM depends on the selection of kernel and soft margin parameter . A common choice is RBF kernel.

**RBF Kernel**:

Two tuning parameters:

1. Kernel parameter : Intuitively, the defines how far the influence of a single training data reaches, with low values meaning “far” and high values meaning “close”.
2. Cost parameter C: It trades off misclassification of training data against simplicity of the decision boundary. A low cost makes the decision boundary smooth, while a high cost makes the decision boundary rough.









* No matter what the value of the kernel parameter gamma is, sensitivity always increases when the cost parameter increases.
* Sensitivity achieves its maximum at kernel parameter gamma=0.1 and cost >=10000.



* In contrast to sensitivity, specificity always decays, and achieves its maximum at Cost=1.





* **F1-score achieves its maximum when gamma=0.1 and cost=1000. Hence, the optimal SVM model is RBF kernel with gamma=0.1 and cost=1000.**

### RBF Kernel with gamma=0.1 & C=1000 on Test Dataset

F1= 0.1409922

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | True Class | |  |
| 1=Positive | 0=Negative |
| Predicted  Class | 1  =Positive | 27 | 118 | Precision=18.62% |
| 0  =Negative | 211 | 3644 |  |
|  | | Sensitivity= 11.34% | Specificity= 96.86% |

* **The SVM with RBF kernel model able to identify 11.34% of customers who actually having caravan insurance policy, 96.86% of those who actually don't have, and about 18.62% of those who are predicted to have caravan policy are actually policy owners.**

3. Classification Tree

Unlike SVMs, Naïve Bayes we have discussed, a classification tree partitions the feature space in a recursive manner and fit local methods in each region instead of a global model in a large feature space. Its most attractive advantages are interpretability and built-in feature selection by the impurity measure “*information gain*”.

### Un-Pruned Tree

### 

### Feature Importance by Tree



The top-10 strongest features are:

1. Car policy contribution
2. Number of Car policies
3. Fire Policy contribution

Expected insurance ownership relating to car

Other insurance product ownership

Sociodemographic variable

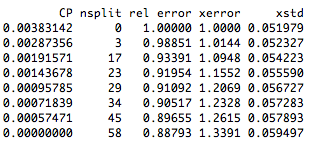
1. Average income
2. Customer subtype
3. Income < 30
4. Customer main type
5. Number of fire policy
6. Home owner
7. Singles

There’re some difference of the most important features by tree and individual predictive power, although they’re both based on “information gain”. **Tree select subsets of features one after another rather than just evaluate features individually and independently.** There may be several reasons to look at subset instead of single features. A feature with high individual predictive power but also high correlation to variables that are already selected does not add much information to the model, so it should not be included. A feature with low individual predictive power may have some complex joint relationship with a selected variable that is highly predictive, in which case it may be advisable to include it.

### Overfitting and Pruning

We have built a complete but too complex tree. To grow each branch of the tree just deeply enough to perfectly classify the training data, in fact it can lead to difficulties when there is noise in the training data, or when the number of trainings is too small to produce a representative sample of the true population. In either of these cases, this can produce trees that overfits the training data. **The accuracy of the tree over the trainings increases monotonically as the tree grows, however, the accuracy over the test set first increase then decreases. So we need to prune the tree.**





* The column labeled **CP** is the complexity parameter. Its serves as a penalty term to control tree size and is always monotonically decreases with the number of splits (**nsplit**). The smaller the CP, the more complex will be the tree (i.e. the larger the number of splits).
* The 10-fold cross-validation error (**xerror**) is the misclassification rate relative to the simplest tree with only the root node and no splitting. Thus, xerror=1 when nspli=0.
* From the misclassification perspective, for TIC data, since the xerror achieves minimum with no splitting, the best tree would be just assign every data to class 0.

### Model Selection

The effective of tree depends on how complex the tree grows, that is, how many split the tree contains. Thus, we use the F1-score to prune the tree and tune the parameter “**nplit**”.



* The training accuracy increases monotonically as the tree grows; however, the testing accuracy decreases monotonically as the tree grows.



* The green line shows that specificity are always very large, i.e., tree always has good ability to identify class 0;
* The red line shows that sensitivity are generally small, i.e., tree ability to identify class 1 increases when the number of features included in the model increases;
* The purple line shows that precision achieves maximum at nsplit=3.



* **F1-score achieves its maximum when nsplit=58. Hence, the optimal classification tree is no pruning.**

### Best Classification Tree on Test Dataset

F1= 0.160221

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | True Class | |  |
| 1=Positive | 0=Negative |
| Predicted  Class | 1  =Positive | 29 | 95 | Precision=23.39% |
| 0  =Negative | 209 | 3667 |  |
|  | | Sensitivity= 12.18% | Specificity= 97.47% |

* **The classification tree able to identify 12.18% of customers who actually having caravan insurance policy, 97.47% of those who actually don't have, and about 23.39% of those who are predicted to have caravan policy are actually policy owners.**

### A Simple Tree for Interpretation Task



* For the description task, **if a customer with Car policy contribution greater than 6, Fire policy contribution less than 2, and Boat policy contribution equal to 0, that is, very high level of Car policy contribution, very low level of Fire policy contribution, and no Boat policy contribution, then he/she is 33.3% likely to have or will buy the Caravan insurance policy**. This information provided by tree is extremely informative and intuitively interpretable, thus can be used in practice for the insurance company.

4. Boosting: Ensemble Method

An **ensemble method** makes a prediction by combining the predictions of many classifiers into a single vote. The individual classifiers are usually required to perform only slightly better than random. This means slightly more than 50% of the data are classified correctly. Such a classifier is called a **weak learner**. If the weak learners are random and independent, the prediction accuracy of the majority vote will increase with the number of weak learners. Since the weak learners all have to be trained on the same training set, producing random and independent weak learners is difficult. Different ensemble methods (Bagging, Boosting, and Random Forest, etc) use different strategyes to train and combine weak learners that behave relatively independent.

**Boosting** is arguably the most popular (and historically the first) ensemble method. The independence of weak learners is obtained by modifying the training data using weights after training each weak learner. The **AdaBoost** algorithm of Freud and Schapire was the first practiceal boosting algorithm, and remains one of the most widely used and standard.

### Pseudocode for AdaBoost

In AdaBoost, a weight value is assigned to each training data. At each step, data points which are correctly classified correctly are weighted down and which are misclassified are weighted up, i.e. the weight is smaller the more the weak learners already trained classify the data point correctly, and vice versa. So the next weak learner which is trained on the weighted dataset focuses more on data points which are “hard to classify” by previous weak learners. Roughly speaking, each weak learner tries to get those points right which are currently not classified correctly.

Input:

* Training data
* Algorithm parameter: number of weak learners T

Training Algorithm:

* Initialize weights:
* For t=1,…,T:

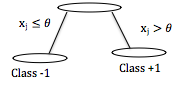
Train a weak learner using weights

Weight Update:

* Output the ensemble classifier:

### AdaBoost using Decision Stump as Weak Learner

The simplest tree is a tree of depth 1. Such a classifier is called a **decision stump**. A decision stump is parameterized by a pair of an axis j and a splitting point . Decision boundary is a hyperplane which is perpendicular to axis j and intersects the axis at



### 2-Fold CV and Model Selection

The effective of AdaBoost depends on the tuning parameter: number of weak learners. Thus, we use the F1-score to do cross validation and tune the parameter.



* The training accuracy increases when the number of weak learners increases, and when greater than 500, the training accuracy >99%.
* The test accuracy does not increase with the number of weak learners.



* Sensitivity, specificity, and precision do not vary significantly with the number of weak learners, this means that AdaBoost algorithm does not have good improvement regarding sensitivity, specificity, and precision. This is reasonable, since AdaBoost focuses on error rate.



* **F1-score achieves its maximum when combining 500 weak learners, but ensemble method does not have superior performance for the TIC dataset.**

### AdaBoost with 500 Decision Stumps on Test Dataset

F1= 0.1007194

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | True Class | |  |
| 1=Positive | 0=Negative |
| Predicted  Class | 1  =Positive | 14 | 26 | Precision= 35% |
| 0  =Negative | 224 | 3636 |  |
|  | | Sensitivity= 5.88% | Specificity= 99.31% |

* **The classification tree able to identify 5.88% of customers who actually having caravan insurance policy, 99.31% of those who actually don't have, and about 35% of those who are predicted to have caravan policy are actually policy owners.**

**Part III**

Conclusion



**The insurance company’s main question is who would buy the product, thus we should focus on tests’ ability to identify positive results—Sensitivity.** The greater the sensitivity the more potential customers we would detect. **But we also need to look at the precision, since there is a trade-off between cost of mailing and profit of positive** **customers**, but in this case, the four algorithms’ precision do not have major difference. Thus, specificity is not a major concern in this case. **The F1-score is a weighted average of sensitivity and precision, and is the statistics we have used to do model selection.**

**Both of sensitivity and F1-score give the same suggestion that Naïve Bayes is the best classifier for our case**. The **Naïve Bayes model** with the top-10 most important features is able to:

* identify 25.63% of customers who actually having caravan insurance policy, so **about ¼ potential customers will recognized**;
* identify 92.21% of those who actually don't have, so **more than 9/10 of those won’t buy the caravan policy will be excluded from the mailing list, which save a lot of unnecessary expense for the company**;
* about 17.23% of those who are predicted to have caravan policy are actually policy owners, so we have the confidence that **about 1/5 of those remain in the mailing list will buy caravan policy**.

**Part IV Appendix**

**Appendix A: Data Dictionary**

Nr Name Description Domain

1 MOSTYPE Customer Subtype see L0

2 MAANTHUI Number of houses 1 – 10

3 MGEMOMV Avg size household 1 – 6

4 MGEMLEEF Avg age see L1

5 MOSHOOFD Customer main type see L2

6 MGODRK Roman catholic see L3

7 MGODPR Protestant ...

8 MGODOV Other religion

9 MGODGE No religion

10 MRELGE Married

11 MRELSA Living together

12 MRELOV Other relation

13 MFALLEEN Singles

14 MFGEKIND Household without children

15 MFWEKIND Household with children

16 MOPLHOOG High level education

17 MOPLMIDD Medium level education

18 MOPLLAAG Lower level education

19 MBERHOOG High status

20 MBERZELF Entrepreneur

21 MBERBOER Farmer

22 MBERMIDD Middle management

23 MBERARBG Skilled labourers

24 MBERARBO Unskilled labourers

25 MSKA Social class A

26 MSKB1 Social class B1

27 MSKB2 Social class B2

28 MSKC Social class C

29 MSKD Social class D

30 MHHUUR Rented house

31 MHKOOP Home owners

32 MAUT1 1 car

33 MAUT2 2 cars

34 MAUT0 No car

35 MZFONDS National Health Service

36 MZPART Private health insurance

37 MINKM30 Income < 30.000

38 MINK3045 Income 30-45.000

39 MINK4575 Income 45-75.000

40 MINK7512 Income 75-122.000

41 MINK123M Income >123.000

42 MINKGEM Average income

43 MKOOPKLA Purchasing power class

44 PWAPART Contribution private third party insurance see L4

45 PWABEDR Contribution third party insurance (firms) ...

46 PWALAND Contribution third party insurane (agriculture)

47 PPERSAUT Contribution car policies

48 PBESAUT Contribution delivery van policies

49 PMOTSCO Contribution motorcycle/scooter policies

50 PVRAAUT Contribution lorry policies

51 PAANHANG Contribution trailer policies

52 PTRACTOR Contribution tractor policies

53 PWERKT Contribution agricultural machines policies

54 PBROM Contribution moped policies

55 PLEVEN Contribution life insurances

56 PPERSONG Contribution private accident insurance policies

57 PGEZONG Contribution family accidents insurance policies

58 PWAOREG Contribution disability insurance policies

59 PBRAND Contribution fire policies

60 PZEILPL Contribution surfboard policies

61 PPLEZIER Contribution boat policies

62 PFIETS Contribution bicycle policies

63 PINBOED Contribution property insurance policies

64 PBYSTAND Contribution social security insurance policies

65 AWAPART Number of private third party insurance 1 - 12

66 AWABEDR Number of third party insurance (firms) ...

67 AWALAND Number of third party insurane (agriculture)

68 APERSAUT Number of car policies

69 ABESAUT Number of delivery van policies

70 AMOTSCO Number of motorcycle/scooter policies

71 AVRAAUT Number of lorry policies

72 AAANHANG Number of trailer policies

73 ATRACTOR Number of tractor policies

74 AWERKT Number of agricultural machines policies

75 ABROM Number of moped policies

76 ALEVEN Number of life insurances

77 APERSONG Number of private accident insurance policies

78 AGEZONG Number of family accidents insurance policies

79 AWAOREG Number of disability insurance policies

80 ABRAND Number of fire policies

81 AZEILPL Number of surfboard policies

82 APLEZIER Number of boat policies

83 AFIETS Number of bicycle policies

84 AINBOED Number of property insurance policies

85 ABYSTAND Number of social security insurance policies

86 CARAVAN Number of mobile home policies 0 – 1

L0:

Value Label

1 1 High Income, expensive child

2 2 Very Important Provincials

3 3 High status seniors

4 4 Affluent senior apartments

5 5 Mixed seniors

6 6 Career and childcare

7 7 Dinki's (double income no kids)

8 8 Middle class families

9 9 Modern, complete families

10 10 Stable family

11 11 Family starters

12 12 Affluent young families

13 13 Young all american family

14 14 Junior cosmopolitan

15 15 Senior cosmopolitans

16 16 Students in apartments

17 17 Fresh masters in the city

18 18 Single youth

19 19 Suburban youth

20 20 Etnically diverse

21 21 Young urban have-nots

22 22 Mixed apartment dwellers

23 23 Young and rising

24 24 Young, low educated

25 25 Young seniors in the city

26 26 Own home elderly

27 27 Seniors in apartments

28 28 Residential elderly

29 29 Porchless seniors: no front yard

30 30 Religious elderly singles

31 31 Low income catholics

32 32 Mixed seniors

33 33 Lower class large families

34 34 Large family, employed child

35 35 Village families

36 36 Couples with teens 'Married with children'

37 37 Mixed small town dwellers

38 38 Traditional families

39 39 Large religous families

40 40 Large family farms

41 41 Mixed rurals

L1:

1 20-30 years

2 30-40 years

3 40-50 years

4 50-60 years

5 60-70 years

6 70-80 years

L2:

1 Successful hedonists

2 Driven Growers

3 Average Family

4 Career Loners

5 Living well

6 Cruising Seniors

7 Retired and Religeous

8 Family with grown ups

9 Conservative families

10 Farmers

L3:

0 0%

1 1 - 10%

2 11 - 23%

3 24 - 36%

4 37 - 49%

5 50 - 62%

6 63 - 75%

7 76 - 88%

8 89 - 99%

9 100%

L4:

0 f 0

1 f 1 – 49

2 f 50 – 99

3 f 100 – 199

4 f 200 – 499

5 f 500 – 999

6 f 1000 – 4999

7 f 5000 – 9999

8 f 10.000 - 19.999

9 f 20.000 - ?

**Appendix B: R Code**

# ============ Import Data =========================================================================

getwd()

setwd("/Users/cindy/Downloads")

ticdata2000<-read.table("ticdata2000.txt",header=FALSE,sep="\t") #training

ticeval2000<-read.table("ticeval2000.txt",header=FALSE,sep='\t') #test

tictgts2000<-read.table("tictgts2000.txt",header=FALSE,sep="\t") #test label

# =========== Exploratory Data Analysis ================================================================

counts <- table(ticdata2000[,4])

barplot(counts,col="green",names.arg=c("1: 20-30 years", "2: 30-40 years", "3: 40-50 years", "4: 50-60 years", "5: 60-70 years", "6: 70-80 years"))

title(main="Barplot for Levels of Average Age",col.main="blue",font.main=4)

counts <- table(ticdata2000[,47])

barplot(counts,col="green",names.arg=c("Level=0","Level=4","Level=5","Level=6","Level=7","Level=8"))

title(main="Barplot for Levels of Car Policy Contribution",col.main="blue",font.main=4)

counts <- table(ticdata2000[,68])

barplot(counts,col="green",names.arg=c("0 Car Policy","1 Car Policy","2 Car Policy","3 Car Policy","4 Car Policy","6 Car Policy","7 Car Policy"))

title(main="Barplot for Number of Car Policy",col.main="blue",font.main=4)

par(mfrow=c(1,2))

counts1 <- table(ticdata2000[,86])

barplot(counts1,col="red",names.arg=c("Y=0:No Caravan Insurance Policy","Y=1:Having Caravan Insurance Policy"),xlab="Number of Caravan Insurance Policy",ylab="Count",col.lab=rgb(0,0.5,0),cex.names=0.7)

title(main="Distribution of Target Var(Y) on Training Set",col.main="blue",font.main=4)

legend("topright", c("Y=1%: 6%"), cex=0.9)

counts2 <- table(tictgts2000)

barplot(counts2,col="red",names.arg=c("Y=0:No Caravan Insurance Policy","Y=1:Having Caravan Insurance Policy"),xlab="Number of Caravan Insurance Policy",ylab="Count",col.lab=rgb(0,0.5,0),cex.names=0.7)

title(main="Distribution of Target Var(Y) on Test Set",col.main="blue",font.main=4)

legend("topright", c("Y=1%: 6%"), cex=0.9)

# ============Feature Selection====================================================================

library(infotheo)

# vector MI contains the mutual informations for each feature

MI<-rep(0,85)

for(i in 1:85){

MI[i]<-mutinformation(ticdata2000[,i],ticdata2000[,86],method="emp")

}

MI.percent<-MI/MI[which.max(MI)]

# vector o contains the ordering according to MI

o<-order(MI,decreasing=TRUE)

var.name<-names(ticdata2000)[-86]

# data.frame MI.sort.df contains:feature names,mutual information and percentage,sorted from high to low

MI.sort.df<-data.frame(var=var.name[o],mutualinfo=MI[o],percent=MI.percent[o])

#plot feature importance

mycolor<-c(rep("red",10),rep("hotpink",10),rep("pink",10),rep("yellow",10),rep("lightskyblue",10),rep("slateblue",10),rep("gray",25))

barplot(MI.sort.df$percent,col=mycolor,xlab="Feature",ylab="Percentage of IG relative to Best Feature ",col.lab=rgb(0,0.5,0))

title(main="Predictive Power by Information Gain",col.main="blue",font.main=4)

#top 10 features

MI.sort.df$var[1:10]

# ===================Make Training and Test Sets======================================================

tic.train.x<-ticdata2000[,o]

tic.train.y<-ticdata2000[,86]

tic.test.x<-ticeval2000[,o]

tic.test.y <-tictgts2000$V1

fold1 <- c(1:2911)

fold2 <- c(2912:5822)

# ========================= Naive Bayes Classification==================================================

NaiveBayes <- function(Xtrain,Ytrain,Xtest,d){

#class probability y=0

prob.y0<-length(which(Ytrain==0))/length(Ytrain)

#class probability y=1

prob.y1<-length(which(Ytrain==1))/length(Ytrain)

# predicted value vector for test set

Ytest<-rep(0,dim(Xtest)[1])

# log probability matrices for two class

log.prob.y0 <- mat.or.vec(dim(Xtest)[1],d)

log.prob.y1 <- mat.or.vec(dim(Xtest)[1],d)

for(i in 1:d){

# extract possible values for each feature from both training and test sets

values<-unique(c(Xtrain[,i],Xtest[,i]))

# mulinomial parameters for feature i,row 1 for class y=0,row 2 for class y=1

par.est<-mat.or.vec(2,length(values))

for(j in 1:length(values)){

par.est[1,j]<-(1+length(which(Ytrain==0 & Xtrain[,i]==values[j])))/(length(values)+length(which(Ytrain==0)))

par.est[2,j]<-(1+length(which(Ytrain==1 & Xtrain[,i]==values[j])))/(length(values)+length(which(Ytrain==1)))

}

for(obs in 1:dim(Xtest)[1]){

log.prob.y0[obs,i]<-log(par.est[1,which(values==Xtest[obs,i])])

log.prob.y1[obs,i]<-log(par.est[2,which(values==Xtest[obs,i])])

}

}

prob.class0<-rowSums(log.prob.y0)+log(prob.y0)

prob.class1<-rowSums(log.prob.y1)+log(prob.y1)

for(h in 1:dim(Xtest)[1]){if(prob.class1[h]>=prob.class0[h]){Ytest[h]<-1}}

return(Ytest)

}

tr.accuracy <-rep(0,17)

te.accuracy <-rep(0,17)

sens <-rep(0,17)

spec <-rep(0,17)

prec<-rep(0,17)

F1 <-rep(0,17)

npar <-seq(from=5,to=85,by=5)

for(n in npar){

y1 <- NaiveBayes(tic.train.x[fold1,],tic.train.y[fold1],tic.train.x,n)

y2 <- NaiveBayes(tic.train.x[fold2,],tic.train.y[fold2],tic.train.x,n)

# train accuracy

tr.a1 <- (length(which(y1[fold1]==1&tic.train.y[fold1]==1))+length(which(y1[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

tr.a2 <- (length(which(y2[fold2]==1&tic.train.y[fold2]==1))+length(which(y2[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

tr.accuracy[n/5] <-(tr.a1+tr.a2)/2

# test accuracy

te.a1 <- (length(which(y1[fold2]==1&tic.train.y[fold2]==1))+length(which(y1[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

te.a2 <- (length(which(y2[fold1]==1&tic.train.y[fold1]==1))+length(which(y2[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

te.accuracy[n/5] <- (te.a1+te.a2)/2

# sensitivity

sens1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(tic.train.y[fold2]==1))

sens2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(tic.train.y[fold1]==1))

sens[n/5]<-(sens1+sens2)/2

# specificity

spec1 <- length(which(y1[fold2]==0&tic.train.y[fold2]==0))/length(which(tic.train.y[fold2]==0))

spec2 <- length(which(y2[fold1]==0&tic.train.y[fold1]==0))/length(which(tic.train.y[fold1]==0))

spec[n/5] <-(spec1+spec2)/2

# precision

prec1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(y1[fold2]==1))

prec2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(y2[fold1]==1))

prec[n/5]<-(prec1+prec2)/2

# F1-score

F1.1<- 2\*(prec1\*sens1)/(prec1+sens1)

F1.2<- 2\*(prec2\*sens2)/(prec2+sens2)

F1[n/5]<-(F1.1+F1.2)/2

}

# accuracy plot

plot(te.accuracy,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(te.accuracy,tr.accuracy)),ylab="Train/Test Accuracy %",col.lab=rgb(0,0.5,0))

lines(tr.accuracy,type="b",col="green")

title(main="Train and Test Accuracy",col.main="blue",font.main=4)

axis(1,at=1:length(npar),labels=npar)

axis(2,at=c(0.8,0.825,0.85,0.875,0.9,0.925,0.95),labels=c("80%","82.5%","85%","87.5%","90%","92.5%","95%"))

legend("topright", c("Test Accuracy","Training Accuracy"), cex=0.6,

col=c("red","green"), lty=1)

# sensitivity,specificity,precision plot

plot(sens,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(sens,spec,prec)),ylab="Sensitivity/Specificity/Precision %",col.lab=rgb(0,0.5,0))

lines(spec,type="b",col="green")

lines(prec,type="b",col="purple")

title(main="Sensitivity/Specificity/Precision",col.main="blue",font.main=4)

axis(1,at=1:length(npar),labels=npar)

axis(2,at=seq(from=0.1,to=1,by=0.1),labels=c("10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"))

legend("topright", c("Sensitivity","Specificity","Precision"), cex=0.6,

col=c("red","green","purple"), lty=1)

# F1 plot

plot(F1,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(F1)),ylab="F1-Score",col.lab=rgb(0,0.5,0))

title(main="F1-Score",col.main="blue",font.main=4)

axis(1,at=1:length(npar),labels=npar)

axis(2,at=seq(from=0.15,to=0.275,by=0.025),labels=seq(from=0.15,to=0.275,by=0.025))

# Test Set

y <- NaiveBayes(tic.train.x,tic.train.y,tic.test.x,10)

table(y,tic.test.y)

sensitivity<-length(which(y==1&tic.test.y==1))/length(which(tic.test.y==1))

specificity<-length(which(y==0&tic.test.y==0))/length(which(tic.test.y==0))

precision<-length(which(y==1&tic.test.y==1))/length(which(y==1))

F1<-2\*(precision\*sensitivity)/(precision+sensitivity)

# ======================SVM======================================================================

library(class)

library(e1071)

# =================2-Fold Cross Validation:RBF Kernels===================================================

# Tuning Parameter: cost, gamma

tr.accuracy <-mat.or.vec(4,7)

te.accuracy <-mat.or.vec(4,7)

sens <-mat.or.vec(4,7)

spec <-mat.or.vec(4,7)

prec<-mat.or.vec(4,7)

F1 <-mat.or.vec(4,7)

for(g in c(0.01,0.1,1,10)){

for(c in c(1,10,100,1000,10000,100000,1000000)){

model1<-svm(tic.train.x[fold1,1:10],tic.train.y[fold1],type="C",kernel="radial",cost=c,gamma=g)

y1<-predict(model1,tic.train.x[,1:10])

model2<-svm(tic.train.x[fold2,1:10],tic.train.y[fold2],type="C",kernel="radial",cost=c,gamma=g)

y2<-predict(model2,tic.train.x[,1:10])

# train accuracy

tr.a1 <- (length(which(y1[fold1]==1&tic.train.y[fold1]==1))+length(which(y1[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

tr.a2 <- (length(which(y2[fold2]==1&tic.train.y[fold2]==1))+length(which(y2[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

tr.accuracy[log10(g)+3,log10(c)+1] <-(tr.a1+tr.a2)/2

# test accuracy

te.a1 <- (length(which(y1[fold2]==1&tic.train.y[fold2]==1))+length(which(y1[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

te.a2 <- (length(which(y2[fold1]==1&tic.train.y[fold1]==1))+length(which(y2[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

te.accuracy[log10(g)+3,log10(c)+1] <- (te.a1+te.a2)/2

# sensitivity

sens1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(tic.train.y[fold2]==1))

sens2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(tic.train.y[fold1]==1))

sens[log10(g)+3,log10(c)+1]<-(sens1+sens2)/2

# specificity

spec1 <- length(which(y1[fold2]==0&tic.train.y[fold2]==0))/length(which(tic.train.y[fold2]==0))

spec2 <- length(which(y2[fold1]==0&tic.train.y[fold1]==0))/length(which(tic.train.y[fold1]==0))

spec[log10(g)+3,log10(c)+1] <-(spec1+spec2)/2

# precision

prec1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(y1[fold2]==1))

prec2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(y2[fold1]==1))

prec[log10(g)+3,log10(c)+1]<-(prec1+prec2)/2

# F1-score

F1.1<- 2\*(prec1\*sens1)/(prec1+sens1)

F1.2<- 2\*(prec2\*sens2)/(prec2+sens2)

F1[log10(g)+3,log10(c)+1]<-(F1.1+F1.2)/2

}

}

# accuracy plot

plot(tr.accuracy[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(tr.accuracy),ylab="Training Accuracy %",col.lab=rgb(0,0.5,0))

lines(tr.accuracy[2,],type="b",col="purple")

lines(tr.accuracy[3,],type="b",col="seagreen")

lines(tr.accuracy[4,],type="b",col="hotpink")

title(main="Training Accuracy",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0.94,0.95,0.96,0.97,0.98,0.99),labels=c("94%","95%","96%","97%","98%","99%"))

legend("bottomright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

plot(te.accuracy[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(te.accuracy),ylab="Test Accuracy %",col.lab=rgb(0,0.5,0))

lines(te.accuracy[2,],type="b",col="purple")

lines(te.accuracy[3,],type="b",col="seagreen")

lines(te.accuracy[4,],type="b",col="hotpink")

title(main="Test Accuracy",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0.87,0.88,0.89,0.90,0.91,0.92,0.93,0.94,0.95),labels=c("87%","88%","89%","90%","91%","92%","93%","94%","95%"))

legend("bottomright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

# sensitivity,specificity,precision plot

plot(sens[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(sens),ylab="Sensitivity %",col.lab=rgb(0,0.5,0))

lines(sens[2,],type="b",col="purple")

lines(sens[3,],type="b",col="seagreen")

lines(sens[4,],type="b",col="hotpink")

title(main="Sensitivity",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0,0.05,0.10,0.15,0.20),labels=c("0%","5%","10%","15%","20%"))

legend("topright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

plot(spec[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(spec),ylab="Specificity %",col.lab=rgb(0,0.5,0))

lines(spec[2,],type="b",col="purple")

lines(spec[3,],type="b",col="seagreen")

lines(spec[4,],type="b",col="hotpink")

title(main="Specificity",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0.9,0.92,0.94,0.96,0.98,1.00),labels=c("90%","92%","94%","96%","98%","100%"))

legend("topright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

plot(prec[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(prec),ylab="Precision %",col.lab=rgb(0,0.5,0))

lines(prec[2,],type="b",col="purple")

lines(prec[3,],type="b",col="seagreen")

lines(prec[4,],type="b",col="hotpink")

title(main="Precision",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0,0.04,0.08,0.12,0.16,0.20),labels=c("0%","4%","8%","12%","16%","20%"))

legend("topright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

# F1 plot

plot(F1[1,],type="b",col="yellowgreen",xlab="Cost C",axes=FALSE,ylim=range(F1),ylab="F1",col.lab=rgb(0,0.5,0))

lines(F1[2,],type="b",col="purple")

lines(F1[3,],type="b",col="seagreen")

lines(F1[4,],type="b",col="hotpink")

title(main="F1-Score",col.main="blue",font.main=4)

axis(1,at=1:7,labels=c(1,10,100,1000,10000,100000,1000000))

axis(2,at=c(0,0.2,0.4,0.6,0.8,0.10,0.12,0.14,0.16),labels=c(0,0.2,0.4,0.6,0.8,0.10,0.12,0.14,0.16))

legend("topright", c("gamma=0.01","gamma=0.1","gamma=1","gamma=10"), cex=0.6,

col=c("yellowgreen","purple","seagreen","hotpink"), lty=1)

# Test Set

model<-svm(tic.train.x[,1:10],tic.train.y,type="C",kernel="radial",cost=1000,gamma=0.1)

y<-predict(model,tic.test.x[,1:10])

table(y,tic.test.y)

sensitivity<-length(which(y==1&tic.test.y==1))/length(which(tic.test.y==1))

specificity<-length(which(y==0&tic.test.y==0))/length(which(tic.test.y==0))

precision<-length(which(y==1&tic.test.y==1))/length(which(y==1))

F1<-2\*(precision\*sensitivity)/(precision+sensitivity)

# =========================Classification Tree=======================================================

library(rpart)

library(cluster)

library(foreach)

library(lattice)

library(plyr)

library(reshape2)

library(caret)

# use all features to build a full tree

fit<-rpart(ticdata2000$V86~.,method="class", y=TRUE,control=rpart.control(cp=0,xval=2), parms=list(split="information"),data=ticdata2000)

#plot tree

plot(fit, margin=0,uniform=T, branch=1)

text(fit, use.n=TRUE, all=TRUE, cex=.5)

title(main="TIC Data using all features",col.main="blue",font.main=4)

# Feature importance

var.imp<-fit$variable.importance

mycolor1<-c(rep("red",10),rep("hotpink",10))

barplot(var.imp[c(1:20)],col=mycolor1,xlab="Feature",ylab="Importance",col.lab=rgb(0,0.5,0))

title(main="Feature Importance by Tree",col.main="blue",font.main=4)

# CP table

printcp(fit)

plotcp(fit,upper="size")

# Prune the tree

num.split<-fit$cptable[,"nsplit"] #possible number of splits

tr.accuracy <-rep(0,length(num.split))

te.accuracy <-rep(0,length(num.split))

sens <-rep(0,length(num.split))

spec <-rep(0,length(num.split))

prec<-rep(0,length(num.split))

F1 <-rep(0,length(num.split))

for(i in 1:length(num.split)){

fit.prune <- prune(fit,cp=fit$cptable[i,"CP"])

y.tr<- predict(fit.prune,ticdata2000[,1:85],type="class")

y.te <- predict(fit.prune,ticeval2000,type="class")

# train accuracy

tr.accuracy[i] <-length(which(y.tr==tic.train.y))/length(tic.train.y)

# test accuracy

te.accuracy[i] <- length(which(y.te==tic.test.y))/length(tic.test.y)

# sensitivity

sens[i]<-length(which(y.te==1&tic.test.y==1))/length(which(tic.test.y==1))

# specificity

spec[i] <-length(which(y.te==0&tic.test.y==0))/length(which(tic.test.y==0))

# precision

prec[i]<-length(which(y.te==1&tic.test.y==1))/length(which(y.te==1))

# F1-score

F1[i]<-2\*(prec[i]\*sens[i])/(prec[i]+sens[i])

}

prec[1]<-0

F1[1]<-0

# accuracy plot

plot(te.accuracy,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(te.accuracy,tr.accuracy)),ylab="Train/Test Accuracy %",col.lab=rgb(0,0.5,0))

lines(tr.accuracy,type="b",col="green")

title(main="Train and Test Accuracy",col.main="blue",font.main=4)

axis(1,at=1:length(num.split),labels=num.split)

axis(2,at=c(0.90,0.91,0.92,0.93,0.94,0.95),labels=c("90%","91%","92%","93%","94%","95%"))

legend("topleft", c("Test Accuracy","Training Accuracy"), cex=0.8,

col=c("red","green"), lty=1)

# sensitivity,specificity,precision plot

plot(sens,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(sens,spec,prec)),ylab="Sensitivity/Specificity/Precision %",col.lab=rgb(0,0.5,0))

lines(spec,type="b",col="green")

lines(prec,type="b",col="purple")

title(main="Sensitivity/Specificity/Precision",col.main="blue",font.main=4)

axis(1,at=1:length(num.split),labels=num.split)

axis(2,at=seq(from=0.1,to=1,by=0.1),labels=c("10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"))

legend("topleft", c("Sensitivity","Specificity","Precision"), cex=0.6,

col=c("red","green","purple"), lty=1)

# F1 plot

plot(F1,type="b",col="red",xlab="Number of Splits",axes=FALSE,ylim=range(c(F1)),ylab="F1-Score",col.lab=rgb(0,0.5,0))

title(main="F1-Score",col.main="blue",font.main=4)

axis(1,at=1:length(num.split),labels=num.split)

axis(2,at=seq(from=0,to=0.16,by=0.04),labels=seq(from=0,to=0.16,by=0.04))

# Test Set

y <- predict(fit,ticeval2000,type="class")

table(y,tic.test.y)

sensitivity<-length(which(y==1&tic.test.y==1))/length(which(tic.test.y==1))

specificity<-length(which(y==0&tic.test.y==0))/length(which(tic.test.y==0))

precision<-length(which(y==1&tic.test.y==1))/length(which(y==1))

F1<-2\*(precision\*sensitivity)/(precision+sensitivity)

# Pruned tree with nsplit=3

colnames(ticdata2000)[c(47,59,61)] <- c("Contribution car policies","Contribution fire policies","Contribution boat policies")

fit<-rpart(ticdata2000$V86~.,method="class", y=TRUE,control=rpart.control(cp=0,xval=2), parms=list(split="information"),data=ticdata2000)

pruned.tree <- prune(fit,cp=fit$cptable[2,"CP"])

# Plot Pruned Tree

plot(pruned.tree, margin=0.1,uniform=T, branch=1)

text(pruned.tree, use.n=TRUE, all=FALSE, cex=0.8,col="red")

# =============================== AdaBoost ========================================================

library(rpart)

library(ada)

T<-c(1,20,40,60,80,100,500,1000)

tr.accuracy <-rep(0,length(T))

te.accuracy <-rep(0,length(T))

sens <-rep(0,length(T))

spec <-rep(0,length(T))

prec<-rep(0,length(T))

F1 <-rep(0,length(T))

fold1 <- c(1:2911)

fold2 <- c(2912:5822)

data1<-ticdata2000[fold1,]

data2<-ticdata2000[fold2,]

for(t in 1:length(T)){

model1 <- ada(data1$V86~.,data=data1,iter = T[t], loss = "e", type = "discrete")

y1 <-predict(model1,ticdata2000[,1:85])

model2 <- ada(data2$V86~.,data=data2,iter = T[t], loss = "e", type = "discrete")

y2 <-predict(model2,ticdata2000[,1:85])

# train accuracy

tr.a1 <- (length(which(y1[fold1]==1&tic.train.y[fold1]==1))+length(which(y1[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

tr.a2 <- (length(which(y2[fold2]==1&tic.train.y[fold2]==1))+length(which(y2[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

tr.accuracy[t] <-(tr.a1+tr.a2)/2

# test accuracy

te.a1 <- (length(which(y1[fold2]==1&tic.train.y[fold2]==1))+length(which(y1[fold2]==0&tic.train.y[fold2]==0)))/length(fold2)

te.a2 <- (length(which(y2[fold1]==1&tic.train.y[fold1]==1))+length(which(y2[fold1]==0&tic.train.y[fold1]==0)))/length(fold1)

te.accuracy[t] <- (te.a1+te.a2)/2

# sensitivity

sens1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(tic.train.y[fold2]==1))

sens2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(tic.train.y[fold1]==1))

sens[t]<-(sens1+sens2)/2

# specificity

spec1 <- length(which(y1[fold2]==0&tic.train.y[fold2]==0))/length(which(tic.train.y[fold2]==0))

spec2 <- length(which(y2[fold1]==0&tic.train.y[fold1]==0))/length(which(tic.train.y[fold1]==0))

spec[t] <-(spec1+spec2)/2

# precision

prec1 <- length(which(y1[fold2]==1&tic.train.y[fold2]==1))/length(which(y1[fold2]==1))

prec2 <- length(which(y2[fold1]==1&tic.train.y[fold1]==1))/length(which(y2[fold1]==1))

prec[t]<-(prec1+prec2)/2

# F1-score

F1.1<- 2\*(prec1\*sens1)/(prec1+sens1)

F1.2<- 2\*(prec2\*sens2)/(prec2+sens2)

F1[t]<-(F1.1+F1.2)/2

}

# accuracy plot

plot(te.accuracy,type="b",col="red",xlab="Number of Weak Learner",axes=FALSE,ylim=range(c(te.accuracy,tr.accuracy)),ylab="Train/Test Accuracy %",col.lab=rgb(0,0.5,0))

lines(tr.accuracy,type="b",col="green")

title(main="Train and Test Accuracy",col.main="blue",font.main=4)

axis(1,at=1:length(T),labels=T)

axis(2,at=seq(from=0.9,to=1,by=0.02),labels=c("90%","92%","94%","96%","98%","100%"))

legend("topright", c("Test Accuracy","Training Accuracy"), cex=0.8,

col=c("red","green"), lty=1)

# sensitivity,specificity,precision plot

plot(sens,type="b",col="red",xlab="Number of Weak Learner",axes=FALSE,ylim=range(c(sens,spec,prec)),ylab="Sensitivity/Specificity/Precision %",col.lab=rgb(0,0.5,0))

lines(spec,type="b",col="green")

lines(prec,type="b",col="purple")

title(main="Sensitivity/Specificity/Precision",col.main="blue",font.main=4)

axis(1,at=1:length(T),labels=T)

axis(2,at=seq(from=0.1,to=1,by=0.1),labels=c("10%","20%","30%","40%","50%","60%","70%","80%","90%","100%"))

legend("topright", c("Sensitivity","Specificity","Precision"), cex=0.8,

col=c("red","green","purple"), lty=1)

# F1 plot

plot(F1,type="b",col="red",xlab="Number of Weak Learner",axes=FALSE,ylim=range(c(F1)),ylab="F1-Score",col.lab=rgb(0,0.5,0))

title(main="F1-Score",col.main="blue",font.main=4)

axis(1,at=1:length(T),labels=T)

axis(2,at=seq(from=0,to=0.01,by=0.001),labels=seq(from=0,to=0.01,by=0.001))

# Test Set

model <- ada(ticdata2000$V86~.,data=ticdata2000,iter = 500, loss = "e", type = "discrete")

y <-predict(model,ticeval2000)

table(y,tic.test.y)

sensitivity<-length(which(y==1&tic.test.y==1))/length(which(tic.test.y==1))

specificity<-length(which(y==0&tic.test.y==0))/length(which(tic.test.y==0))

precision<-length(which(y==1&tic.test.y==1))/length(which(y==1))

F1<-2\*(precision\*sensitivity)/(precision+sensitivity)