# Chapter 5
# Preparing to Model the Data

# Setup Phase

- Partitioning the Data
- Validating the Data Partition
- Balancing the Data
- Establishing Baseline Model Performance

# Statistical Inference

- A widespread tool for performing estimation and prediction is *statistical inference*.
- Statistical Inference
  - Methods for estimating and testing hypotheses about *population* characteristics based on information contained in a sample
- Population
  - A population is a collection of <u>all</u> elements of interest for a particular study
  - A *parameter* is a characteristic of a population, such as the mean number of customer service calls of all cell phone customers
    - Example: Cell phone company wants actionable results for all their present and future customers (population), not only the 3333 customers for which they gathered the data (sample)
- Sample
  - A sample is a <u>representative subset</u> of the population
  - If the sample characteristics deviate systematically from the population characteristics, *statistical inference should not be applied*
  - A *statistic* is a characteristic of a sample, such as the mean number of customer service calls of the 3333 customers in the sample (1.563).

# Statistical methodology and data mining methodology

- Statistical methods and data mining differ in two ways
  1. Applying statistical inference using the huge sample sizes tends to result in statistical significance, even when the results are of no practical significance
  2. In statistical methodology, the data analyst has an *a priori* hypothesis in mind
     - Data mining procedures usually do not have an *a priori* hypothesis, instead freely trolling through the data for actionable results

# Partitioning the Data

- Unless properly conducted, data mining can return phantom spurious results due to random variation rather than real effects
- This data dredging can be avoided through *cross-validation*
- Cross-validations ensures that results uncovered in an analysis are generalizable to an independent, unseen, data set
- The most common methods are *twofold cross-validation* and *k-fold cross-validation*
- In twofold cross-validation, the data are partitioned, using random assignment, into a *training data set* and a *test data set*
  - The only systematic difference between the training data set and the test data set is that the training data includes the target variable and the test data does not; the training dataset will be *preclassified*
  - For highly complex data sets, more training records would be recommended, such as 75-90% of the original data
  - For smaller or less complex data sets, 50-67% of the original data would be recommended for the training data set

# Partitioning the Data *(cont'd)*

- The training set does not include new/future data
- The algorithm must not memorize and blindly apply patterns from training set into new/future data
  - Example: If all customers named "David" in the training set fall in the same income bracket, we don't want to algorithm to assign income bracket based on the "David" name
  - Such a pattern is a spurious artifact of the training set and needs to be verified before deployment
- The next step is to examine how the data model performs in the test set of the data
  - The target variable of the test set is hidden temporarily, and classification is performed according to the predictor variables only
  - The efficacy of the classification are evaluated by comparing the predicted values against the true values of the target variable
  - The provisional data mining model is adjusted to minimize the error on the test set

# Partitioning the Data in Python

- import pandas as pd
- from sklearn.model_selection import train_test_split
- import random
- bank = pd.read_csv("bank-additional.csv“,sep=“;”)
- #create a training and test data set
- #the test data set is 25% of the original data set
- #random_state sets the seed for the random number generator
- bank_train, bank_test = train_test_split(bank,test_size = 0.25,random_state=7)
- #check if the data set sizes are as expected
- print(bank.shape)
- print(bank_train.shape)
- print(bank_test.shape)

# Partitioning the Data in R

- bank <- read.csv("bank-additional.csv",sep=';')
- set.seed(7);
- n <- dim(bank)[1]
- #runif() randomly draws numbers between 0 and 1,
- #each with equal probability
- #n generates n numbers
- #about 75% will be TRUE
- train_ind <- runif(n) < 0.75
- bank_train <- bank[train_ind,]
- bank_test <- bank[!train_ind,]

# Validating the Partition

- Cross-validation guards against spurious results because it is highly unlikely that the same random variation is found in both the training and test set
- But the data analyst must ensure that the training and test sets are indeed independent, by *validating the partition.*
- Validate the partition into training and test sets by graphical and statistical comparison
    - We might find that a significantly higher proportion of positive values of an important flag variable were assigned to the training set – this assignment would bias the results and hurt the prediction/classification
- The suggested hypothesis test for validating different types of target variables
    - Numeric – two-sample t-test for the difference in means
    - Categorical variable with 2 classes – two-sample Z-test for the difference in proportions
    - Categorical variable with more than 2 classes – test for the homogeneity of proportions

# Two-sample t Test for difference in means

- The test statistic for the difference in population mean is:

$$t_{data} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

which follows an approximate *t* distribution with degrees of freedom the smaller of $n_1 - 1$ and $n_2 - 1$ when both populations are normally distributed or both samples are large

- Example: We divided the bank dataset into a training and a test data set
  - Assess the validity of the partition by testing whether the age mean differs between the two data sets

| Data Set | Sample Mean | Sample Standard Deviation | Sample Size |
|---|---|---|---|
| Training Set | $\bar{x}_1 = 40.07122$ | $s_1 = 10.24584$ | $n_1 = 3103$ |
| Test Set | $\bar{x}_2 = 40.24311$ | $s_2 = 10.52096$ | $n_2 = 1016$ |

# Two-sample t Test for difference in means *(cont'd)*

- The sample means in the table on the previous slide are not too different
- Need to perform hypothesis test to make sure
- Hypothesis is:

$$H_0: \mu_1 = \mu_2 \;\; vs. \;\; H_a: \mu_1 \neq \mu_2$$

- The test statistic is:

$$t_{data} = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} = \frac{40.07122 - 40.24311}{\sqrt{\frac{10.24584^2}{3103} + \frac{10.52096^2}{1016}}} = -0.454901$$

- The two-tailed p-value for $t_{data} = -0.4549$ is:

$$\text{p-value} = 2 \cdot P(t > -0.4549) = 0.649278$$

- p-value is large
- There is no evidence that mean age differs between test and training data sets
- For this variable, the partition seems valid

# Two-sample Z Test for difference in proportions

- Not all variables are numeric
- For a flag variable (like no/yes) we need the two-sample Z test for the difference in proportions

$$Z_{data} = \frac{p_1 - p_2}{\sqrt{p_{pooled} \cdot (1 - p_{pooled})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

where $p_{pooled} = \frac{x_1+x_2}{n_1+n_2}$, and $x_i$ and $p_i$ represents the number of and proportion of records with value 1 (for example) for sample *i*, respectively

- Example: The Training partition resulted in $x_1$=336 of $n_1$=3103 for attribute "y" being "yes", while the Test set has $x_2$=115 of $n_2$=1016
- Therefore, $p_1 = \frac{x_1}{n_1} = \frac{336}{3103} = 0.1083, p_2 = \frac{x_2}{n_2} = \frac{115}{1016} = 0.1131$ and $p_{pooled} = \frac{x_1+x_2}{n_1+n_2} = \frac{336+115}{3103+1016} = 0.1095$
- The hypotheses are:

$$H_0:\ \pi_1 = \pi_2\ \ vs.\ H_a:\ \pi_1 \neq \pi_2$$

# Two-sample Z Test for difference in proportions *(cont'd)*

- The test statistic is:

$$Z_{data} = \frac{p_1 - p_2}{\sqrt{p_{pooled} \cdot (1 - p_{pooled})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} = \frac{0.1083 - 0.1131}{\sqrt{0.1095 \cdot (0.8095)\left(\frac{1}{3103} + \frac{1}{1016}\right)}} = -0.4460$$

- The p-value is:

$$\text{p}-\text{value} = 2 \cdot P(Z > -0.4460) = 0.6556$$

- There is no evidence that the proportion of value "y" for attribute "y" differs between the training and test data sets.
- For this variable, the partition is valid

# Test for the homogeneity of proportions

- *Multinomial data* is an extension of binomial data to $k > 2$ categories
  - Example multinomial variable: *marital* can be *married*, *single*, *divorced*, *unknown*
  - The table below shows a training set of 3103 people and test set of 1016 people
- Test for the homogeneity of proportions
  - To determine whether significant differences exist between multimodal proportions
- Hypotheses are:

$$H_0: p_{married,training} = p_{married,test},$$
$$p_{single,training} = p_{single,test},$$
$$p_{divorced,training} = p_{divorced,test}$$
$$p_{unknown,training} = p_{unknown,test}$$

$H_a$: At least one of the claims in $H_0$ is wrong.

| Data Set | Married | Single | Divorced | Unknown | Total |
|---|---|---|---|---|---|
| Training set | 1903 | 853 | 338 | 9 | 3103 |
| Test set | 606 | 300 | 108 | 2 | 1016 |
| Total | 2509 | 1153 | 446 | 11 | 4119 |

# Test for the homogeneity of proportions *(cont'd)*

- Compare observed frequencies against expected frequencies if $H_0$ were true
- Example:
  - A. Find overall proportion of married people in whole dataset (training+test sets): $^{2509}/_{4119}$
  - B. Multiply this overall proportion by the number of people in training set, 3103, yields the expected proportion of married people in the training set to be:

$$\text{Expected frequency}_{married,training} = \frac{(3103)(2509)}{4119} = 1890$$

- Step A above uses the overall proportion because $H_0$ states that both partitions are equal

# Test for the homogeneity of proportions *(cont'd)*

- Generalizing, the expected frequencies are calculated as follows:

$$\text{Expected frequency} = \frac{(\text{row total})(\text{column total})}{\text{grand total}}$$

- Applying this formula yields the expected frequencies in the table below
- Observed frequencies (O) and expected frequencies (E) are compared using test statistics from the $\chi^2_{data}$ (chi-square distribution:

$$\chi^2_{data} = \sum \frac{(O - E)^2}{E}$$

| Data Set | Married | Single | Divorced | Unknown | Total |
|---|---|---|---|---|---|
| Training set | 1890 | 869 | 336 | 8 | 3103 |
| Test set | 619 | 284 | 110 | 3 | 1016 |
| Total | 2509 | 1153 | 446 | 11 | 4119 |

# Test for the homogeneity of proportions *(cont'd)*

- Large differences between observed and expected frequencies, and large value for $\chi^2_{data}$, leads to small p-value, and rejection of null hypothesis
- The table (next slide) illustrates how the test statistic is calculated
- The p-value is the area to the right of $\chi^2_{data}$ under the $\chi^2$ curve with degrees of freedom equal to (number of rows – 1)(number of columns – 1) = (1)(3) = 3:

$$p - value = P\left(\chi^2 > \chi^2_{data}\right) = P(\chi^2 > 2.06) = 0.5600$$

- Because this p-value is large, there is no evidence that the frequencies significantly differ between the training and the test data sets
- The partition is valid

$\frac{(Obs - Exp)^2}{Exp}$

$\chi^2_{data} = 4.15$

# Test for the homogeneity of proportions *(cont'd)*

| Cell | | Observed Frequency | Expected Frequency | $\frac{(Obs - Exp)^2}{Exp}$ |
|---|---|---|---|---|
| Married | Training | 1903 | 1890 | $\frac{(1903 - 1890)^2}{1890} = 0.09$ |
| Married | Test | 606 | 619 | $\frac{(606 - 619)^2}{619} = 0.27$ |
| Single | Training | 853 | 869 | $\frac{(853 - 869)^2}{869} = 0.29$ |
| Single | Test | 300 | 284 | $\frac{(300 - 284)^2}{284} = 0.90$ |
| Divorced | Training | 338 | 336 | $\frac{(338 - 336)^2}{336} = 0.01$ |
| Divorced | Test | 108 | 110 | $\frac{(108 - 110)^2}{110} = 0.04$ |
| Unknown | Training | 9 | 8 | $\frac{(9 - 8)^2}{8} = 0.13$ |
| Unknown | Test | 2 | 3 | $\frac{(2 - 3)^2}{3} = 0.33$ |
| | | | | $\chi^2_{data}$=2.06 |

# Validating the Partition in R

- #two-sample t-test
- t.test(bank_train$age,bank_test$age)

- #two-sample z-test
- p1<-sum(bank_train$y=="yes")/dim(bank_train)[1]
- p2<-sum(bank_test$y=="yes")/dim(bank_test)[1]
- p_pooled<-(sum(bank_train$y=="yes")+sum(bank_test$y=="yes"))/
- (dim(bank_train)[1]+dim(bank_test)[1])
- z<-(p1-p2)/sqrt(p_pooled *(1-p_pooled) *(1/dim(bank_train)[1]+1/dim(bank_test)[1]))

# Validating the Partition in R

- #multinomial test
- Observed=matrix(c(1903,853,338,9,606,300,108,2),nrow=2,byrow=TRUE)
- Expected=matrix(c(1890,869,336,8,619,284,110,3),nrow=2,byrow=TRUE)
- chi.sq=sum((Observed-Expected)^2/Expected)
- 1-pchisq(chi.sq,3)

# Cross-validation *(cont'd)*

- In k-fold cross validation, the data is partitioned into k subsets or folds
- The model is then built using the data from k – 1 subsets, using the $k^{th}$ fold as the test set
- This is repeated k times, always using a different fold as the test subset, until we have k different models
- The results from the k models are then combined using averaging or voting
- A popular choice for k is 10
- A benefit of using k-fold cross-validation is that each record appears in the test set exactly once; a drawback is that the requisite validation task is made more difficult.

# Cross-validation *(cont'd)*

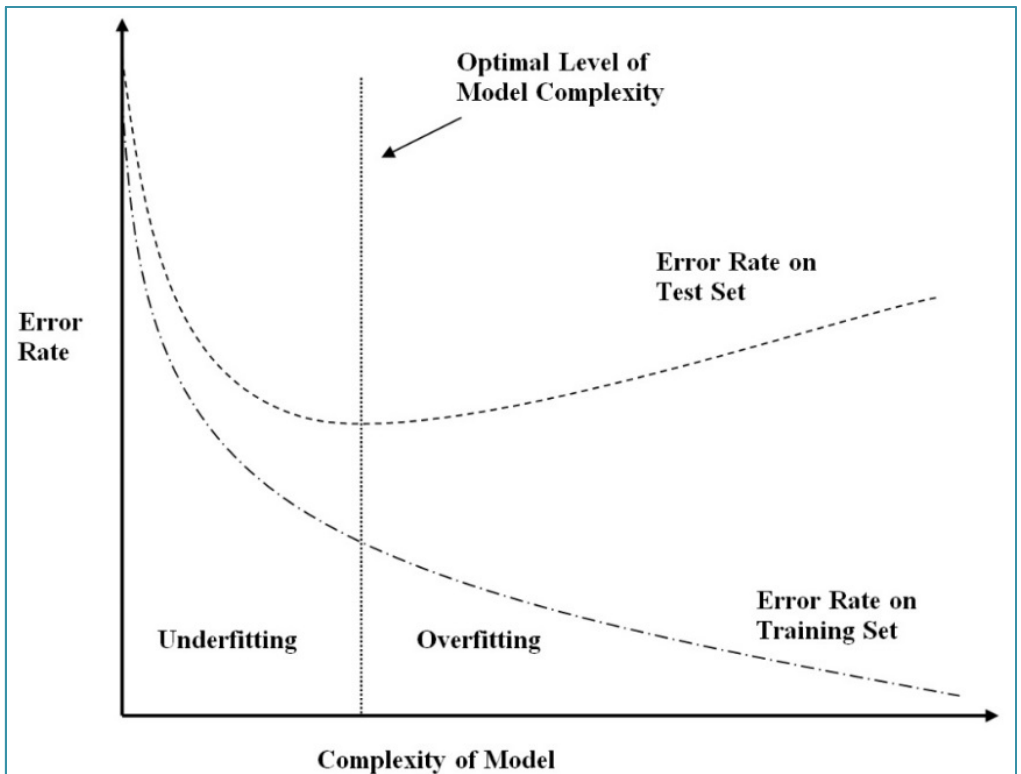
- In summary
  1. *Partition* the available data into a *training set* and a *test set*. Validate the partition.
  2. Build a data mining model using the training set data.
  3. Evaluate the data mining model using the test set data.

# Overfitting

- Usually, the accuracy of model is not as high on the test set as it is on the training set
  - This might be caused by *overfitting* on the training set
- Overfitting occurs when the model tries to fit every possible trend/structure in the training set
- There is a need to balance the model complexity (resulting in high accuracy in training set) and generalizability to the test/validation sets
  - Increasing complexity leads to degradation of generalizability of the model to the test set, as shown in the graph (next slide)
- Per the graph, as the model begins to grow in complexity, the error rates for both training sets start to fall
- As the model complexity increases, the error in the training set continues to fall as the error in the test set starts to increase
  - The model has memorized the training set rather than leaving room for generalization to unseen data

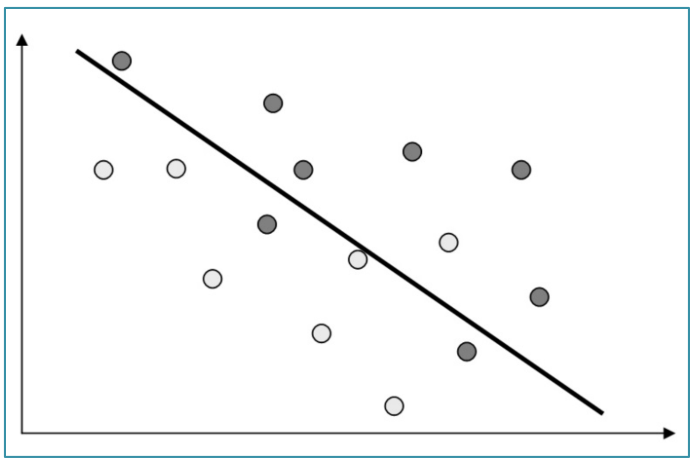
# Overfitting *(cont'd)*

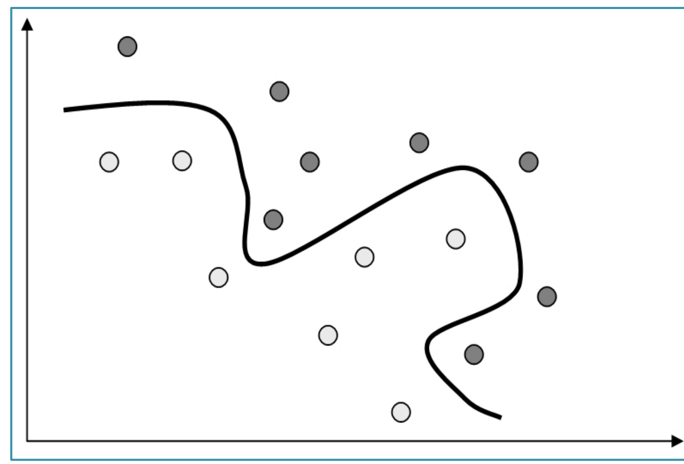
- The optimal model complexity is the point where the minimal error rate on the test set is located
- Complexity greater than the optimal is considered overfitting
- Less than the optimal is considered underfitting



# Bias-Variance trade-off

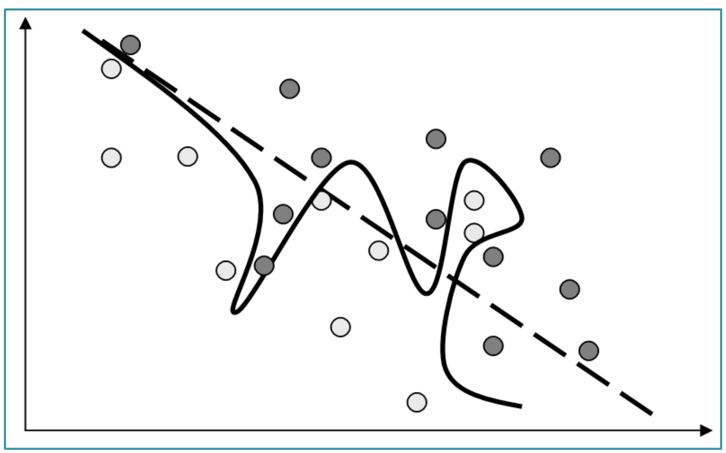
- Example: Building the optimal curve (or line) that separates dark gray points from light gray points in the left figure
  - A straight line has low complexity, but some classification errors
- In the right figure, we have reduced the error to zero, but at the cost of complexity
- One might be tempted to use the more accurate model
  - However, we should be careful not to depend on the idiosyncrasies of the training set





# Bias-Variance trade-off (*cont'd*)

- Suppose that we add more data points to the scatter plot, as per the figure
- In this case, the low-complexity separator need not change much
  - Meaning that this separator has *low variance*
- But the high-complexity separator, the curvy line, must alter considerably to maintain its low error rate
  - This high degree of change indicates that this separator has *high variance*



# Bias-Variance trade-off *(cont'd)*

- Even though the high-complexity separator has *low bias* (low error rate on the training set), it has *high variance*
- And, even though the low-complexity model has a high *bias*, it has low *variance*
- Known as the *bias-variance trade-off*
  - Another way of describing the overfitting/underfitting dilemma from the prior section
  - As model complexity increases, the bias of the training set decreases, but the variance increases
  - The goal is to construct a model in which neither bias nor variance is too high

# Balancing the training data

- Balancing is recommended on classification models when one target variable class has a much lower frequency than the other classes
  - The algorithm has a chance to learn about all types of records, not just those with high frequency
- Example: In a credit card fraud classification model with 100,000 transactions, only 1000 are fraudulent
  - The model could achieve 99% accuracy by labeling all transactions as "non-fraudulent" – this behavior is not desired
  - Instead, balance should be performed so that relative frequency of fraudulent transactions is increased
- There are two ways of doing balancing
  1. Resample a number of fraudulent records
  2. Set aside a number of non-fraudulent records

# Balancing the training data *(cont'd)*

- *Resampling* refers to sampling at random with replacement from a data set
- Example: Use Resampling so that the fraudulent records represent 25% of the balanced training set, rather than 1%
  - Solution: Add 32,000 resampled fraudulent records so that we have 33,000 fraudulent records, out of a total of 100,000+32,000=132,000 records in all

$$\frac{33{,}000}{132{,}000} = 0.25 = 25\% \text{ desired records}$$

- The formula to determine the number of records to resample is

$$x = \frac{p(records) - rare}{1 - p}$$

where:

***x*** is the required number of resampled records

***p*** is the desired proportion of rare values in the balanced data set,

***records*** represents the number of records in the unbalanced data set, and

***rare*** represents the current number of rare target values

# Balancing the training data *(cont'd)*

- Set aside a number of non-fraudulent records
  - When resampling is not desired, a number of non-fraudulent records would be set aside instead
- To achieve a 25% balance proportion, we would retain only 3000 non-fraudulent records
- We would need to discard 96,000 out of 99,000 non-fraudulent records from the analysis
  - It would not be surprising if the model suffered from such a large deletion of data
- When choosing a desired balancing proportion, recall the rationale for doing so: in order to allow the model a sufficiently rich variety of records to learn how to classify the rarer value of the target variable across a range of situations
  - The balancing proportion can be lower if the analyst is confident that the rare target value exposes a rich variety of records
  - The balancing proportion should be higher if the analyst is not confident

# Balancing the training data *(cont'd)*

- **The test data set should never be balanced**
  - The test data set represents new data that the model has not seen yet
  - Real work data is unbalanced, therefore, the test data set should not be balanced either
  - All model evaluation will take place using the test data set, so that evaluative measures will be applied to unbalanced data

# Balancing the Data in Python

- #balancing the data - increase "yes" to 30%
- bank_train['y'].value_counts()
- x=(0.3*3089-338)/0.7
- print(x)
- to_resample = bank_train.loc[bank_train['y']=="yes"]
- our_resample = to_resample.sample(n=841,replace=True)
- bank_train_rebal = pd.concat([bank_train,our_resample])
- bank_train_rebal['y'].value_counts()

# Balancing the Data in R

- #balancing the data - increase "yes" to 30%
- table(bank_train$y)
- x<-(.3*3103-336)/.7
- to.resample<-which(bank_train$y=="yes")
- our.resample<-sample(x=to.resample,size=850,replace=TRUE)
- our.resample.records<-bank_train[our.resample,]
- train_bank_rebal<-rbind(bank_train,our.resample.records)
- t.v1<-table(train_bank_rebal$y)
- t.v2<-rbind(t.v1,round(prop.table(t.v1),4))
- colnames(t.v2) <- c("y=no","y=yes")
- rownames(t.v2) <- c("count","proportion")
- t.v2

# Establishing baseline performance

- Without a baseline, it is not possible to determine whether our results are any good
- Example: Suppose that we naively report that "only" 28.4% of the customers adopting the International Plan will churn
  - It does not sound too bad, until we notice that the overall churn rate is only 14.49%
  - This overall churn rate may be considered our *baseline*, against which any further results can be calibrated
  - Thus, belonging to the International Plan nearly doubles the churn rate – Not good!

# Establishing baseline performance *(cont'd)*

- The type of baseline to use depends on the way the results are reported
  - Binary Classification
    - All Positive Model – Accuracy p
    - All Negative Model – Accuracy 1-p
    - Accuracy to beat is max(p,1-p)
    - Example Credit Card Fraud: "non-fraudulent" at 99%, fraudulent at 1% (model should have at least 99% accuracy)
  - K-nary Classification
    - Let $P_i$ represent the proportion of class $C_i$
    - Accuracy to beat is the largest $P_i$
    - Example Marital: "married" at 40%, "single" at 30%, "divorced" at 20%, "unknown" at 10% (model should have at least 40% accuracy identifying "married")
- Another Example: Suppose the data mining model resulted in a predicted churn rate of 9.99%
  - This represents a 14.49%-9.99%=4.5% absolute decrease in the churn rate
  - But also a 4.5%/14.49%=31% in relative decrease in the churn rate
  - The analyst should make it clear for the client which comparison method is being used

# Establishing baseline performance *(cont'd)*

- In an estimation task using a regression model, our baseline may take the form of a "$\bar{y}$ model"
  - $\bar{y}$ model - The model simply finds the mean of the response variable, and predicts that value for every record
- No data mining model should have a problem beating this $\bar{y}$ model
  - If your data mining model cannot outperform the $\bar{y}$ model, then something is clearly wrong
  - We measure the goodness of a regression model using the standard error of the estimate $\boldsymbol{s}$
- A more challenging baseline would be using results already existing in the field
  - If the current algorithm your company uses succeeds identifying 90% of all fraudulent transactions, then your model would be expected to outperform this 90%