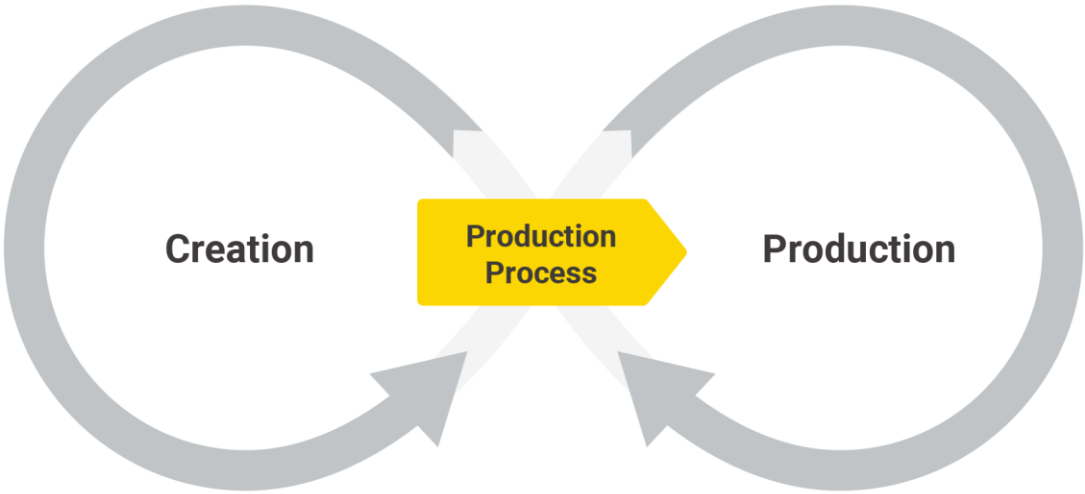
Technical Collections

MAARIT **WIDMANN** ALFREDO **ROCCATO**

Scoring Metrics

Evaluating Machine Learning Models

KNIME v5.1



Open for Innovation
KNIME

Copyright © 2023 by KNIME Press

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording or likewise.

This book has been updated for **KNIME 5.1**.

For information regarding permissions and sales, write to:

KNIME Press
Talacker 50
8001 Zurich
Switzerland

knimepress@knime.com

www.knime.com

Preface

Machine learning models can automate different kinds of processes → prove customer credit worthiness, flag emails as spam, detect fraudulent transactions, forecast weather, optimize the electricity supply, and more! The overarching goal of all these applications is to have accurate predictions. But how do we define “accurate”? We can quantify the accuracy from different perspectives with scoring metrics, such as overall accuracy, area under the ROC curve, sensitivity and specificity.

In general, scoring metrics have two functions in the model building process: First, they provide objective values to optimize model parameters and compare models. And second, they tell the expected accuracy of the model in production data. Careful data preparation, model selection and configuration can make a difference between the first - accuracy of validation data - and the second- expected accuracy of production data. So, it is worth to run a few rounds in the data science creation cycle to then be able to move happily into the production cycle!

In this booklet, we provide an overview of scoring metrics to evaluate a classification and regression model. We begin with an introduction to the confusion matrix and class statistics, numeric scoring metrics and visual scoring techniques. We then take a look at a special situation that occurs frequently in practice, namely, imbalanced target classes. We apply resampling to imbalanced data and correct the prediction results for bias, we explain why Cohen’s kappa works better than overall accuracy for imbalanced data, and we investigate why resampling might fail for highly imbalanced data. Finally, we take a look at a few examples of interpreting the model results. Here, we check how much profit a credit scoring model generates in terms of money and how we can interpret the coefficients of a logistic regression model as percentage effects.

We wish you insightful reading!

Maarit Widmann & Alfredo Roccato

Table of Contents

CONFUSION MATRIX AND CLASS STATISTICS 1

EMAIL CLASSIFICATION: SPAM VS. USEFUL 1
CONFUSION MATRIX 3
MEASURES FOR CLASS STATISTICS 3
MULTIVARIATE CLASSIFICATION MODEL 5
SUMMARY 7

NUMERIC SCORING METRICS 8

QUANTITATIVE DATA HAVE ENDLESS STORIES TO TELL! 8
WHY ARE NUMERIC SCORING METRICS NEEDED? 8
FIVE METRICS – FIVE DIFFERENT PERSPECTIVES ON PREDICTION ACCURACY 9
A REVIEW OF THE FIVE NUMERIC SCORING METRICS 13
SUMMARY 14

VISUAL SCORING TECHNIQUES FOR CLASSIFICATION MODELS 15

USE CASE: CHURN PREDICTION MODEL 15
COMPARING PERFORMANCES ACROSS CLASSIFICATION THRESHOLDS 16
COMPARING MULTIPLE MODELS 19
SAVING RESOURCES WITH A MODEL 19
VISUAL MODEL EVALUATION TECHNIQUES – SUMMARY 22
TIPS AND TRICKS 23

CORRECTING PREDICTED CLASS PROBABILITIES IN IMBALANCED DATASETS 25

CLASSIFICATION ON IMBALANCED DATASETS 25
RESAMPLING TO BALANCE DATASETS 26
CORRECTING PREDICTED CLASS PROBABILITIES 26
EXAMPLE: FRAUD DETECTION 27
SUMMARY 31

COHEN'S KAPPA 33

AN ALTERNATIVE FOR WHEN OVERALL ACCURACY IS BIASED, YET NOT TRUSTING THE STATISTICS BLINDLY 33
MEASURE PERFORMANCE IMPROVEMENT ON IMBALANCED DATASETS 34
COHEN'S KAPPA 36
PAIN POINTS OF COHEN'S KAPPA 36
SUMMARY 39
EXAMPLE WORKFLOW: COHEN'S KAPPA FOR EVALUATING CLASSIFICATION MODELS 40

RESAMPLING IMBALANCED DATA LIMITS 41

BUILDING A CLASSIFICATION MODEL FOR FRAUD DETECTION 42
RESAMPLING TECHNIQUES 43
EFFECTS OF RESAMPLING ON FRAUD DETECTION PERFORMANCE 44
DEMONSTRATING OVER- AND UNDERFITTING IN FRAUD DETECTION 45
DIAGNOSING PROBLEMS OF RESAMPLING IN FRAUD DETECTION 47
CONCLUSIONS 48

FINDING AN OPTIMAL CLASSIFICATION THRESHOLD BASED ON COST AND PROFIT 50

PENALIZING AND REWARDING CLASSIFICATION RESULTS WITH A PROFIT MATRIX 50
FROM MODEL ACCURACY TO EXPECTED PROFIT 51
RESULTS 55

EASY INTERPRETATION OF A LOGISTIC REGRESSION MODEL WITH DELTA-P STATISTICS 57

KEY TAKEAWAYS 57
ASSESSING THE EFFECT OF A SINGLE PREDICTOR WITH THE DELTA-P STATISTICS 58
CONCLUSIONS 63

TOPIC INDEX 64

Confusion Matrix and Class Statistics

Wheeling like a Hamster in the Data Science Cycle

Author: Maarit Widmann

Workflow on KNIME Community Hub: Evaluating Classification Model Performance

Model evaluation – or model scoring - is an important part of a data science project and it's exactly this part that quantifies how good your model is, how much it has improved from the previous version, how much better it is than your colleague's model, and how much room for improvement there still is.

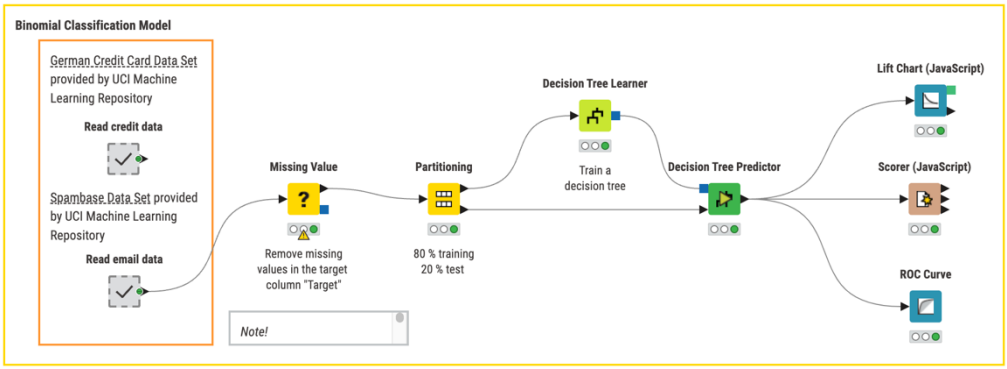
In this article we talk about the confusion matrix — a compact representation of the model performance, and the source of many scoring metrics for classification models.

A classification model predicts two or more known classes, for example, customer churn/no churn, spam/normal email, red/white wine, or malignant/benign tumor. The confusion matrix shows the distribution of actual and predicted classes, and it is the starting point in evaluating a classification model of any nature. The classes can be equally important, for example, when classifying wines to red and white. Or, predicting one class correctly can be more important, for example, when trying to find the malignant tumors. In the latter case we're often dealing with imbalanced data and trying to predict the minority class correctly.

Email classification: Spam vs. Useful

Let's take the case of the email classification problem. The goal is to classify incoming emails in two classes: spam vs. useful ("normal") email. For that, we use the Spambase Data Set provided by UCI Machine Learning Repository. This dataset contains 4601 emails described through 57 features, such as text length and presence of specific words like "buy", "subscribe", and "win". The "Spam" column provides two possible labels for the emails: "spam" (1) and "normal" (0).

The figure below shows a workflow that covers the steps to build a classification model: reading and preprocessing the data, partitioning into a training set and a test set, training the model, making predictions by the model, and evaluating the prediction results.



This workflow, [Evaluating Classification Model Performance](), is building, applying, and evaluating a supervised classification model that classifies incoming emails as spam or useful. Download it from the KNIME Community Hub.

The last step, model scoring, is based on comparing the actual and predicted target column values in the test set. The whole scoring process of a model consists of a match count: how many data rows have been correctly classified and how many data rows have been incorrectly classified by the model. These counts are summarized in the confusion matrix and class statistics and displayed in the interactive view of the Scorer (JavaScript) node shown in the figure below:

Confusion Matrix

Scorer View

Confusion Matrix

	1 (Predicted)	0 (Predicted)	
1 (Actual)	320	43	0.882
0 (Actual)	20	538	0.964
	0.941	0.926	

Class Statistics

Class	True Positives	False Positives	True Negatives	False Negatives	Recall	Precision	Sensitivity	Specificity	F-measure
1	320	20	538	43	0.882	0.941	0.882	0.964	0.910
0	538	43	320	20	0.964	0.926	0.964	0.882	0.945

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
0.932	0.068	0.855	858	63

Settings

Reset Apply Close

Confusion matrix and class statistics in the interactive view of the Scorer (JavaScript) node.

By looking at the confusion matrix we can see the performance of the model in absolute numbers, for example, how many of the actual spam emails were predicted as spam. On the other hand, a number of class statistics and overall accuracy statistics, which show the performance of the model as relative measures, are calculated based on the numbers in the confusion matrix.

Let's see now exactly what these numbers in a confusion matrix are.

Confusion Matrix

The confusion matrix was initially introduced to evaluate results from binomial classification. Thus, the first thing to do is to take one of the two classes as the class of interest, i.e., the **positive class**. In the target column, we need to choose (arbitrarily) one value as the positive class. The other value is then automatically considered the negative class. Keep in mind that the class statistics will show different values if we change the positive class. Here we chose the spam emails as the positive class and the normal emails as the negative class, which produces the confusion matrix shown in the following table:

# of rows: 921	Spam (Predicted)	Normal (Predicted)
Spam (Actual)	320	43
Normal (Actual)	20	538

A confusion matrix showing actual and predicted positive and negative classes in the test set.

The confusion matrix reports the count of:

- Spam emails classified correctly as spam (the positive class). These are called **True Positives (TP)**. The number of true positives is placed in the top left cell of the confusion matrix.
- Spam emails classified incorrectly as normal (the negative class). These are called **False Negatives (FN)**. The number of false negatives is placed in the top right cell of the confusion matrix.
- Normal emails classified incorrectly as spam. These are called **False Positives (FP)**. The number of false positives is placed in the lower left cell of the confusion matrix.
- Normal emails classified correctly as normal. These are called **True Negatives (TN)**. The number of true negatives is placed in the lower right cell of the confusion matrix.

Therefore, the correct predictions are on the diagonal with a gray background; the incorrect predictions are on the diagonal with a white background.

In the next section, using the four counts in the confusion matrix, we can calculate a few class statistics measures to quantify the model performance.

Measures for Class Statistics

The class statistics, as the name implies, summarizes the model performance for the positive and negative classes separately. This is the reason why their values and interpretation change with a different definition of the positive class and why they are

often expressed in pairs: sensitivity & specificity and recall & precision. These pairs of statistics provide a more comprehensive view of the model's performance.

Notice that both pairs of statistics are characterized by an inverse relationship: improving one often happens with the cost of reducing the other. For example, if we use a stricter spam filter, we'll reduce the number of dangerous emails in the inbox but increase the number of normal emails that have to be collected from the spam box folder afterwards.

Sensitivity and Specificity

The table below shows the formulas to calculate sensitivity and specificity based on the counts in the confusion matrix:

# of rows: 921	Spam (Predicted)	Normal (Predicted)	
Spam (Actual)	320	43	$Sensitivity = \frac{320}{320+43} = 0.882$
Normal (Actual)	20	538	$Specificity = \frac{538}{20+538} = 0.879$

Sensitivity and specificity values and their formulas, which are based on the counts in the confusion matrix.

Sensitivity measures the model's prediction performance for the positive class. So, given that spam emails are the positive class, sensitivity quantifies which proportion of the actual spam emails are correctly predicted as spam.

$$Sensitivity = \frac{TP}{TP+FN} = \frac{320}{320+43} = 0.882$$

We divide the number of true positives by the number of all positive events in the dataset: the positive class events predicted correctly (TP) and the positive class events predicted incorrectly (FN). The model in this example reaches the sensitivity value of 0.882. This means that about 88% of the spam emails in the dataset were correctly predicted as spam.

Specificity measures the model's prediction performance for the negative class, so which proportion of the actual normal emails are correctly predicted as normal.

$$Specificity = \frac{TN}{FP+TN} = \frac{538}{20+538} = 0.964$$

We divide the number of true negatives by the number of all negative events in the dataset: the negative class events predicted incorrectly (FP), and the negative class events predicted correctly (TN). The model reaches the specificity value of 0.964, so less than 4% of all normal emails are predicted incorrectly as spam.

Recall, Precision, and F-Measure

The following table shows the formulas to calculate recall and precision based on the counts in the confusion matrix:

# of rows: 921	Spam (Predicted)	Normal (Predicted)	
Spam (Actual)	320	43	$Recall = \frac{320}{320+43} = 0.882$
Normal (Actual)	20	538	
	$Precision = \frac{320}{320+20} = 0.941$		

Recall and precision values and their formulas, which are based on the counts in the confusion matrix.

Similarly to sensitivity, **recall** measures the prediction performance for the positive class. Therefore, the formula for recall is the same as for sensitivity.

$$Recall = \frac{TP}{TP+FN} = \frac{320}{320+43} = 0.882$$

Precision measures the prediction performance of the positive class. That is, which proportion of the predicted spam emails are actually spam emails.

$$Precision = \frac{TP}{TP+FP} = \frac{320}{320+20} = 0.941$$

We divide the number of true positives by the number of all events assigned to the positive class, i.e., the sum of true positives and false positives. The precision value for the model is 0.941. Therefore, almost 95% of the emails predicted as spam were actually spam emails.

Recall and precision can also be reported by one measure that combines them. One example is called F-measure, which is the harmonic mean of recall and precision:

$$F-measure = 2 * \frac{recall*precision}{recall+precision} = 2 * \frac{0.882*0.941}{0.882+0.941} = 0.910$$

In the next section, we introduce the confusion matrix for a multinomial classification model.

Multivariate Classification Model

In case of a multinomial classification model, the target column has three or more values. The emails could be labeled as “spam”, “ad”, and “normal”, for example.

Similarly to a binomial classification model, the target class values are assigned to the positive and the negative class. The difference is that multiple classes must be assigned the same label, positive or negative. Here we define spam as the positive class and the normal and ad emails as the negative class. Now, the confusion matrix looks as shown in the table below:

# of rows: 921	Spam (Predicted)	Ad (Predicted)	Normal (Predicted)
Spam (Actual)	TRUE POSITIVES 27	286	FALSE NEGATIVES 40
Ad (Actual)	1	37	9
Normal (Actual)	FALSE POSITIVES 5	TRUE NEGATIVES 16	500

Confusion matrix showing the distribution of predictions to true positives, false negatives, false positives, and true negatives for a multinomial classification model (3 classes).

To calculate the class statistics, we have to re-define the true positives, false negatives, false positives, and true negatives using the values in a multivariate confusion matrix:

- The cell identified by the row and column for the positive class contains the **True Positives**, i.e., where the actual and predicted class is spam.
- Cells identified by the row for the positive class and columns for the negative class contain the **False Negatives**, where the actual class is spam, and the predicted class is normal or ad.
- Cells identified by rows for the negative class and the column for the positive class contain the **False Positives**, where the actual class is normal or ad, and the predicted class is spam.
- Cells outside the row and column for the positive class contain the **True Negatives**, where the actual class is ad or normal, and the predicted class is ad or normal. An incorrect prediction inside the negative class is still considered as a true negative.

Now, these four statistics can be used to calculate class statistics using the formulas introduced in the previous section.

Summary

In this article, we've shown how to evaluate a classification model with the confusion matrix and class statistics. The confusion matrix lays the first stone in the evaluation of a classification model by showing the counts of correct and incorrect predictions into the target classes. The class statistics, such as sensitivity and specificity, recall and precision, and the F-measure, are calculated based on these counts.

Confusion matrix and class statistics have been defined for binomial classification problems. However, we have shown how they can be easily extended to address multinomial classification problems.

Numeric Scoring Metrics

Find the Right Metric for a Prediction Model

Author: Maarit Widmann

Workflows on KNIME Community Hub: Evaluating the Performance of a Regression Model and Forecasting and Reconstructing Time Series

Quantitative Data have endless Stories to tell!

Daily closing prices tell us about the dynamics of the stock market, small smart meters about the energy consumption of households, smartwatches about what's going on in the human body during an exercise, and surveys about some people's self-estimation of a topic at some point in time. Different types of experts can tell these stories: financial analysts, data scientists, sports scientists, sociologists, psychologists and so on. Their stories are based on models, for example, regression models, time series models and ANOVA models.

Why are Numeric Scoring Metrics needed?

These models have many consequences in the real world, from the decisions of the portfolio managers to the pricing of electricity at different times of the day, week and year. Numeric scoring metrics are needed in order to:

- Select the most accurate model
- Estimate the real-world impact of the error of the model

In this article, we will describe five real-world use cases of numeric prediction models, and in each use case, we measure the prediction accuracy from a slightly different point of view. In one case, we measure if a model has a systematic bias, and in another, we measure a model's explanation power. The article concludes with a review of the numeric scoring metrics, showing the formulas to calculate them, and a summary of their properties. We'll also link to a few example implementations of building and evaluating a prediction model in KNIME Analytics Platform.

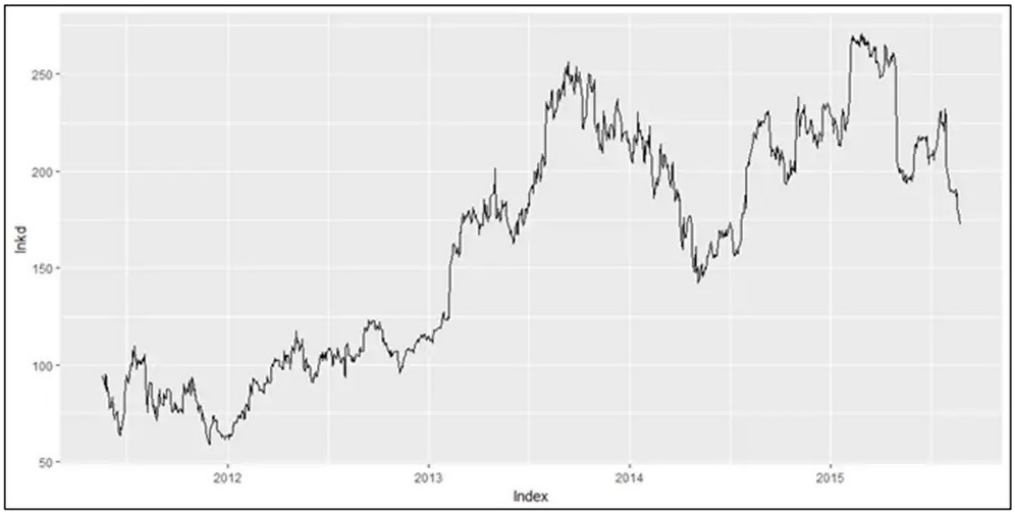
Five Metrics – Five different Perspectives on Prediction Accuracy

(Root) Mean Squared Error, (R)MSE

Which model best captures the rapid changes in the volatile stock market?

In the figure below, you see the development of the LinkedIn closing price from 2011 to 2016. Within the time period, the behavior includes sudden peaks, sudden lows, longer periods of increasing and decreasing value, and a few stable periods. Forecasting this kind of volatile behavior is challenging, especially in the long term. However, for the stakeholders of LinkedIn, it's valuable. Therefore, we prefer a forecasting model that captures the sudden changes to a model that performs well on average over the period of five years.

We select the model with the lowest (root) mean squared error because this metric weights big errors more compared to small errors and favors a model that can react to short-term changes and save the stakeholders' money.

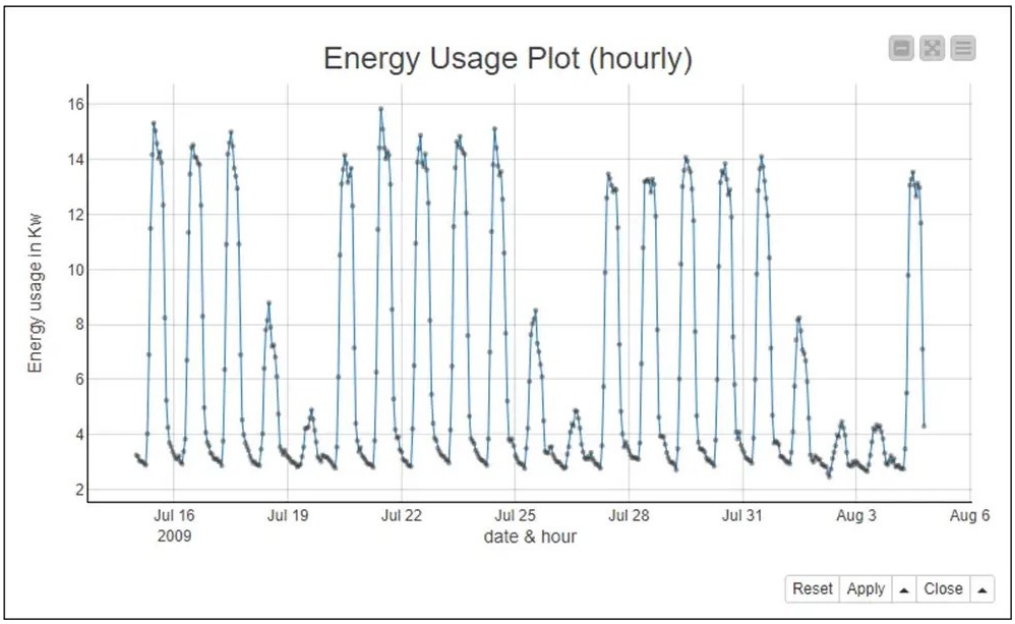


LinkedIn daily stock market closing price from 2011 to 2016: data with few regular patterns and many sudden changes with low forecastability. We select the forecasting model with the lowest (root) mean squared error because it weights the big forecast errors more and favors a model that can capture the sudden peaks and lows.

Mean Absolute Error, MAE

Which model best estimates the energy consumption in the long term?

In the following figure, you can see the hourly energy consumption values in July 2009 in Dublin, collected from a cluster of households and industries. The energy consumption shows a relatively regular pattern, with higher values during working hours and on weekdays and lower values at night and during weekends. This kind of a regular behavior can be forecasted relatively accurately, allowing for long-term planning of the energy supply. Therefore, we select a forecasting model with the lowest mean absolute error. We do this because it weights big and small errors equally, is therefore robust to outliers, and shows which model has the highest forecast accuracy over the whole time period.



Hourly energy consumption values in June 2009 in Dublin, collected from a cluster of households and industries. The data shows a relatively regular behavior and can therefore be forecasted in the long term. We select the forecasting model with the lowest mean absolute error because this metric is robust to outliers.

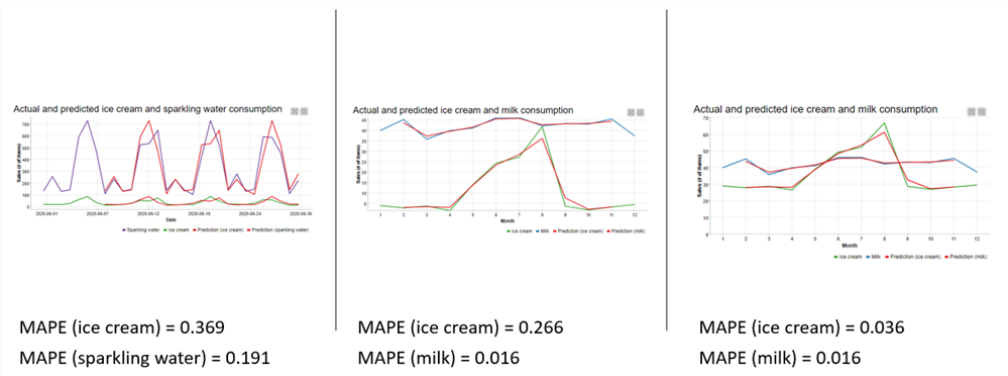
Mean Absolute Percentage Error, MAPE

Are the sales forecasting models for different products equally accurate?

On a hot summer day, the supply of both sparkling water and ice cream should be guaranteed! We want to check if the two forecasting models that predict the sales of these two products are equally accurate.

Both models generate forecasts in the same unit, the number of sold items, but at a different scale since sparkling water is sold in much larger volumes than ice cream. In this kind of a case, we need a relative error metric and use mean absolute percentage error, which reports the error relative to the actual value. The line plot on the left in the figure below shows the sales of sparkling water (purple line) and the sales of ice cream (green line) in June 2020 as well as the predicted sales of both products (red lines). The prediction line seems to deviate slightly more for sparkling water than for ice cream. However, the larger actual values of sparkling water bias the visible comparison. Actually, the forecasting model performs better for sparkling water than for ice cream, as reported by the MAPE values 0.191 for sparkling water and 0.369 for ice cream.

Notice, though, that MAPE values can be biased when the actual values are close to zero. For example, the sales of ice cream are relatively low during the winter months compared to summer months, whereas sales of milk remain pretty constant through the entire year. When we compare the accuracies of the forecasting models for milk vs. ice cream by their MAPE values, the small values in the ice cream sales make the forecasting model for ice cream look unreasonably bad compared to the forecasting model for milk.



Three line plots showing actual and predicted values of ice cream and sparkling water (line plot on the left) and ice cream and milk (line plots in the middle and on the right). In the line plot on the right, the ice cream sales values are scaled up by 25 in order to avoid the bias in mean absolute percentage error introduced by small actual values.

In the line plot in the middle, you see the sales of milk (blue line) and ice cream (green line) and the predicted sales of both products (red lines). If we take a look at the MAPE values, the forecasting accuracy is apparently much better for milk (MAPE = 0.016)

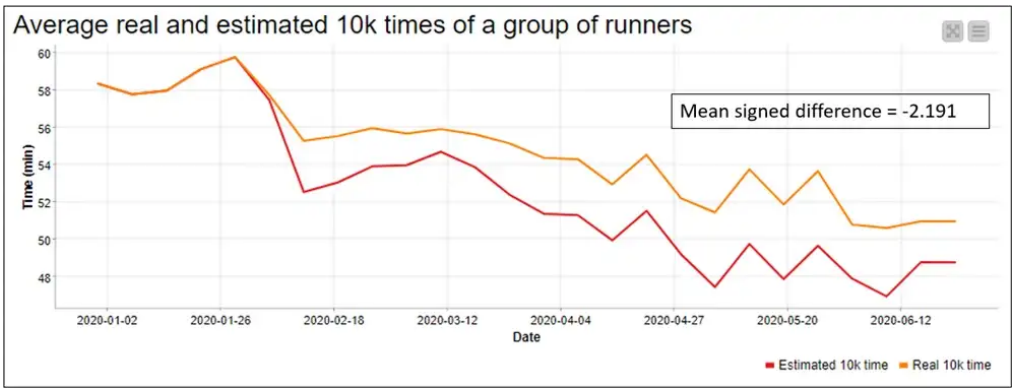
than for ice cream (0.266). However, this huge difference is due to the low values of ice cream sales in the winter months. The line plot on the right in the figure below shows exactly the same actual and predicted sales of ice cream and milk, with ice cream sales scaled up by 25 items for each month. Without the bias from the values close to zero, the forecasting accuracies for ice cream (MAPE=0.036) and milk (MAPE=0.016) are now much closer to each other.

Mean Signed Difference

Does a running app provide unrealistic expectations?

A smartwatch can be connected to a running application which then estimates the finishing time in a 10k run. It could be that, as a motivator, the app estimates the time lower than what's realistically expected.

To test this, we collect the estimated and realized finishing times from a group of runners for six months and plot the average values in the line plot below. As you can see, during the six months, the realized finishing time (orange line) decreases more slowly than the estimated finishing time (red line). We confirm the systematic bias in the estimates by calculating the mean signed difference between the actual and estimated finishing times. It's negative (-2.191), so the app indeed raises unrealistic expectations! Notice, though, that this metric is not informative about the magnitude of the error because if there's a runner who actually runs faster than the expected time, this positive error compensates a part of the negative error.



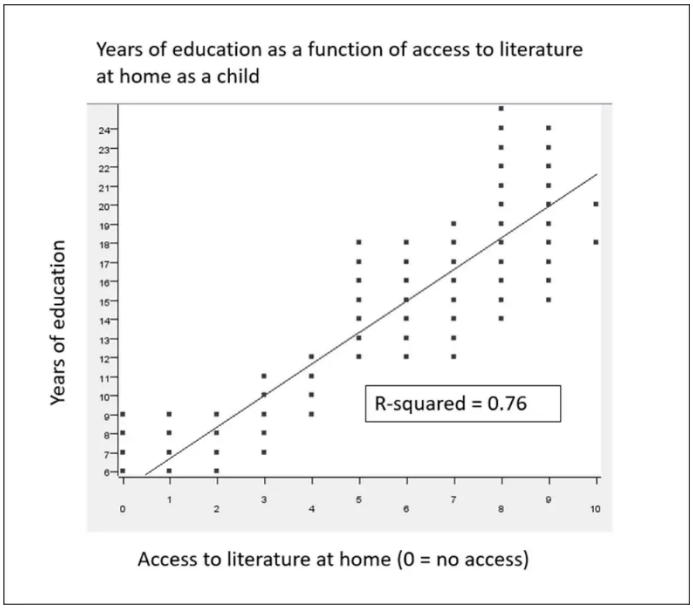
Estimated (red line) and realized (orange line) finishing times in a 10k run in the period of six months. The estimated times are biased downwards, also shown by the negative value of mean signed difference.

R-squared

How much of our years of education can be explained through access to literature?

The figure below shows the relationship between the access to literature (x-axis) and years of education (y-axis) in a sample of the population. A linear regression line is fitted to the data to model the relationship between these two variables. To measure the fit of the linear regression model, we use R-squared.

R-squared tells how much of the variance of the target column (years of education) the model explains. Based on the R-squared value of the model, 0.76, the access to literature explains 76% of the variance in the years of education.



Linear regression line modeling the relationship between access to literature and years of education. R-squared is used to measure the model fit, i.e., how much of the variance in the target column (years of education) can be explained by the model, 76% in this case.

A review of the five Numeric Scoring Metrics

The numeric scoring metrics introduced above are shown in following table. The metrics are listed along with the formulas used to calculate them and a few key properties of each. In the formulas, y_i is the actual value and $f(x_i)$ is the predicted value.

Example Use Case	Numeric Scoring Metric	Formula	Properties
Which model best captures the rapid changes in the volatile stock market?	**1.1** Mean squared error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(f(x_i)-y_i)^2$	Weights big differences more
	1.2 Root mean squared error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i)-y_i)^2}$	
Which model best estimates the energy consumption in the long term?	**2**. Mean absolute error (MAE)	$\frac{1}{n}\sum_{i=1}^{n}\lvert f(x_i)-y_i\rvert$	Equal weights to all distances
Are the sales forecasting models for different products equally accurate?	**3**. Mean absolute percentage error (MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert f(x_i)-y_i\rvert}{\lvert y_i\rvert}$	Requires non-zero target column values
Does a running app provide unrealistic expectations?	**4**. Mean signed difference	$\frac{1}{n}\sum_{i=1}^{n}(f(x_i)-y_i)$	Only informative about the direction of the error
How much of our years of education can be explained through access to literature?	**5**. R-squared	$1-\frac{\sum_{i=1}^{n}(f(x_i)-y_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$	Universal range: the closer to 1 the better

Common numeric scoring metrics, their formulas, and key properties. In the formulas, y_i is the actual value, $f(x_i)$ is the forecasted value, and n is the sample size.

Summary

In this article, we've introduced the most commonly used error metrics and the perspectives that they provide to the model's performance.

It's often recommended to take a look at multiple numeric scoring metrics to gain a comprehensive view of the model's performance. For example, by reviewing the mean signed difference, you can see if your model has a systematic bias, whereas by studying the (root) mean squared error, you can see which model best captures the sudden fluctuations. Visualizations, a line plot, for example, complement the model evaluation.

For a practical implementation, take a look at the example workflows built in the visual data science tool KNIME Analytics Platform.

Download and inspect these free workflows from the KNIME Community Hub:

- Evaluating the Performance of a Regression Model
- Forecasting and Reconstructing Time Series

This chapter was first published in The New Stack.

Visual Scoring Techniques for Classification Models

Author: Maarit Widmann

Workflow on KNIME Community Hub: Visual Scoring Techniques for Classification Models

Is 99% accuracy good for a churn prediction model? If in reality 1% of the customers churn and 99% don't, the model is doing equally well as a random guess. If 10% of the customers churn and 90% don't, then the model is doing better than the random guess.

Accuracy statistics, such as overall accuracy, quantify the expected performance of a Machine Learning model on new data without any baseline, such as a random guess or existing models.

That's why we also need visual model evaluation - or scoring - techniques that show the model performance in a broader context: for varying classification thresholds, compared to other models, and also from the perspective of resource usage. In this article, we explain how to evaluate a classification model with an ROC curve, a lift chart, and a cumulative gain chart, and provide a link to their practical implementation in a KNIME workflow, Visual Scoring Techniques for Classification Models.

Use Case: Churn Prediction Model

We'll demonstrate the utility of visual model evaluation techniques through a Random Forest model (100 trees) for churn prediction.

We use a dataset with 3333 telco customers, including their contract data and phone usage, available on GitHub. The target column "Churn?" shows whether the customer churned (True) or not (False). 483 customers (14%) churned, and 2850 customers (86%) didn't churn.

The accuracy statistics of the model are shown in the figure below:

Accuracy Statistics

Random Forest model for churn prediction

Confusion Matrix

	True (Predicted)	False (Predicted)	
True (Actual)	57	40	58.76%
False (Actual)	3	567	99.47%
	95.00%	93.41%	

Class Statistics

Class	Recall	Precision	Sensitivity	Specificity	F-measure
True	58.76%	95.00%	58.76%	99.47%	72.61%
False	99.47%	93.41%	99.47%	58.76%	96.35%

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
93.55%	6.45%	0.692	624	43

Reset Apply Close

Confusion matrix, class statistics, and overall accuracy statistics of the Random Forest model for churn prediction.

From the accuracy statistics table, we draw the following conclusions:

- The overall accuracy is around 94%, which means that 94 out of every 100 customers in the test data got a correct class prediction as Churn = True or Churn = False.
- The sensitivity value is around 59% for the class True. This implies that around 6 out of every 10 customers who churned (Churn=True) were predicted correctly to churn, while the remaining 4 were predicted incorrectly to not churn.
- The specificity value around 99% for the class True indicates that almost all of the customers who didn't churn (Churn=False) were classified correctly.

In the following sections, we use different visual scoring technique to gain additional information about the model's performance.

Comparing Performances across Classification Thresholds

The accuracy statistics are calculated based on the actual and predicted target classes. The predicted classes, here True and False, are based on class probabilities (or scores) predicted by the model, ranging between 0 and 1. In a binary classification problem, the model outputs two probabilities, one for each class. By default, the class with the highest probability determines the predicted class, which in a binary

classification problem means that the class with a probability above 0.5 gets predicted. However, sometimes a different classification threshold can lead to a better performance. If this is the case, we can find it out in an ROC curve.

ROC Curve

An ROC curve (Receiving Operator Characteristics curve) plots the model performance for varying classification thresholds using two metrics: the false positive rate on the x-axis and the true positive rate on the y-axis.

In a binary classification task, one of the target classes is arbitrarily assumed to be the positive class, while the other class becomes the negative class. In our churn prediction problem, we have selected True to be the positive class and False the negative class.

The number of true positives (TP), false negatives (FN), false positives (FP), and true negatives (TN), as reported in the confusion matrix, are used to calculate the false positive rate and true positive rate.

- The **false positive rate**

  $FPR = \frac{FP}{TN+FP}$ measures the proportion of the customers who didn't churn but were incorrectly predicted to churn. This equals *1-specificity*.

- The **true positive rate**

  $TPR = \frac{TP}{TP+FN}$ measures the proportion of the customers who churned and were correctly predicted to churn. This equals *sensitivity*.

The first point of the ROC curve in the bottom left corner indicates the false positive rate (FPR) and true positive rate (TPR) obtained using the **maximum threshold value 1.0**. With this threshold, all customers with probability P(churn=True) > 1.0 will be predicted to churn, that is none. No customers are predicted to churn either correctly or incorrectly, and therefore FPR and TPR are both 0.0.

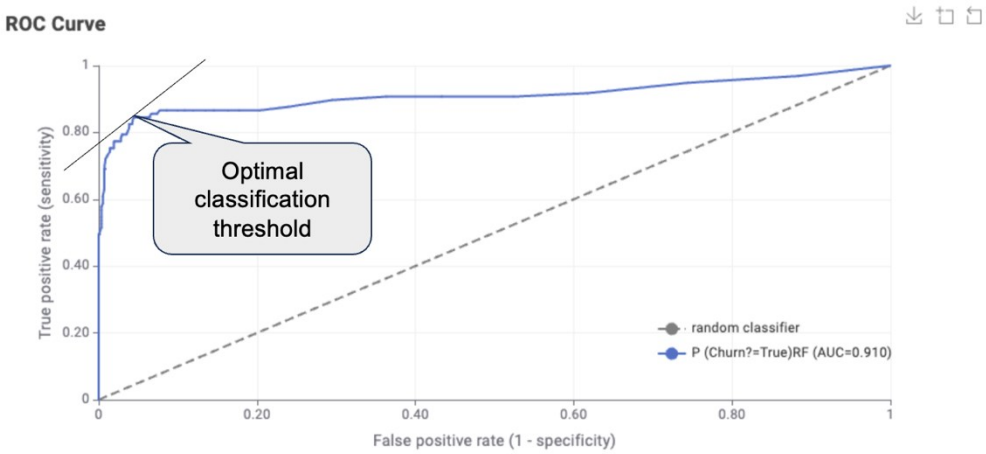
The second point in the ROC curve is drawn by decreasing the threshold value, for example by 0.1. Now all customers with *P(churn=True) > 0.9* will be predicted to churn. The threshold is still high, but it is now possible that some customers will be predicted to churn, therefore producing small non-zero values of TPR or FPR. So, this point will be located in the ROC curve close to the previous point.

The third point is drawn by decreasing the threshold value some more, and so on, till we reach the last point in the curve drawn for the **minimum classification threshold 0.0**. With this threshold, **all** customers are assigned to the positive class *True*, either correctly or incorrectly, and therefore both TPR and FPR are 1.0.

A perfect model would produce TPR=1.0 and FPR=0.0. On the opposite, a random classifier would always make an equal number of correct and incorrect predictions into the positive class, which corresponds to the black diagonal line where FPR=TPR. This line is reported in each ROC curve as a reference for a useless model.

Notice that a model can of course perform worse than the random guess, but that might be a mistake of the data scientist rather than the model! It would be enough to take the opposite of the model decision to implement a better performing solution.

Below, you see an ROC Curve, drawn for the Random Forest model for churn prediction. Notice the dotted line corresponding to the random guess.



ROC curve shows the false positive rate on the x-axis and true positive rate on the y-axis for all possible classification thresholds from 0 to 1.

The optimal classification threshold for the model is located as close as possible to the top left corner - with TPR=1.0 and FPR=0.0 - occupied by the perfect model. This optimal point has the closest tangent to (0.0, 1.0). With this optimal classification threshold we have the least false positives for each true positive. Our example model predicts on average

$$\frac{0.85}{0.05} = 17$$

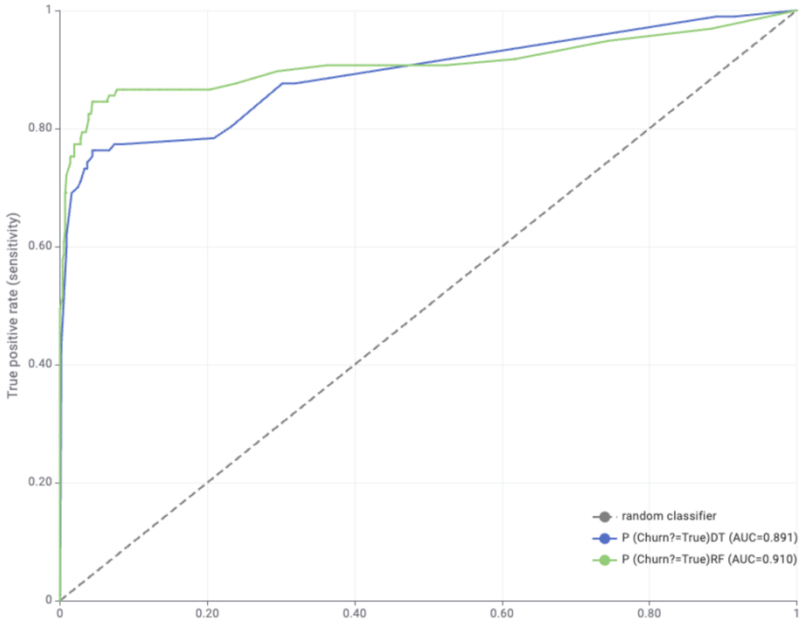
true positives for each false positive, when we use the optimal classification threshold.

If we compare these numbers to the class statistics, we can say that, after optimizing the classification threshold, the sensitivity increases quite a lot from 59% to 85%, while specificity decreases only a bit from 99% to 100% - 5% = 95%.

Comparing multiple Models

An ROC curve is also useful for comparing models. Let's train another model, a Decision Tree, for the same churn prediction task and compare its performance with the Random Forest model.

The figure below shows two ROC curves in the same view. The green curve is for the Random Forest model and the blue curve is for the Decision Tree:



ROC curves of two models - a Random Forest and a Decision Tree - for churn prediction. The model which reaches closer to the top left corner and has a greater AuC statistics performs better.

In the ROC curve for multiple models, a curve closer to the top left corner, in this case the green curve of the Random Forest model, implies better performance. The view also shows the Area under the Curve (AuC) statistics for both models in the bottom right corner. It measures the area underlying each ROC curve and allows for a more refined comparison between the performances.

In the next section, we show how you can visually evaluate the model based on the required efforts to reach the positive class.

Saving Resources with a Model

Besides getting accurate predictions, we can also save resources with the model. In our example of churn prediction, some kind of an action follows the prediction, and the action requires resources, such as less revenue or more time investment. A model can

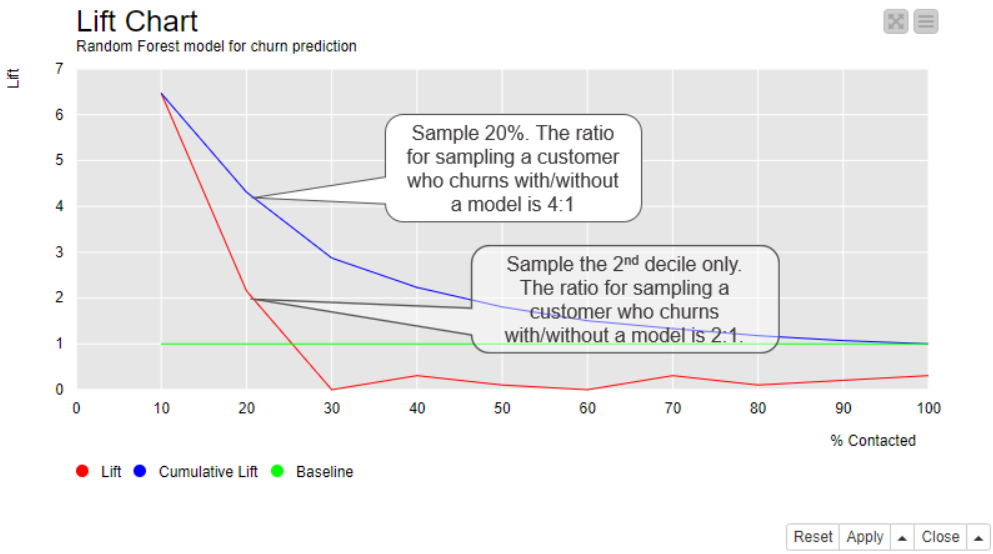
help us to use the resources more efficiently: apply fewer actions but still reach more of the customers who are likely to churn.

The lift and cumulative gain charts compare the resource usage against the correct predictions.

Lift Chart

A lift chart compares the number of target customers - here the customers who churn - reached in a sample that has been extracted based on the model predictions vs in a random sample.

Below you see the lift chart of the Random Forest model:



Lift chart shows the ratio of target customers reached in a sample which is drawn based on the model predictions vs in a random sample.

The x-axis shows each decile of the data ordered according to their predicted positive class probabilities from the highest to the lowest. For example, if we have 100 customers in the data, the first decile contains 10 customers with the highest predicted positive class probability, that is 10 customers who are most likely to churn. In the second decile we have other 10 customers with a lower probability than the first 10 customers, but higher than the remaining 80 customers. The tenth decile contains 10 customers with the lowest probability; 10 customers who are least likely to churn.

The lift shown by the cumulative lift line (the blue line) is the ratio of target customers reached in a sample which is drawn from ordered data as shown on the x-axis vs in a random sample. The lift is about 6 for the first decile. Since 14% of the customers in the original data churn, the probability of reaching a target customer in a random

sample is 14%. If we randomly sample 10 customers out of 100 customers, we expect to reach $0.14 * 10 = 1.4$ target customers. If we sample 10 first customers from the ordered data, we expect to reach 6 times more, that is $6 * 1.4 = 8.4$ target customers.

If we increase the sample size by further 10%, the cumulative lift is around 4. We would now reach $0.14 * 20 = 2.8$ target customers in a random sample, and $4 * 2.8 = 11.2$ target customers in a sample from the ordered data. The more data we sample, the more target customers are reached also by random sampling. This explains why the difference between the cumulative lift and the baseline (the green line) decreases towards the tenth decile.

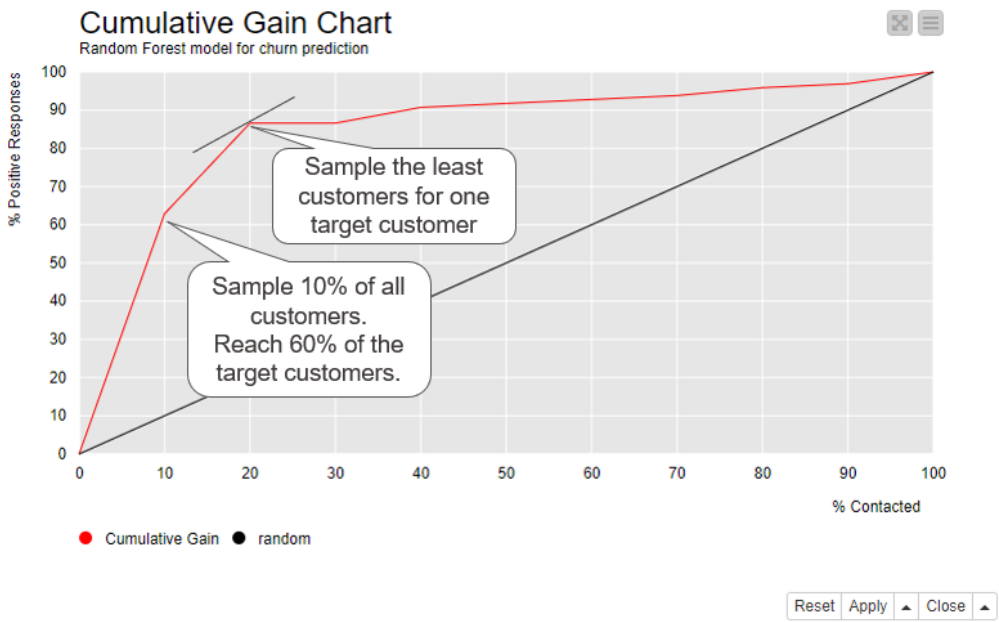
The lift line (the red line) shows the lift for each decile separately. The lift is above the baseline for the first two deciles and below it from the third decile forward. This means that if we sample the first decile from 100 customers, we expect to have $6 * 1.4 = 8.4$ target customers. If we sample the second decile but not the first decile, we expect to have $2 * 1.4 = 2.8$ target customers. If we sample any of the 3th to the 10th decile, we expect to have maximum $0.25 * 1.4 = 0.35$ target customers, because among these 80% of the data, the lift stays at a very low level between 0 and 0.25.

Cumulative Gain Chart

A cumulative gain chart shows which proportion of the target customers can be reached with which sample size. Similarly to the lift chart, the cumulative gain chart shows the data ordered by their positive class probabilities on the x-axis. On the y-axis it shows the proportion of target customers reached.

The figure below shows the cumulative gain chart for the Random Forest model.

If we follow the curve, we can see that if we only sample 10% of the customers, those with the highest probability (x-axis), we expect to reach around 60% of all customers who churn (y-axis). If we sample 20% of the customers, again those with the highest probability, we expect to reach more than 80% of all customers who churn. This point also has the closest tangent to the top left corner. With this number of sampled customers, the average sample size required to reach one target customer is the lowest.



Cumulative gain chart shows which proportion of target customers we reach when we contact 10%, 20%, …, 100% of the customers ordered by their positive class probability.

Visual Model Evaluation Techniques – Summary

The table below collects the techniques described above and summarizes what they report about the model's performance.

Evaluation perspective	ROC curve	Lift chart	Cumulative gain chart
Show the performances across classification thresholds	✅	❌	❌
Compare to the performance of a random classifier	✅	✅	✅
Show the ratio of target class events reached with/ without a model	❌	✅	❌
Show which sample size is required to reach which proportion of the target class	❌	❌	✅

Summary of the visual evaluation techniques for a classification model.

These visual techniques complement the accuracy statistics in that they show the optimal classification threshold, compare the performance to a random guess, compare multiple models in one view, and indicate the optimal sample size and quality.

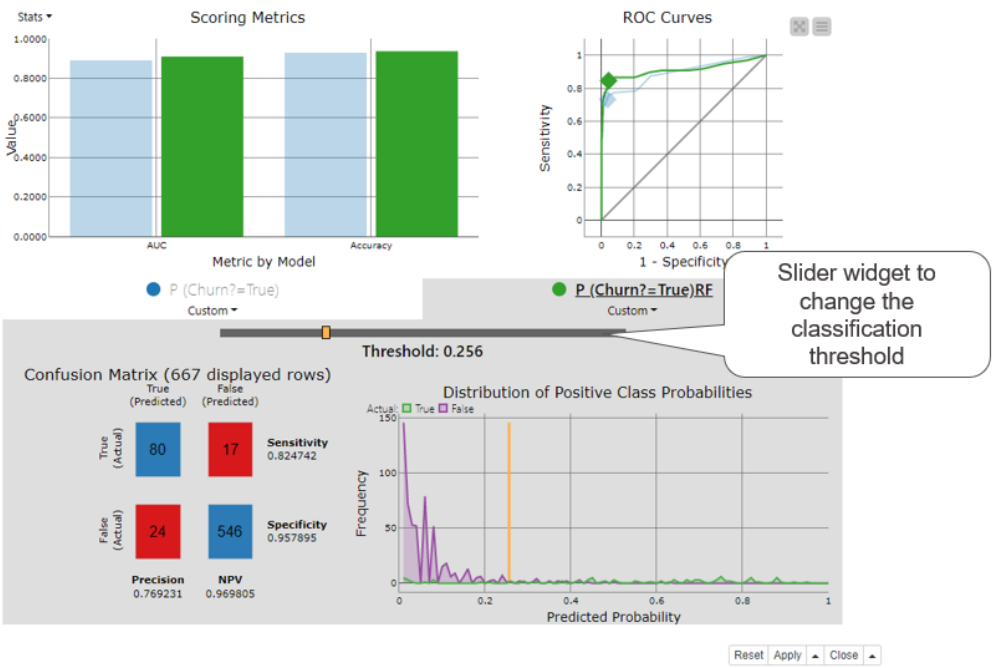
The ROC curve shows the performances across different classification thresholds, compares the performance to a random guess, and also compares the performances of multiple models. The lift chart and cumulative gain charts show if the model enables us to invest less resources but still reach the desired outcome.

These visual techniques complement but don't replace the accuracy statistics. For a comprehensive model evaluation, it's good to take a look at both.

Tips and Tricks

KNIME Analytics Platform provides a Binary Classification Inspector node that can be used to compare the accuracy statistics and ROC curves of multiple models and also find the optimal classification threshold. Its interactive view shows:

- A bar chart for overall accuracy and class statistics
- An ROC curve
- The confusion matrix
- The distribution of positive class probabilities
- A slider widget for the classification threshold

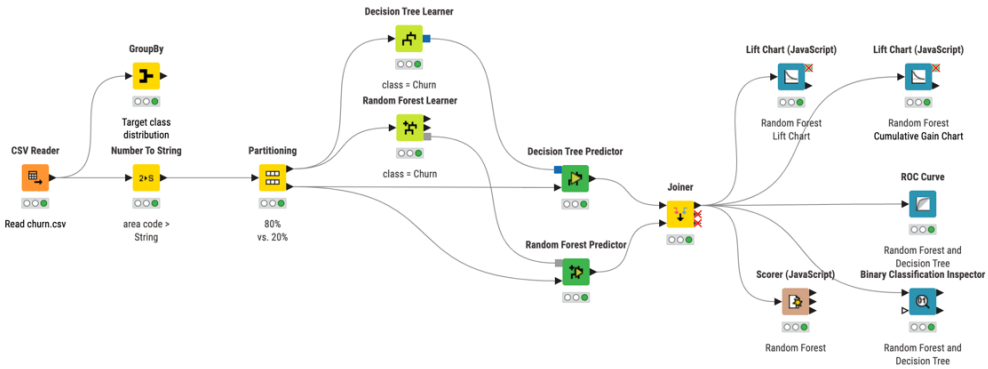


Binary classification inspector node's interactive view displays a bar chart for accuracy statistics, ROC curve, confusion matrix, and distribution of positive class probabilities in one view for one or more classification models. All views will be updated when the slider widget is used to adjust the classification threshold.

The top part of the Binary Classification Inspector node's view shows a bar chart for accuracy statistics and an ROC curve. Each model is displayed by a different color, here green for the Random Forest model and blue for the Decision Tree. We can select one of the models for a more detailed inspection by clicking on a colored bar. We've selected the Random Forest model in the figure above. The bottom part of the view activates and shows the confusion matrix and the distribution of the positive class probabilities for the selected model. The purple and green line in the distribution chart show the number of customers for each predicted probability separately for the two target classes. The orange vertical line shows the current classification threshold.

The classification threshold is at 0.5 by default. Using the threshold slider widget we can change this value: to the left towards zero or to the right towards 1. When we do, all other charts and plots in the view are automatically adjusted according to the new classification threshold. For example, when we move it to the left, the diamond in the ROC curve moves to the right. We stop moving the threshold when the diamond reaches the point for the optimal threshold, in this case at 0.256. This is the optimal classification threshold for the Random Forest model. Quite a lot different from the default 0.5!

The binary classification inspector view and the visual model evaluation techniques introduced in this article are implemented in the [Visual Scoring Techniques for a Classification Model]() workflow. You can inspect and download this workflow for free from the KNIME Community Hub.



This workflow, [Visual Scoring Techniques for a Classification Model,]() implements the following visual model scoring techniques: ROC Curve, Lift Chart, and Cumulative Gain Chart. In addition, the Binary Classification Inspector node is used to combine multiple scoring techniques in one interactive view. The workflow is available for download from the KNIME Community Hub.

Correcting Predicted Class Probabilities in Imbalanced Datasets

Authors: Alfredo Roccato and Maarit Widmann

Workflow on KNIME Community Hub: Adjusting Class Probabilities after Resampling

Classification on Imbalanced Datasets

It is not unusual in machine learning applications to deal with imbalanced datasets such as fraud detection, computer network intrusion, medical diagnostics, and many more.

Data imbalance refers to unequal distribution of classes within a dataset, namely that there are far fewer events in one class in comparison to the others. If, for example we have credit card fraud detection dataset, most of the transactions are not fraudulent and very few can be classed as fraud detections. This underrepresented class is called the minority class, and by convention, the positive class.

It is recognized that classifiers work well when each class is fairly represented in the training data.

Therefore if the data are imbalanced, the performance of most standard learning algorithms will be compromised, because their purpose is to maximize the overall accuracy. For a dataset with 99% negative events and 1% positive events, a model could be 99% accurate, predicting all instances as negative, though, being useless. Put in terms of our credit card fraud detection dataset, this would mean that the model would tend to classify fraudulent transactions as legitimate transactions. Not good!

As a result, overall accuracy is not enough to assess the performance of models trained on imbalanced data. Other statistics, such as Cohen's kappa and F-measure, should be considered. F-measure captures both the precision and recall, while Cohen's kappa takes into account the a priori distribution of the target classes.

The ideal classifier should provide high accuracy over the minority class, without compromising on the accuracy for the majority class.

Resampling to balance Datasets

To work around the problem of class imbalance, the rows in the training data are resampled. The basic concept here is to alter the proportions of the classes (a priori distribution) of the training data in order to obtain a classifier that can effectively predict the minority class (the actual fraudulent transactions).

Resampling Techniques

- **Undersampling**

  A random sample of events from the majority class is drawn and removed from the training data. A drawback of this technique is that it loses information and potentially discards useful and important data for the learning process.

- **Oversampling**

  Exact copies of events representing the minority class are replicated in the training dataset. However, multiple instances of certain rows can make the classifier too specific, causing overfitting issues.

- **SMOTE (Synthetic Minority Oversampling Technique)**

  "Synthetic" rows are generated and added to the minority class. The artificial records are generated based on the similarity of the minority class events in the feature space.

Correcting predicted Class Probabilities

Let’s assume that we train a model on a resampled dataset. The resampling has changed the class distribution of the data from imbalanced to balanced. Now, if we apply the model to the test data and obtain predicted class probabilities, they won’t reflect those of the original data. This is because the model is trained on training data that are not representative of the original data, and thus the results do not generalize into the original or any unseen data. This means that we can use the model for prediction, but the class probabilities are not realistic: we can say whether a transaction is more probably fraudulent or legitimate, but we cannot say how probably it belongs to one of these classes. Sometimes we want to change the classification threshold because we want to take more/less risks, and then the model with the corrected class probabilities that haven't been corrected wouldn't work anymore.

After resampling, we have now trained a model on balanced data i.e. that contain an equal number of fraudulent and legitimate transactions, which is luckily not a realistic scenario for any credit card provider and therefore - without correcting the predicted class probabilities - would not be informative about the risk of the transactions in the next weeks and months.

If the final goal of the analysis is not only to classify based on the highest predicted class probability, but also to get the correct class probabilities for each event, we need to apply a transformation to the obtained results. If we don't apply the transformation to our model, grocery shopping with a credit card in a supermarket might raise too much interest!

The following formula[1] shows how to correct the predicted class probabilities for a binary classifier:

$$P'_{+} = \frac{P_{+} * (\frac{p_{o+}}{p_{t+}})}{P_{+} * (\frac{p_{o+}}{p_{t+}}) + (1 - P_{+}) * \frac{(1 - p_{o+})}{(1 - p_{t+})}}$$

$$P'_{-} = \frac{(1 - P_{+}) * \frac{(1 - p_{o+})}{(1 - p_{t+})}}{P_{+} * (\frac{p_{o+}}{p_{t+}}) + (1 - P_{+}) * \frac{(1 - p_{o+})}{(1 - p_{t+})}}$$

where

P_{+} *is the positive class probability predicted by the model,*

p_{o+} *is the proportion of the positive class in the original data,*

p_{t+} *is the proportion of the positive class in the training data,*

and P'_{+} *is the corrected positive class probability.*

P'_{-} *is its complement to 1, i.e., the corrected negative class probability.*

For example, if the proportion of the positive class in the original dataset is 1% and after resampling it is 50%, and the predicted positive class probability is 0.95, applying the correction it gives:

$$P'_{+} = \frac{0.95 * (\frac{0.01}{0.50})}{0.95 * (\frac{0.01}{0.50}) + (1 - 0{,}95) * \frac{(1 - 0.01)}{(1 - 0.50)}} \approx 0.161$$

Example: Fraud Detection

When we apply a classification model to detect fraudulent transactions, the model has to work reliably on imbalanced data. Although few in number, fraudulent transactions can have remarkable consequences. Therefore, it's worth checking how much we can

---

[1] *Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. Neural computation 14(1):21–41, 2002.*

improve the performance of the model and its usability in practice by resampling the data and correcting the predicted class probabilities.

Evaluating the Cost of a Classification Model

In the real world, the performance of a classifier is usually assessed in terms of cost-benefit analysis: correct class predictions bring profit, whereas incorrect class predictions bring cost. In this case, fraudulent transactions predicted as legitimate cost the amount of fraud, and transactions predicted as fraudulent - correctly or incorrectly - bring administrative costs.

Administrative costs (Adm) are the expected costs of contacting the card holder and replacing the card if the transaction was correctly predicted as fraudulent, or reactivating it if the transaction was legitimate. Here we assume, for simplicity, that the administrative costs for both cases are identical.

The cost matrix below summarizes the costs assigned to the different classification results. The minority class, “fraudulent”, is defined as the positive class, and “legitimate” is defined as the negative class.

	Predicted class	
Actual class	Legitimate (Negative)	Fraudulent (Positive)
Legitimate (Negative)	0	Adm
Fraudulent (Positive)	Amount of fraud	Adm

The cost matrix that shows the costs assigned to different classification results as obtained by a model for fraud detection. Correctly classified legitimate transactions bring no cost. Fraudulent transactions predicted as legitimate cost the amount of fraud. Transactions predicted as fraudulent bring administrative costs.

Based on this cost matrix, the total cost of the model is:

$$Cost\ of\ the\ model = FN * AMOUNT_j + FP * Adm + TP * Adm$$

where

$FN\ (false\ negatives)$*is the number of fraudulent transactions predicted as legitimate,*

$FP\ (false\ positives)$*is the number of legitimate transactions predicted as fraudulent,*

$TP\ (true\ positives)$*is the number of fraudulent transactions predicted as fraudulent,*

and $AMOUNT_j$ *is the amount of fraud for the fraudulent transaction j predicted as legitimate.*

Finally, the cost of the model will be compared to the amount of fraud. Cost reduction tells how much cost the classification model brings compared to the situation where we don’t use any model:

$$Cost\ reduction = \frac{Amount\ of\ fraud - Cost\ of\ the\ model}{Amount\ of\ fraud}$$

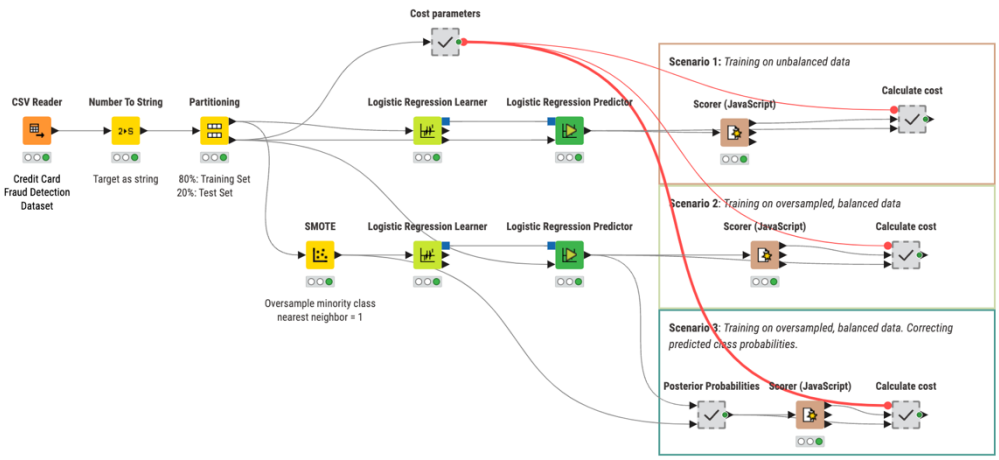
The Workflow

In this example we use the "Credit Card Fraud Detection" dataset provided by Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset contains 284 807 transactions made by European credit card holders during two days in September 2013. The dataset is highly imbalanced: 0.172% (492 transactions) were fraudulent and the rest were normal. Other information on the transactions has been transformed into principal components.

The workflow in the figure below shows the overall process of reading the data, partitioning the data into a training and test set, resampling the data, training a classification model, predicting and correcting the class probabilities, and evaluating the cost reduction. We selected SMOTE as the resampling technique and logistic regression as the classification model. Here we estimate administrative costs to be 5 euros.

The workflow provides three different scenarios for the same data:

- training and applying the model using imbalanced data,
- training the model on balanced data and applying the model to imbalanced data without correcting the predicted class probabilities,
- training the model on balanced data and applying the model to imbalanced data where the predicted class probabilities have been corrected.



This workflow, Adjusting Class Probabilities after Resampling, compares three ways of training and applying a classification model using imbalanced data. Firstly, the model training is done on imbalanced data. Secondly, the training set is resampled using SMOTE to make it balanced. Thirdly, the training set is resampled using SMOTE and predicted class probabilities are corrected based on the a priori class distribution of the data. The workflow is available for download from the KNIME Community Hub.

Estimating the Cost for Scenario 1 without Resampling

A logistic regression model provides these results:

	Predicted class	
Actual class	Legitimate (Negative)	Fraudulent (Positive)
Legitimate (Negative)	56854	10
Fraudulent (Positive)	40	58

False Negative rate	False Positive rate	Specificity	Sensitivity/Recall	Precision	Accuracy	F-measure	Cohen's kappa	Cost of the model	Amount of fraud	Cost reduction
40.82 %	0.02 %	99.98 %	59.18 %	85.29 %	99.91 %	0.699	0.698	7627 €	14806 €	0.419

The confusion matrix, class statistics and estimated cost reduction obtained by a fraud detection model that was trained on imbalanced data. The cost reduction is evaluated using the formula in the "Evaluating the cost of a classification model" section.

The setup in this scenario provides good values for F-measure and Cohen's Kappa statistics, but a relatively high False Negative Rate (40.82%). This means that more than 40% of the fraudulent transactions were not detected by the model - increasing the amount of fraud and therefore the cost of the model. The cost reduction of the model compared to not using any model is 42%.

Estimating the Cost for Scenario 2 with Resampling

A logistic regression model trained on a balanced training set (oversampled using SMOTE) yields these results:

	Predicted class	
Actual class	Legitimate (Negative)	Fraudulent (Positive)
Legitimate (Negative)	56473	391
Fraudulent (Positive)	12	86

False Negative Rate	False Positive Rate	Specificity	Sensitivity/Recall	Precision	Accuracy	F-measure	Cohen's kappa	Cost of the model	Amount of fraud	Cost reduction
12.24 %	0.69 %	99.31 %	87.76 %	18.03 %	99.29 %	0.299	0.297	4776 €	14806 €	0.636

The confusion matrix, class statistics and estimated cost obtained by a fraud detection model that was trained on an oversampled, balanced data. The cost is evaluated using the formula in the "Evaluating the cost of a classification model" section.

The False Negative Rate is very low (12.24 %), which means that almost 90 % of the fraudulent transactions were detected by the model. However, there are a lot of "false alarms" (391 legitimate transactions predicted as fraud) that increase the administrative costs. However, the cost reduction achieved by training the model on a

balanced dataset is 64% - higher than what we could reach without resampling the training data. The same test set was used for both scenarios.

Estimating the Cost for scenario 3 with Resampling and Correcting the predicted Class Probabilities

A logistic regression model trained on a balanced training set (oversampled using SMOTE) yields these results when the predicted probabilities have been corrected according to the a priori class distribution of the data.

As the results for this scenario in the table above show, correcting the predicted class probabilities leads to the best model of these three scenarios in terms of the greatest cost reduction.

In this scenario, where we train a classification model on an oversampled data and correct the predicted class probabilities according to the a priori class distribution in the data, we reach a cost reduction of 75 % compared to not using any model.

	Predicted class	
Actual class	Legitimate (Negative)	Fraudulent (Positive)
Legitimate (Negative)	56841	23
Fraudulent (Positive)	22	76

False Negative Rate	False Positive Rate	Specificity	Sensitivity/Recall	Precision	Accuracy	F-measure	Cohen's kappa	Cost of the model	Amount of fraud	Cost reduction
22.45 %	0.04 %	99.96 %	77.55 %	76.77 %	99.92 %	0.772	0.771	3233 €	14806 €	0.754

The confusion matrix, class statistics and estimated cost as obtained by a fraud detection model that was trained on an oversampled, balanced data, and where the predicted class probabilities were corrected according to the a priori class distribution. The cost is evaluated using the formula in the "Evaluating the cost of a classification model" section.

Of course, the cost reduction depends on the value of the administrative costs. Indeed, we tried this by changing the estimated administrative costs and found out that this last scenario can attain cost reduction as long as the administrative costs are 0.80 euros or more.

Summary

Often, when we train and apply a classification model, the interesting events in the data belong to the minority class and are therefore more difficult to find: fraudulent transactions among the masses of transactions, disease carriers among the healthy people, and so on.

From the point of view of the performance of a classification algorithm, it's recommended to make the training data balanced. We can do this by resampling the training data. Now, the training of the model works better, but how about applying it to new data, which we assume to be imbalanced? This setup leads to biased values for the predicted class probabilities, because the training set does not represent the test set or any new, unseen data.

Therefore, to obtain an optimal performance of a classification model together with reliable classification results, correcting the predicted class probabilities by the information on the a priori class distribution is recommended. As the use case in this blog post shows, this correction leads to better model performance and concrete profit.

Cohen’s Kappa

What it is, When to use it, How to avoid Pitfalls

Author: Maarit Widmann

Workflow on KNIME Community Hub: [Cohen's Kappa for Evaluating Classification Models]

An Alternative for when Overall Accuracy is biased, yet not trusting the Statistics blindly

Cohen’s kappa is a metric often used to assess the agreement between two raters. It can also be used to assess the performance of a classification model.

For example, if we had two bankers, and we asked both to classify 100 customers in two classes for credit rating, i.e., good and bad, based on their creditworthiness, we could then measure the level of their agreement through Cohen's kappa.

Similarly, in the context of a classification model, we could use Cohen’s kappa to compare the machine learning model predictions with the manually established credit ratings.

Like many other scoring metrics, Cohen’s kappa is calculated based on the confusion matrix. However, in contrast to calculating overall accuracy, for example, Cohen’s kappa takes imbalance in class distribution into account and can therefore be more complex to interpret.

In this article we will:

- Guide you through the calculation and interpretation of Cohen’s Kappa values, particularly in comparison with overall accuracy values,
- Show that where overall accuracy fails because of a large imbalance in the class distribution, Cohen’s kappa might supply a more objective description of the model performance,
- Introduce a few tips to keep in mind when interpreting Cohen’s kappa values!

Measure Performance Improvement on imbalanced Datasets

Let's focus on a classification task on bank loans, using the German credit data provided by the UCI Machine Learning Repository. In this dataset, bank customers have been assigned either a "bad" credit rating (30%) or a "good" credit rating (70%) according to the criteria of the bank. For the purpose of this article, we exaggerated the imbalance in the target class credit rating via bootstrapping, giving us 10% with a "bad" credit rating and 90% with a "good" credit rating: a highly imbalanced dataset. Exaggerating the imbalance helps us to make the difference between "overall accuracy" and "Cohen's kappa" clearer in this article.

Let's partition the data into a training set (70%) and a test set (30%) using stratified sampling on the target column and then train a simple model, a decision tree, for example. Given the high imbalance between the two classes, the model will not perform too well. Nevertheless, let's use its performance as the baseline for this study.

Baseline Model

In the figure below you can see the confusion matrix and accuracy statistics for this baseline model. The overall accuracy of the model is quite high (87%) and hints at an acceptable performance by the model. However, in the confusion matrix, we can see that the model is able to classify only 9 out of the 30 credit customers with a bad credit rating correctly. This is also visible by the low sensitivity value of class "bad" - just 30%.

Credit Rating Prediction - Baseline Model

Confusion Matrix

	bad (Predicted)	good (Predicted)
bad (Actual)	9	21
good (Actual)	18	252

Class Statistics

Class	Recall	Precision	Sensitivity	Specificity	F-measure
bad	30.00%	33.33%	30.00%	93.33%	31.58%
good	93.33%	92.31%	93.33%	30.00%	92.82%

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
87.00%	13.00%	0.244	261	39

Confusion matrix and accuracy statistics for the baseline model, i.e. a decision tree model trained on the highly imbalanced training set. The overall accuracy is relatively high (87%), although the model detects just a few of the customers with a bad credit rating (sensitivity just at 30%).

Basically, the decision tree is classifying most of the "good" customers correctly and neglecting the necessary performance on the few "bad" customers. The imbalance in

the class a priori probability compensates for such sloppiness in classification. Let's note for now that Cohen's kappa value is just 0.244, within its range of [-1,+1].

Improved Model

Let's try to improve the model performance by forcing it to acknowledge the existence of the minority class. We train the same model this time on a training set where the minority class has been oversampled using the SMOTE technique, reaching a class proportion of 50 % for both classes.

To provide more detail about the confusion matrix for this model, 18 out of the 30 customers with a "bad" credit rating are detected by the model, leading to a new sensitivity value of 60% over the previous 30%. Cohen's kappa statistics is now 0.452 for this model, which is a remarkable increase from the previous value 0.244. But what about overall accuracy? For this second model it's 89%, not very different from the previous value 87%.

When summarizing we get two very different pictures. According to the overall accuracy, model performance hasn't changed very much at all. However, according to Cohen's kappa a lot has changed! Which statement is right?

Credit Rating Prediction - Improved Model

Confusion Matrix

	bad (Predicted)	good (Predicted)
bad (Actual)	18	12
good (Actual)	22	248

Class Statistics

Class	Recall	Precision	Sensitivity	Specificity	F-measure
bad	60.00%	45.00%	60.00%	91.85%	51.43%
good	91.85%	95.38%	91.85%	60.00%	93.58%

Overall Statistics

Overall Accuracy	Overall Error	Cohen's kappa (κ)	Correctly Classified	Incorrectly Classified
88.67%	11.33%	0.452	266	34

Confusion matrix and accuracy statistics for the improved model. The decision tree model trained on a more balanced training set, where the minority class has been oversampled. The overall accuracy is almost the same as for the baseline model (89% vs. 87%). However, Cohen's kappa value shows a remarkable increase from 0.244 to 0.452.

From the numbers in the confusion matrix, it seems that Cohen's kappa has a more realistic view of the model's performance when using imbalanced data.

Why does Cohen's kappa take more notice of the minority class? How is it actually calculated? Let's take a look!

Cohen's Kappa

Cohen's kappa is calculated with the following formula[2]:

$$\kappa = \frac{p_0 - p_e}{1 - p_e},$$

where p_0 is the overall accuracy of the model and p_e is the measure of the agreement between the model predictions and the actual class values as if happening by chance.

In a binary classification problem, like ours, p_e is the sum of p_{e1}, the probability of the predictions agreeing with actual values of class 1 ("good") by chance, and p_{e2}, the probability of the predictions agreeing with the actual values of class 2 ("bad") by chance. Assuming that the two classifiers - model predictions and actual class values - are independent, these probabilities, p_{e1} and p_{e2}, are calculated by multiplying the proportion of the actual class and the proportion of the predicted class.

Considering "bad" as the positive class, the baseline model assigned 9% of the records (false positives plus true positives) to class "bad", and 91% of the records (true negatives plus false negatives) to class "good". Thus p_e is:

$$\begin{aligned} p_e &= p_{e1} + p_{e2} \\ &= p_{e1,target} * p_{e1,pred} + p_{e2,target} * p_{e2,pred} \\ &= 0.90 * 0.91 \ + \ 0.10 * 0.09 \\ &= 0.828 \end{aligned}$$

And therefore Cohen's kappa statistics:

$$\kappa = \frac{0.870 - 0.828}{1 - 0.828} \approx 0.244$$

which is the same value as reported in accuracy statistics table.

Practically, Cohen's kappa removes the possibility of the classifier and a random guess agreeing and measures the number of predictions it makes that cannot be explained by a random guess. Furthermore, Cohen's kappa tries to correct the evaluation bias by taking into account the correct classification by a random guess.

Pain Points of Cohen's Kappa

At this point, we know that Cohen's kappa is a useful scoring metric when dealing with imbalanced data. However, Cohen's kappa has some downsides, too. Let's have a look at them one by one.

---

[2] *Bland, Martin. "Cohen's kappa." University of York Department of Health Sciences https://www-users.york.ac.uk/~mb55/msc/clinimet/week4/kappa_text.pdf. [Accessed May 29 2020] (2008).*

Full Range [-1, +1], but not equally reachable

It's easier to reach higher values of Cohen's kappa, if the target class distribution is balanced.

For the baseline model, the distribution of the predicted classes follows closely the distribution of the target classes: 27 predicted as "bad" vs. 273 predicted as "good" and 30 being actually "bad" vs. 270 being actually "good".

For the improved model, the difference between the two class distributions is greater: 40 predicted as "bad" vs. 260 predicted as "good" and 30 being actually "bad" vs. 270 being actually "good".

As the formula for maximum Cohen's kappa shows, the more the distributions of the predicted and actual target classes differ, the lower the maximum reachable Cohen's kappa value is. The maximum Cohen's kappa value represents the edge case of either the number of false negatives or false positives in the confusion matrix being zero, i.e. all customers with a good credit rating, or alternatively all customers with a bad credit rating, are predicted correctly.

$$\kappa_{max} = \frac{p_{max} - p_e}{1 - p_e},$$

where p_{max} is the maximum reachable overall accuracy of the model given the distributions of the target and predicted classes:

$$p_{max} = min\ (p_{target="bad"}, p_{predicted="bad"})\ +\ min\ (p_{target="good"}, p_{predicted="good"})$$

For the baseline model, we get the following value for p_{max}:

$$p_{max} = min\ (0.10, 0.09) + min(0.90, 0.91)\ =\ 0.99$$

Whereas for the improved model it is:

$$p_{max} = min\ (0.10, 0.13) + min(0.90, 0.87)\ =\ 0.97$$

The maximum value of Cohen's kappa is then for the baseline model:

$$\kappa_{max} = \frac{p_{max} - p_e}{1 - p_e} = \frac{0.99 - 0.828}{1 - 0.828} = 0.942$$

For the improved model it is:

$$\kappa_{max} = \frac{p_{max} - p_e}{1 - p_e} = \frac{0.97 - 0.796}{1 - 0.796} = 0.853$$

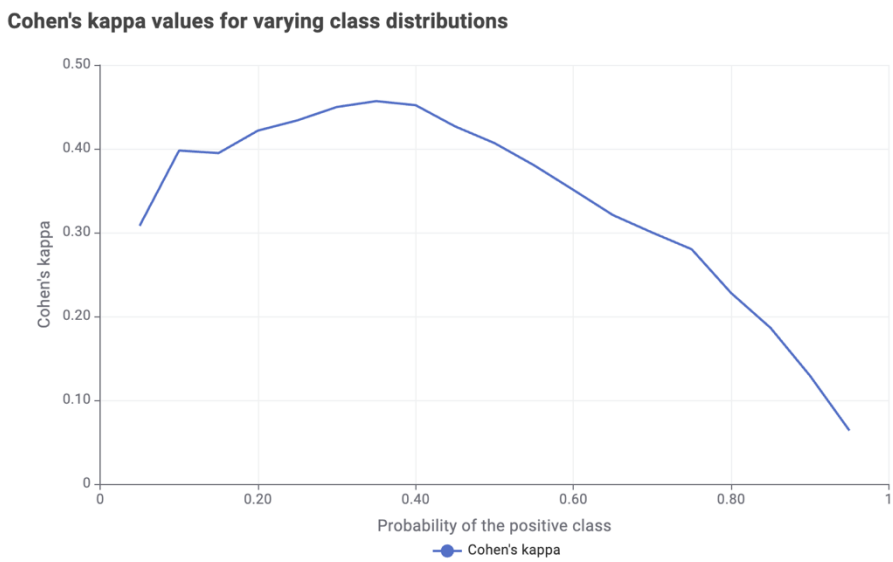
As the results show, the improved model with a greater difference in the distributions between the actual and predicted target classes can only reach a Cohen's kappa value as high as 0.853. Whereas the baseline model can reach the value 0.942, despite the worse performance.

Cohen's Kappa is higher for balanced Data

When we calculate Cohen's kappa, we strongly assume that the distributions of target and predicted classes are independent and that the target class doesn't affect the probability of a correct prediction. In our example this would mean that a credit customer with a good credit rating has an equal chance of getting a correct prediction as a credit customer with a bad credit rating. However, since we know that our baseline model is biased towards the majority "good" class, this assumption is violated.

If this assumption were not violated, like in the improved model where the target classes are balanced, we could reach higher values of Cohen's kappa. Why is this? We can rewrite the formula of Cohen's kappa as the function of the probability of the positive class, and the function reaches its maximum when the probability of the positive class is 0.5 [1]. We test this by applying the same improved model to different test sets, where the proportion of the positive "bad" class varies between 5% and 95%. We create 100 different test sets per class distribution by bootstrapping the original test data and calculate the average Cohen's kappa value from the results.

The figure below shows the average Cohen's kappa values against the positive class probabilities - and yes! Cohen's kappa does reach its maximum when the model is applied to balanced data!



Cohen's kappa values (on the y-axis) obtained for the same model with varying positive class probabilities in the test data (on the x-axis). The Cohen's kappa values on the y-axis are calculated as averages of all Cohen's kappas obtained via bootstrapping the original test set 100 times for a fixed class distribution. The model is the Decision Tree model trained on balanced data, introduced at the beginning of the article.

Cohen's Kappa says little about the expected Prediction Accuracy

The numerator of Cohen's kappa, p_0-p_e tells the difference between the observed overall accuracy of the model and the overall accuracy that can be obtained by chance. The denominator of the formula, 1-p_e, tells the maximum value for this difference.

For a good model, the observed difference and the maximum difference are close to each other, and Cohen's kappa is close to 1. For a random model, the overall accuracy is all due to the random chance, the numerator is 0, and Cohen's kappa is 0. Cohen's kappa could also theoretically be negative. Then, the overall accuracy of the model would be even lower than what could have been obtained by a random guess.

Given the explanation above, Cohen's kappa is not easy to interpret in terms of an expected accuracy, and it's often not recommended to follow any verbal categories as interpretations. For example, if you have 100 customers and a model with an overall accuracy of 87 %, then you can expect to predict the credit rating correctly for 87 customers. Cohen's kappa value 0.244 doesn't provide you with an interpretation as easy as this.

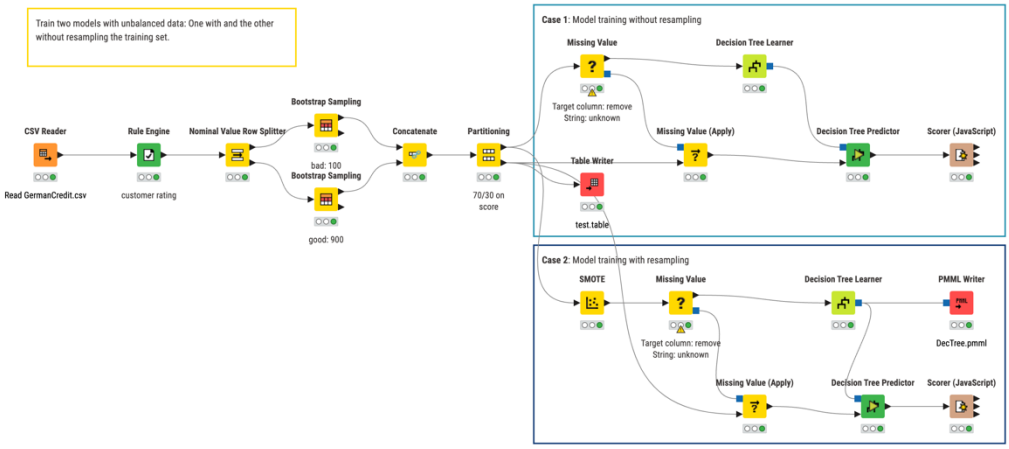
Summary

In this article we have explained how to use and interpret Cohen's kappa to evaluate the performance of a classification model. While Cohen's kappa can correct the bias of overall accuracy when dealing with unbalanced data, it has a few shortcomings. So the next time you take a look at the scoring metrics of your model, remember:

- Cohen's kappa is more informative than overall accuracy when working with unbalanced data. Keep this in mind when you compare or optimize classification models!
- Take a look at the row and column totals in the confusion matrix. Are the distributions of the target/predicted classes similar? If they're not, the maximum reachable Cohen's kappa value will be lower.
- The same model will give you lower values of Cohen's kappa for unbalanced than for balanced test data.
- Cohen's kappa says little about the expected accuracy of a single prediction

Example Workflow: Cohen's Kappa for evaluating Classification Models

The workflow used for this study is shown below. In the workflow we train, apply, and evaluate two decision tree models that predict the creditworthiness of credit customers. In the top branch, we train the baseline model, while in the bottom branch we train the model on the bootstrapped training set using the SMOTE technique.



This workflow, [Cohen’s Kappa for Evaluating Classification Models](), trains two decision trees to predict the credit score of customers. In the top branch, a baseline model is trained on the unbalanced training data (90% “good” vs. 10% “bad” class records). In the bottom branch, an improved model is trained on a new training dataset where the minority class has been oversampled (SMOTE). The workflow available for download from the KNIME Community Hub.

Resampling imbalanced Data Limits

Author: Maarit Widmann

Workflow on KNIME Community Hub: Resampling in Supervised Fraud Detection Models

Car parking ticket machines used to only accept coins. A self-service vegetable stand used to only accept cash. And not such a long time ago I could buy a bus ticket from the bus driver! These days, however, you can (and you're often encouraged to) pay for these and many other products and services by credit card. This leads to more and more transactions and also to types of transactions that didn't exist before. Some time back, a credit card transaction for a vegetable stand would have looked suspicious!

With the increasing variety and volume of credit card usage, fraud is evolving too[3]. This is a huge challenge! For automatic fraud detection and prevention, a number of supervised and unsupervised fraud detection models have been suggested. The unsupervised methods, such as a neural autoencoder, are anomaly detection models and don't require labeled data. The supervised methods, such as a decision tree or a logistic regression model, require labeled data, which are often not available. Imagine someone manually recognizing and labeling the transactions as "fraudulent" or "legitimate"! Another problem is that the fraudulent transactions are very few compared to the large amounts of legitimate transactions. This imbalance of the target classes decreases the performance of the decision tree algorithm and of other classification algorithms[4].

In this article, we will work with labeled, highly imbalanced transactions data: For each fraudulent transaction we have 579 legitimate transactions. We'll check if we can improve the performance of a decision tree model by resampling; that is, by artificially creating more data about fraudulent transactions. Along the way, we'll explain three different resampling methods and evaluate their effects on the fraud prevention application. At the end, we'll provide a link to a KNIME workflow – an example implementation of the different resampling methods.

---

[3] *"The Nilson Report." HSN Consultants, Inc., November 2019, Issue 1164, https://nilsonreport.com/publication_newsletter_archive_issue.php?issue=1164. Accessed 14 Oct. 2020.*

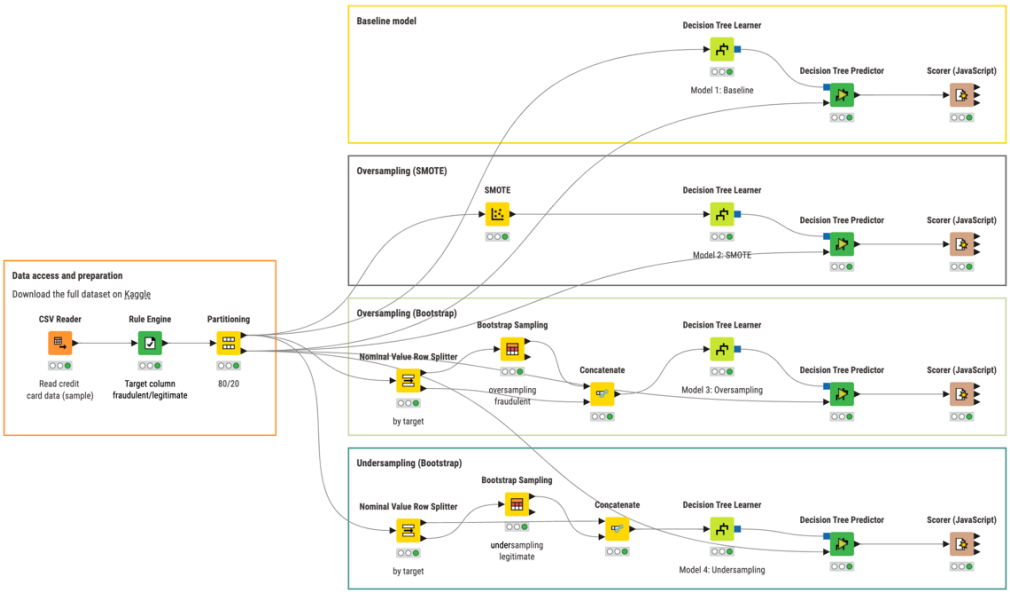
[4] *"Random Oversampling and Undersampling for Imbalanced Classification" Machine Learning Mastery Pty. Ltd., January 2020, https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/. Accessed 14 Oct. 2020.*

Building a Classification Model for Fraud Detection

In our demonstration we use the *creditcard.csv* data available on Kaggle. The data consist of 284807 credit card transactions, performed by EU cardholders in September 2013. 492 (0.2%) of the credit card transactions are fraudulent, and the remaining 284315 (99.8%) transactions are legitimate. The data contain a target class column with possible values *fraudulent / legitimate*, the time and amount of each transaction, and 28 principal components generated from the confidential features of the transactions.

The workflow below shows the steps for accessing, preprocessing, resampling, and modeling the transactions data. Inside the yellow box, we access the transactions data, encode the target column from 0/1 to legitimate/fraudulent, and partition the data into training and test sets using 80/20 split and stratified sampling on the target column. Inside the orange boxes, we build four different versions of a decision tree model for fraud detection: a baseline model trained on the original training data plus three models trained on

- SMOTE oversampled data,
- Bootstrap oversampled data, and
- Bootstrap undersampled data.

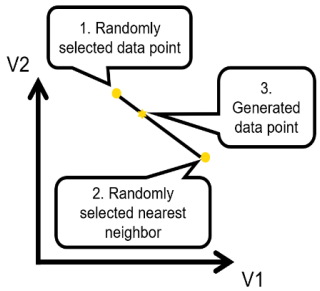
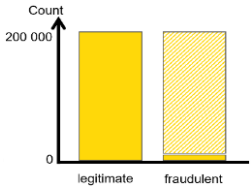


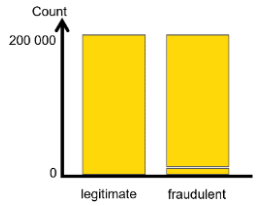
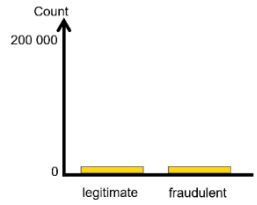
This workflow, Resampling in Supervised Fraud Detection Models, accesses and preprocesses transactions data and implements four different decision tree models for fraud detection: one trained on the original data and three models trained on resampled data. The workflow is available for download from the KNIME Community Hub.

Resampling Techniques

The table below summarizes the resampling methods that we include in our demonstration:

- oversampling (SMOTE)
- oversampling (Bootstrap)
- undersampling (Bootstrap)

Resampling method	Description	Target class distribution after resampling
Oversampling (SMOTE)	Generate new synthetic fraudulent transactions until the number of fraudulent transactions is ca. equal to the number of legitimate transactions: 1. Select one of the fraudulent transactions in the training data randomly 2. Select one of its *n* nearest neighbors in the same fraudulent class randomly 3. Select a random point between the existing fraudulent transaction and its nearest neighbor 	• Original data in yellow • New synthetic data in light patterned yellow 

Oversampling (Bootstrap)	Randomly draw with replacement a sample of fraudulent transactions until the number of fraudulent transactions is ca equal to the number of legitimate transactions	
Undersampling (Bootstrap)	Randomly draw with replacement as many legitimate transactions as there are fraudulent transactions	

Overview of three resampling methods: SMOTE, oversampling (Bootstrap), and undersampling (Bootstrap).

Effects of Resampling on Fraud Detection Performance

Resampling has two drawbacks, especially when the target class is as highly imbalanced as in our case. Firstly, oversampling the minority class might lead to overfitting, i.e. the model learns patterns that only exist in the particular sample that has been oversampled. Secondly, undersampling the majority class might lead to underfitting, i.e. the model fails to capture the general pattern in the data[5].

We compare the performances of the baseline model and the models trained on resampled data in terms of two scoring metrics: *recall and precision*. The metrics are explained in detail.

- **Recall** is the proportion of correctly predicted fraudulent transactions. The higher the recall, the more fraudulent transactions are prevented by the model.
- **Precision** is the proportion of actual fraudulent transactions among those predicted as fraudulent. The higher the precision, the fewer false alarms are raised by the model.

[5] *"Oversampling and Undersampling" Medium, September 2010, https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf. Accessed 14 Oct. 2020.*

Training data	Recall	Precision
Original data (Baseline)	85%	91%
Oversampling (SMOTE)	86%	44%
Oversampling (Bootstrap)	71%	71%
Undersampling (Bootstrap)	92%	2%

Recall and precision statistics obtained by four decision tree models for fraud detection, each one trained on a different training set.

The very low precision value 2% for the undersampled model in the bottom right corner in the table above indicates **underfitting**: The undersampled model failed to learn the patterns underlying the legitimate transactions. This seems reasonable considering that we discarded 99.8% of the legitimate transactions in the undersampling phase! Indeed, with so few examples in the fraudulent class, the only effect of undersampling has been to damage the representation of the legitimate class.

If you take a look at the performances obtained via oversampling in the two middle rows, you can see from their precision value that these models are raising more false alarms than the model trained on the full original data, while at the same time not improving the recognition of the pattern underlying the fraudulent transactions. All of this indicates that the model has **overfitted** the data.

As you can see, our fraud detection model is over-/underfitting when trained on resampled data. What is actually happening under the hood?

Demonstrating Over- and Underfitting in Fraud Detection

Transactions data are confidential, and we can therefore only work with principal components as predictors of the target class fraudulent/legitimate. In order to better understand how a resampled model leads to over- or underfitting, let's imagine we had some of the following columns in our data, since they often characterize fraudulent transactions[6]:

- Shipping address equals billing address: yes/no
- Urgent delivery: yes/no
- Number of different items in the order
- Number of orders of the same item
- Number of credit cards associated with the shipping address

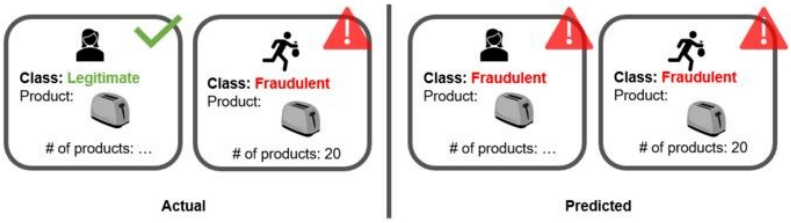
[6] https://www.bluefin.com/support/identifying-fraudulent-transactions/

Since our training data only contain 394 fraudulent transactions, it could be, for example, that the majority of them are characterized by exceptionally many orders of the same item: one for 20 toasters, another for 50 smartphones, yet another for 25 winter coats, and so on. In reality, the fraudulent transactions are much more varying and continuously evolving. On the contrary, the 227451 legitimate transactions in the training data represent a huge variety of ways of using the credit card: for a banana, hotel booking, toaster, car parking, and more!

In the following, we explain how the different resampling methods skew the transactions data, and how this leads to a deterioration in model performance, as we saw before.

Oversampling (SMOTE)

The corresponding model has a low precision value (44%) and it therefore raises many false alarms. The training set contains the original fraudulent transactions plus the synthetically generated fraudulent transactions within the feature space of the fraudulent transactions. For example, if we had one fraudulent transaction that purchases 20 toasters, SMOTE resampling might produce thousands of slightly different fraudulent transactions that all purchase a toaster. Eventually, all legitimate transactions that include a toaster would be predicted as fraudulent. This kind of model is overfitting the training data as illustrated in the figure below.

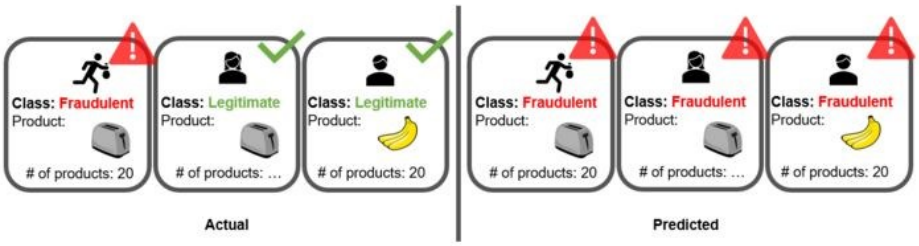


Example class predictions generated by an overfitting fraud detection model trained on SMOTE oversampled data: An arbitrary feature, for example, product=toaster incorrectly determines fraudulent transactions in new data.

Oversampling (bootstrap)

This model performs worse than the baseline model in terms of both recall and precision. The training set contains thousands of exact copies of the original fraudulent transactions. For example, if we had a fraudulent transaction ordering 20 toasters, all legitimate transactions that ordered 20 items of the same product **or** a toaster would be suspicious, because these two features would have characterized so many fraudulent transactions in the oversampled data. At the same time, the model would have failed to generalize on large amounts of the same item as suspicious,

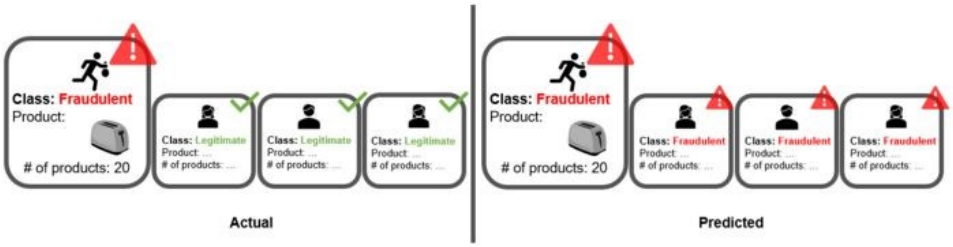
emphasizing instead precisely 20 toasters as suspicious. Also this model is overfitting the training data.



Example class predictions generated by an overfitting fraud detection model trained on Bootstrap oversampled data: an arbitrary set of features, for example, product=toaster and # of products=20 incorrectly determines fraudulent transactions in new data.

Undersampling (bootstrap)

The model performs almost perfectly in terms of recall (92%), yet the worst in terms of precision (2%). Since more than 99% of the original transactions are discarded in the undersampling phase, our training data might only consist of credit card transactions for food and leave out hotel bookings, car parking, and many others. This kind of training data is not representative of the real data. Therefore, almost all transactions are predicted as fraudulent; the model is underfitting.



Example class predictions generated by an underfitting fraud detection model trained on Bootstrap undersampled data: the model predicts all transactions as fraudulent in new data.

Diagnosing Problems of Resampling in Fraud Detection

As our example shows, in this case resampling can't solve the problem of having too few fraudulent transactions in the dataset. However, resampling is shown to lead to performance gain when the a priori distribution is less skewed, for example, in disease

detection[7]. Why is resampling failing then, for this credit card transaction dataset with too few fraudulent transactions in it?

Frauds are perpetrated in a large variety of patterns, and we have only a few fraudulent transactions in our training data. So fraud patterns are definitely under-represented in our training set. Resampling doesn't solve the problem, because it does not increase the variety of the representation of fraudulent transactions, it just replicates in some form the fraud patterns represented in the dataset. Thus, the models trained on resampled data can only perform well in detecting some types of fraud, the types represented in the training data.

In summary, the class of fraudulent transactions is too under-represented (just 0.2% of the whole dataset!) to represent a meaningful description of all the fraud patterns that are out there. Even introducing new similar synthetic fraudulent transactions cannot significantly change the range of the represented fraudulent transactions.

Conclusions

Transactions data are produced in huge volumes every day. In order to build a supervised model for fraud detection, they would need to be labeled. The labeling process, however, in this case, is tricky.

Firstly, even if we had the knowledge to appropriately label fraudulent transactions, the process would be very resource-intensive. Skillful experts at detecting frauds are rare and expensive and usually do not spend their time labeling datasets. Even with reliable and sufficient resources, manual labeling would take a prohibitively long time before a sufficiently large amount of data would be available.

Secondly, expertise in fraud detection is so scarce, because criminals creatively devise ever newer fraud schemes, and it is hard to keep up the pace with the newly introduced patterns. An expert might recognize all types of frauds known till then and still fail at recognizing the new fraud schemes, most recently created.

Finally, and luckily, there are generally fewer fraudulent transactions than legitimate transactions. Even after all this manual effort by extremely skillful people, we might still end up with an insufficient number of data for the fraudulent class.

Those are all reasons why fraud detection is often treated as a rare class problem, rather than an imbalanced class problem.

Yet, we can try. With this dataset, can we artificially increase the sample size of fraudulent transactions by resampling the training data? Not really. Resampling can

---

[7] *"Machine Learning Resampling Techniques for Class Imbalances" Medium, January 2011, https://towardsdatascience.com/machine-learning-resampling-techniques-for-class-imbalances-30cbe2415867. Accessed 14 Oct. 2020.*

improve the model performance if the target classes are imbalanced and yet sufficiently represented. In this case, the problem is really the lack of data. Resampling is subsequently leading to over- or underfitting rather than to a better model performance.

This article just aims at giving you an idea of why, in some cases, resampling cannot work. Of course, better results could be obtained with more sophisticated resampling methods than those we have introduced in this article, like for example, a combination of under- and oversampling[8]. Better results could also be obtained with supervised algorithms other than the decision tree. Some machine learning supervised algorithms, such as logistic regression, are less sensitive to class imbalance than the decision tree, while other algorithms, such as ensemble models, are more robust to overfitting. Even better results could be obtained with the decision tree, for example by applying pruning techniques to avoid the overfitting effect or controlling the tree growth.

However, sometimes we must accept that the data is just not sufficient to describe the minority class. In this case, we must proceed with unlabeled data and try to isolate the events of the rare class via unsupervised algorithms, such as neural autoencoders, isolation forest, and clustering algorithms.

This chapter was first published in KDNuggets.

[8] *How to Combine Oversampling and Undersampling for Imbalanced Classification" Machine Learning Mastery Pty. Ltd., January 2020, https://machinelearningmastery.com/combine-oversampling-and-undersampling-for-imbalanced-classification/. Accessed 14 Oct. 2020.*

Finding an optimal Classification Threshold based on Cost and Profit

Authors: Alfredo Roccato and Maarit Widmann

Workflow on KNIME Community Hub: Finding an Optimal Classification Threshold based on Cost and Profit

Penalizing and rewarding Classification Results with a Profit Matrix

Confusion matrix and class statistics summarize the performance of a classification model: the actual and predicted target class distribution, accuracy of the assignment into the positive class, and the ability to detect the positive class events. However, these statistics do not consider the cost of a mistake, that is, a prediction into the wrong target class.

If the target class distribution is unbalanced, predicting events correctly into the minority class requires high model performance, whereas predicting events into the majority class can easily happen by chance. Wouldn't it be useful to take this into account, and weight the results differently when evaluating the model performance?

Absolutely! However, the final goal of the classification determines whether it makes sense to introduce a cost to certain types of classification results. Cost is useful when incorrect predictions into one target class have more serious consequences than incorrect predictions into the other class(es). Or, put another way, correct predictions into one class have more favorable consequences than correct predictions into the other class(es). For example, not detecting a criminal passenger at the airport security control has more serious consequences than mistakenly classifying a non-threatening passenger as dangerous. Therefore, these two types of incorrect predictions should be weighted differently.

No cost is needed if all target classes are equally interesting or important, and the consequences of a wrong prediction into one target class is as bad as it is for the other classes. This is the case when we predict the color of a wine, for example, or the gender of a customer.

From Model Accuracy to Expected Profit

In addition to accuracy statistics, the performance of a classification model can be measured by expected profit. The profit is measured in a concrete unit defined by the final goal of the classification.

When we use classification results in practice, we assign each predicted class a different treatment: Criminal passengers are arrested, non-threatening passengers are let through. Risky customers are not extended credit, creditworthy customers are! And so on. The most desirable classification results produce profit, such as the security of an airport, or the money that a credit institute makes. We measure this profit in a predefined unit such as the number of days without a terror alarm, or euros. The most undesirable results bring about cost - a terror alarm at the airport, or money lost by a bank - and we measure the cost in the same unit as the profit.

Here, we assess the accuracy and expected profit of a classification model that predicts the creditworthiness of credit applicants. In a credit scoring application, predicting individual customer behavior has a consequence in terms of profit (or loss). Refusing good credit can cause loss of profit margins (commercial risk). Approving credit for high risk applicants can lead to bad debts (credit risk).

Optimizing Classification Threshold

Our goal here is to find a classification model that maximizes the expected profit. A classification model predicts a *positive class score* for each event in the data, which determines the final class prediction. By default the events are assigned to the positive class if their score is higher than 0.5, and otherwise to the negative class. If we change the classification threshold, we change the assignment to the positive and negative class. Consequently, the values of accuracy and expected profit change as well.

Data

In our credit scoring application, we use the well-known German Credit Data Set, as taken from the University of California Archive for Machine Learning and Intelligent Systems.

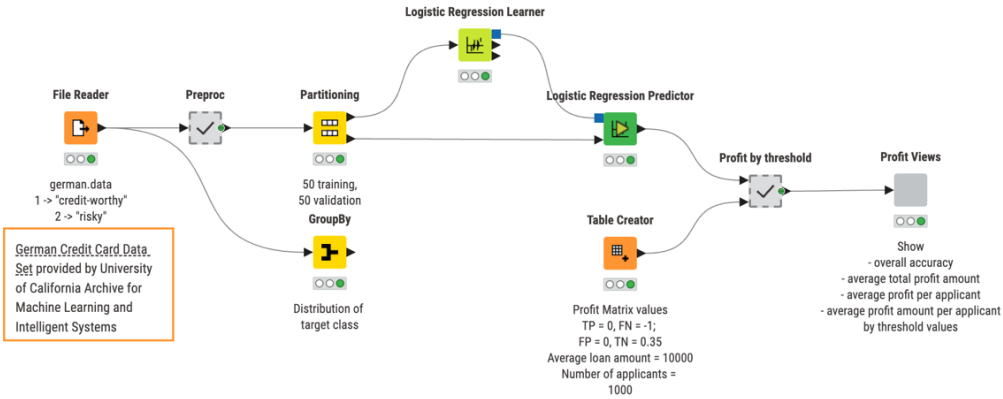
The dataset is composed of 1000 customers. The input variables are the individual characteristics of the subjects, like socio-demographic, financial and personal, as well as those related to the loan, such as the loan amount, the purpose of the subscription, and wealth indicators. The target is the evaluation of the credit applicant's creditability by the bank (2 = risky, and 1 = creditworthy).

In this dataset, 700 applicants (70%) are classified as creditworthy and 300 (30%) as risky.

We refer to the risky customers as the *positive class* and the creditworthy customers as the *negative class*.

Workflow to produce Expected Profit for different Classification Thresholds

The workflow below shows the process of training a classification model for credit scoring and evaluating the expected profit for different classification thresholds.



This workflow, [Finding an Optimal Classification Threshold based on Cost and Profit](), trains a classification model for credit scoring and produces the expected profit for varying values of the classification threshold. The workflow is available for download from the KNIME Community Hub.

The workflow starts with data access and preprocessing. Next, to assess the predictive capabilities of the model, it divides the initial dataset into two tables of equal size, respectively named the training set and the validation set. After that, it trains a logistic regression model on the training set to predict the applicants' creditworthiness as "risky" or "creditworthy".

The "Profit by threshold" metanode predicts the creditworthiness in the validation set for varying classification thresholds, starting with a low value of the threshold and increasing it for each iteration.

Finally, it shows the model performance statistics for different threshold values in an interactive view as produced by the "Profit Views" component.

Before we look at the results, we introduce the profit matrix that we need to transform the accuracy statistics into the expected profit.

Profit Matrix

To evaluate misclassification in terms of expected profit, a profit matrix is requested for assigning cost to undesirable outcomes.

We introduce a negative cost (-1) to the *False Negatives* - risky applicants who are approved a credit - and a positive profit (0.35) to the *True Negatives* - creditworthy applicants who are approved a credit. The table below shows the cost and profit values for these classification results in a profit matrix.

	Predicted class **POSITIVE** (Risky)	Predicted class **NEGATIVE** (Credit-worthy)
Actual class **POSITIVE** (Risky)	**Profit for True Positives** 0	**Profit for False Negatives** -1(Cost)
Actual class **NEGATIVE** (Credit-worthy)	**Profit for False Positives** 0	**Profit for True Negatives** 0.35

Profit matrix that introduces a profit to the classification results: a cost to approved bad credits, and a profit to approved good credits.

The values of cost and profit introduced above are based on the following hypothesis[9]: Let's assume that a correct decision of the bank would result in 35% profit at the end of a specific period, say 3-5 years. If the opposite were true, i.e., the bank predicts that the applicant is creditworthy, but it turns out to be bad credit, then the loss is 100%.

Calculating Expected Profit (Baseline)

The following formulas are used to report the model performance in terms of expected profit:

$$Average\ profit\ per\ applicant\ =\ (1-p) * profit\ +\ p * cost,$$

where p is the share of the positive (risky) class applicants of all data.

$$Average\ amount\ per\ applicant\ =\ Average\ profit\ per\ applicant\ * Average\ loan$$

$$Total\ average\ amount\ = n * Average\ amount\ per\ applicant,$$

where n is the number of credit applicants.

More generally, assuming that the class with negative risk potential is defined as the positive class, an average profit for a classification model with a profit matrix can be calculated using the following formula:

[9] *Wang, C., & Zhuravlev, M. An analysis of profit and customer satisfaction in consumer finance. Case Studies In Business, Industry And Government Statistics, 2(2), pages 147-156, 2014.*

$$Average\ profit\ = \frac{Profit * True\ Negatives - Cost * False\ Negatives}{n},$$

where n is the number of events in the data.

Let's say we have 500 credit applicants with an average loan of 10 000 €. 70% of the applicants are creditworthy and 30% are risky. Then a baseline for the profit statistics without using any classification model is calculated as follows:

$$Average\ profit\ per\ applicant\ = 0.70 * 0.35\ + 0.30 * (-1)\ = -0.055$$

$$Average\ amount\ per\ applicant\ = -0.055 * 10000\ €\ = -550\ €$$

$$Total\ average\ amount\ = 500 * (-550\ €) = -225\ 000\ €$$

If we approve a credit for all of the applicants, the expected loss is 225,000 €.

Calculating Expected Profit (Varying Thresholds)

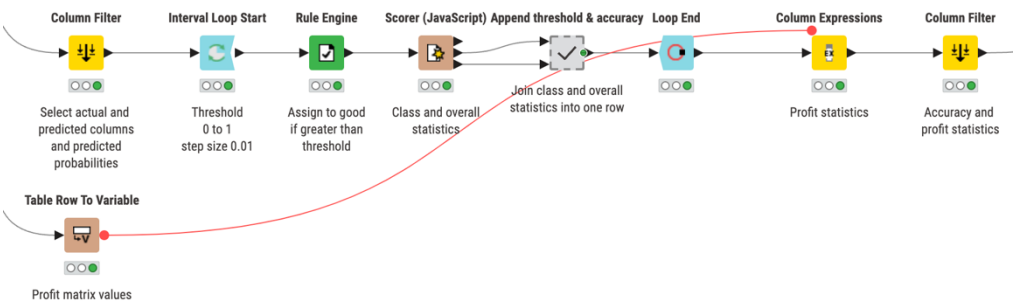
Next, let's calculate what the expected profit is when we evaluate the creditworthiness using a classification model and we weigh the outcomes with the profit matrix.

The minimum threshold for the positive class to achieve non-zero profit can be calculated from the cost matrix as[10]

$$p^* = \frac{-Profit_{TN}}{-Profit_{TN} + Cost_{FN}} = \frac{-0.35}{-0.35 + (-1)} = 0.259$$

This value can be adjusted empirically as described below.

Below you see the workflow inside the "Profit by threshold" metanode. It iterates over different thresholds to the positive class scores and calculates the corresponding accuracy statistics and profit measures.



Producing the accuracy and expected profit for different classification threshold values.

The threshold values range from 0 to 1 with a step size of 0.01. The workflow produces the overall accuracy for each value of the threshold by comparing the actual (unaltered

[10] *C. Elkan. The foundations of cost-sensitive learning. In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pages 973-978, 2001*

in each iteration) and predicted (altered in each iteration) target class values. Furthermore, in order to calculate the expected profit, it weighs the classification results from each iteration by the values in the profit matrix.

The output table of this workflow is shown below:

Accuracy statistics and profit measures for varying thresholds

Rows: 101 | Columns: 11

threshold	Recall	Precision	Sensitivity	Specificity	F-measure	Overall A...	Cohen's ...	Avg_profit...	Avg_amoun...	Avg_amoun
0	1	0.3	1	0	0.462	0.3	0	0	0	0
0.01	1	0.309	1	0.04	0.472	0.328	0.024	0.01	98	98,000
0.02	0.987	0.318	0.987	0.091	0.481	0.36	0.049	0.018	184	184,000
0.03	0.987	0.335	0.987	0.16	0.5	0.408	0.094	0.035	352	352,000
0.04	0.98	0.348	0.98	0.214	0.514	0.444	0.128	0.046	465	465,000
0.05	0.973	0.36	0.973	0.257	0.525	0.472	0.155	0.055	550	550,000
0.06	0.96	0.371	0.96	0.303	0.535	0.5	0.181	0.062	622	622,000
0.07	0.953	0.381	0.953	0.337	0.545	0.522	0.203	0.069	686	686,000
0.08	0.953	0.39	0.953	0.36	0.553	0.538	0.222	0.074	742	742,000
0.09	0.94	0.395	0.94	0.383	0.556	0.55	0.232	0.076	758	758,000
0.1	0.94	0.402	0.94	0.4	0.563	0.562	0.246	0.08	800	800,000
0.11	0.933	0.414	0.933	0.434	0.574	0.584	0.271	0.086	864	864,000
0.12	0.927	0.42	0.927	0.451	0.578	0.594	0.281	0.089	886	886,000

The accuracy statistics and profit measures for varying classification thresholds.

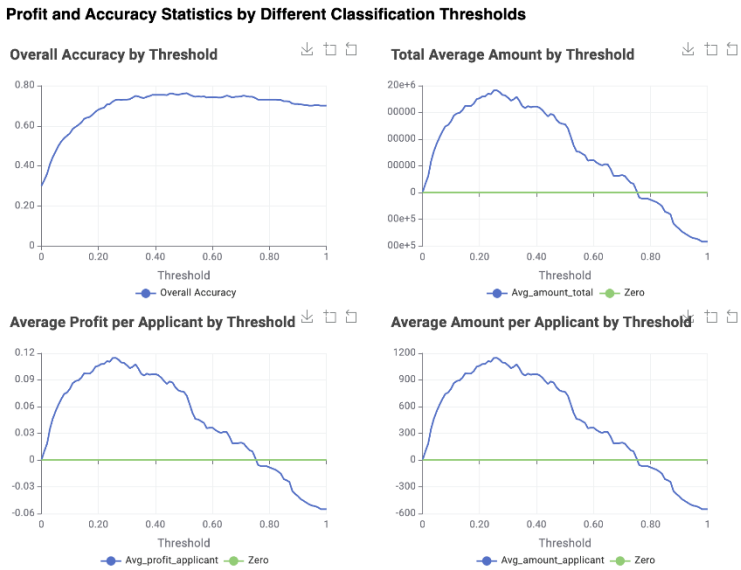
In the output table, every row corresponds to a value of the classification threshold, together with the corresponding model accuracy statistics and profit measures as average profit per applicant, average amount per applicant, and total average amount.

Results

The interactive view below shows the output of the “Profit Views” component. It visualizes the accuracy and profit measures for the varying classification thresholds.

The line plots show how four different model performance indicators develop if the value of the classification threshold increases from 0 to 1. The performance indicators are: 1. Overall accuracy (line plot in the top left corner) 2. Total average amount (line plot in the top right corner), 3. Average profit per applicant (line plot in the bottom left corner), and 4. Average amount per applicant (line plot in the bottom right corner).

Based on an empirical evaluation, the optimal threshold is 0.51 in terms of overall accuracy, and 0.27 in terms of expected profit.



An interactive view to show the development of 1. Overall accuracy, 2. Total average amount, 3. Average profit per applicant, and 4. Average amount per applicant when the classification threshold increases from 0 to 1.

The table below represents the target class distribution (top 2 rows) and the overall accuracy and average profit per applicant (bottom 2 rows) of the credit scoring model, using no model (1st column on the left), and the default and optimized threshold values (3 columns on the right):

		No Prediction	Prediction		
			Threshold 0.27	Threshold 0.50	Threshold 0.51
Actual	Creditable	0.70	0.55	0.72	0.74
	Not Creditable	0.30	0.45	0.28	0.26
	Accuracy		73%	76%	76%
	Profit per applicant	-0.055	**0.113**	0.076	0.072

Expected profit and overall accuracy when creditworthiness is not predicted at all, and when it is predicted using the default and optimized classification thresholds.

Using the optimized threshold 0.27, we can reach 0.113 profit per applicant. This gives an average amount of 1,130 € and, based on 500 applicants, a total average amount of 565,000 €.

The undeniable advantage of using a model is justified by the evidence of 565,000 € versus -225,000€.

Easy Interpretation of a Logistic Regression Model with Delta-p Statistics

Authors: Alfredo Roccato and Maarit Widmann

Workflow on KNIME Community Hub: [Assessing Effects of Single Predictors with Delta-p]

Key Takeaways

- With Delta-p statistics, the predictions based on a logistic regression model are easy to understand by non-technical decision-makers.
- Learn how to calculate the Delta-p statistics based on the coefficients of a logistic regression model for credit application processing.
- Data workflow includes the steps for accessing the raw data to training the logistic regression model, and evaluating the effects of individual predictor columns with Delta-p statistics.
- Keep in mind logistic regression might not be the best choice when working with high dimensional data, with many correlated predictor columns.

Imagine a situation where a credit customer applies for a credit, the bank collects data about the customer - demographics, existing funds, and so on - and predicts the credit-worthiness of the customer with a machine learning model. The customer's credit application is rejected, but the banker doesn't know why exactly. Or, a bank wants to advertise their credits, and the target group should be those who eventually can get a credit. But who are they?

In these kinds of situations, we would prefer a model that is easy to interpret, such as the logistic regression model. Delta-p statistics make interpretation of the coefficients even easier. With Delta-p statistics at hand, the banker doesn't need a data scientist to be able to inform the customer, for example, that the credit application was rejected, because all applicants who apply credit for education purposes have a very low chance of getting a credit. The decision is justified, the customer is not personally hurt, and he or she might come back in a few years to apply for a mortgage.

In this article, we explain how to calculate the Delta-p statistics based on the coefficients of a logistic regression model. We demonstrate the process from raw data to model training and model evaluation with a KNIME workflow where each intermediate step has a visual representation. However, the process could be implemented in any tool.

Assessing the Effect of a single Predictor with the Delta-p Statistics

Logistic Regression Model

When we use the logistic regression algorithm for **classification**, we model the probability of the target class, for example, the probability of a bad credit rating, with a logistic function. Let's say we have a binomial logistic regression model with a target column *y*, credit rating, with two classes that are represented by 0 (good credit rating) and 1 (bad credit rating). The log odds of the target class ($y = 1$) vs. the reference class ($y = 0$) is a linear combination βx of the predictor columns *x* (account balance, credit duration, credit purpose, etc.). A logistic function of βx transforms the log odds into a probability of the target class:

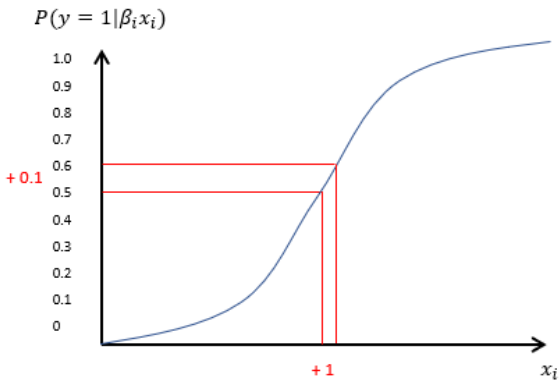
$$P(y = 1 \mid \beta x) = \frac{exp(\beta x)}{1 + exp(\beta x)},$$

where β is the vector of coefficients for the predictor columns x in the logistic regression model that predicts the target class y.

The target and reference classes can be arbitrarily chosen. In our case, the target class is "bad credit rating," and the reference class is "good credit rating."

Delta-p Statistics

If the single predictor column x_i is continuous, the coefficient β_i corresponds to the change in the log odds of the target class when x_i increases by 1. If x_i is a binomial column, the coefficient value β_i is the change in the log odds when x_i increases from 0 to 1. The Delta-p statistics transforms these coefficient values $\beta_i$into values that tell how much the probability of the target class increases or decreases, as shown below:



Logistic function modeling the probability of the target class $y = 1$ as a function of one continuous predictor column x_i.

The Delta-p statistics transforms the coefficient values β_i into percentage effects of single predictor columns to the probability of the target class compared to an average data point e.g., an average credit applicant.

By definition, the Delta-p statistic is a measure of the discrete change in the estimated probability of the occurrence of an outcome given a one-unit change in the independent variable of interest, with all other variables held constant at their mean values. For example, if the Delta-p value of a predictor column x_i is 0.2, then a unit increase in this column (or a change from 0 to 1 in a binomial column) increases the probability of the target class by 20 %.

The following formulas show how to calculate the prior and post probabilities of the target class and, finally, the Delta-p statistics as their difference[11]:

$$P_{prior} = P(y = 1)$$

$$logit_{prior} = ln(\frac{P_{prior}}{1 - P_{prior}})$$

$$logit_{post} = logit_{prior} + \beta_i$$

$$P_{post} = \frac{e^{logit_{post}}}{1 + e^{logit_{post}}}$$

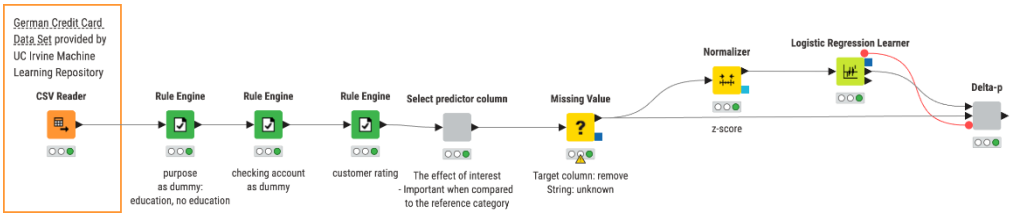
$$Delta - p \ = P_{post} - P_{prior}$$

[11] *Cruce, T. M. (2009). A Note on the Calculation and Interpretation of the Delta-p Statistic for Categorical Independent Variables, Research in High Education, 50(6), 608–622*

Use Case: Effect of Credit Purpose and current Account Balance on Credit Rating

Let's now demonstrate this with an example and check how the credit purpose and balance of an existing account improves or worsens the credit rating. We use the German credit card data provided by the UCI Machine Learning Repository. The dataset contains 21 columns that provide information about demographics and economic conditions of 1,000 credit applicants. Thirty percent of the applicants have a bad credit rating, and 70% have a good rating. You can download the data in *.data* format by clicking "Data Folder" on top of the page and selecting the "*german.data*" item on the next page. The *german.data* file can be opened in a text editor and saved, for example, in csv format. The column names and descriptions of the values in the categorical columns are provided in the german.doc file, accessible via the same "Data Folder" page.

The workflow below shows the process from accessing the raw data to training the logistic regression model and evaluating the effects of individual predictor columns with Delta-p statistics.



This workflow, Assessing Effects of Single Predictors with Delta-p, includes the process from accessing raw credit customer data, to training a credit rating model, and to evaluating the effects of predictor columns to the credit rating with Delta-p statistics. The workflow is available for download from the KNIME Community Hub.

The process is divided into the following steps, each one implemented within a separate colored box: Accessing data (1), preprocessing data as required by a logistic regression model (2), training the model (3), and calculating the Delta-p statistics based on the model coefficients (4). In the preprocessing step, we convert the target column from the 1/2 notation to "bad"/"good." We also transform two originally multinomial columns into binomial columns: We encode the "checking" column into two values "negative"/"some funds or no account" based on the status of the existing bank account. We encode the "purpose" column into values "education"/"no education" to assess the effect of education as a credit purpose. Finally, we handle missing values and normalize the numeric columns in the data.

Below, you can see the coefficient statistics of the logistic regression model, reproducible in any tool. The "Coeff." column shows the coefficient values for the different predictor columns, 0.683 for purpose=education. The "P>|z|" column shows the p-values of the coefficients, 0.055 for purpose=education. This means that

education as a credit purpose increases the probability of a bad credit rating, since the coefficient value is positive, and this effect is significant at 90 % significance level, since the p-value is smaller than 0.1.

Logit *String*	Variable *String*	Coeff. *Number (double)*	Std. Err. *Number (double)*	z-score *Number (double)*	P>|z| *Number (double)*
bad	checking=some funds or no account	-1.063	0.18	-5.912	0
bad	duration	0.31	0.108	2.886	0.004
bad	history=A32	0.885	0.249	3.56	0
bad	history=A33	0.749	0.313	2.392	0.017
bad	history=A30	1.665	0.408	4.082	0
bad	history=A31	1.626	0.414	3.932	0
bad	purpose=education	0.683	0.355	1.921	0.055
bad	amount	0.307	0.118	2.6	0.009
bad	savings=A61	0.994	0.247	4.024	0
bad	savings=A63	0.296	0.425	0.696	0.486
bad	savings=A64	-0.187	0.534	-0.351	0.726
bad	savings=A62	0.882	0.325	2.716	0.007
bad	employed=A73	0.143	0.243	0.591	0.554

Coefficient statistics of a logistic regression model that predicts the credit rating good/bad of a credit applicant.

By looking at the coefficient statistics of the logistic regression model, we find out that education as a credit purpose increases the probability of a bad credit rating compared to other credit purposes. In addition, the coefficient value 0.683 tells that the log odds ratio for getting a bad credit rating with/without education as the credit purpose is 0.683, and the odds ratio of the two groups is $e^{0.683}$=1.979. What would this mean, for example, in a group of 100 credit applicants, let's say 20 of them with education as the purpose (group 1) and the remaining 80 with another purpose (group 2)? If 10 out of the 80 applicants in the group 2 have a bad credit rating, so their odds is 0.125, then according to the odds ratio 1.979, the odds for the group 1 must be ~2 times the odds of the group 2, so 0.25 in this case. Therefore 5 (a quarter) of the applicants in the group 1 must have a bad credit rating!

The coefficient statistics have a universal scale, and we can use them to compare the magnitude and the effect of different predictor columns. However, to understand the effect of a single predictor, the Delta-p statistics provide an easier way! Let's take a look:

Variable *String*	Coeff. *Number (double)*	Std. Err. *Number (double)*	P>|z| *Number (double)*	ODDS_RAT... *Number (double)*	Logit_Prior *Number (double)*	Logit_Post *Number (double)*	Prior *Number (double)*	Post *Number (double)*	Delta-P *Number (double)*	Sig. *String*
checking=some...	-1.063	0.18	0	0.345	-0.847	-1.91	0.3	0.129	-0.171	***
duration	0.31	0.108	0.004	1.364	-0.847	-0.537	0.3	0.369	0.069	**
history=A32	0.885	0.249	0	2.424	-0.847	0.038	0.3	0.51	0.21	***
history=A33	0.749	0.313	0.017	2.114	-0.847	-0.099	0.3	0.475	0.175	*
history=A30	1.665	0.408	0	5.287	-0.847	0.818	0.3	0.694	0.394	***
history=A31	1.626	0.414	0	5.086	-0.847	0.779	0.3	0.686	0.386	***
purpose=educa...	0.683	0.355	0.055	1.979	-0.847	-0.165	0.3	0.459	0.159	
amount	0.307	0.118	0.009	1.359	-0.847	-0.54	0.3	0.368	0.068	**
savings=A61	0.994	0.247	0	2.701	-0.847	0.146	0.3	0.537	0.237	***
savings=A63	0.296	0.425	0.486	1.345	-0.847	-0.551	0.3	0.366	?	
savings=A64	-0.187	0.534	0.726	0.829	-0.847	-1.034	0.3	0.262	?	
savings=A62	0.882	0.325	0.007	2.416	-0.847	0.035	0.3	0.509	0.209	**

Delta-p statistics, its intermediate results, and the corresponding coefficient statistics of a logistic regression model that predicts the credit rating good/bad of a credit applicant.

In the output table below you can see the Delta-p statistics and the intermediate results in calculating it, also shown below for the purpose=education variable:

$$P_{prior} = P(y = 1) \ = \ 0.3$$

$$logit_{prior} = ln(\frac{P_{prior}}{1 - P_{prior}}) \ = ln(\frac{0.3}{1 - 0.3}) = \ -0.847$$

$$logit_{post} = logit_{prior} + \beta_i \ = \ -0.847 \ + \ 0.683 \ = \ -0.165$$

$$P_{post} = \frac{e^{logit_{post}}}{1 + e^{logit_{post}}} = \frac{e^{-0.165}}{1 + e^{-0.165}} = 0.459$$

$$Delta - p \ = P_{post} - P_{prior} = 0.459 - 0.3 \ = \ 0.159$$

The value 0.159 of the Delta-p statistics indicates that education as a credit purpose increases the probability of a bad credit rating by 15.9 % compared to an average credit application.

If we wanted to compare the effect to the opposite situation, i.e., the credit purpose is not education, instead of an average credit applicant, we would need to recalculate the prior probability and also mean-center the binomial values of the predictor column of interest x_i. In our data, 5 % of the people apply the credit for education purposes, so the mean of the "purpose" column x_i is 0.05.

$$logit_{prior} = ln(\frac{P_{\underline{y}}}{1 - P_{\underline{y}}}) \ + \ \beta_i(0 - \underline{x_i}) \ = \ -0.847 + 0.683 * (-0.05) = \ -0.881$$

$$logit_{post} = ln(\frac{P_{\underline{y}}}{1 - P_{\underline{y}}}) \ + \ \beta_i(1 - \underline{x_i}) \ = \ -0.847 \ + \ 0.683 * 0.95 \ = \ -0.199$$

$$P_{prior} = \frac{e^{logit_{prior}}}{1+e^{logit_{prior}}} = \frac{e^{-0.881}}{1+e^{-0.881}} = 0.293$$

$$P_{post} = \frac{e^{logit_{post}}}{1 + e^{logit_{post}}} = \frac{e^{-0.199}}{1 + e^{-0.199}} = 0.45$$

$$Delta - p \ = P_{post} - P_{prior} = 0.45 - 0.293 \ = \ 0.158$$

The value 0.158 of the Delta-p statistics indicates that the credit applied for education purposes increases the probability of a bad credit rating by 15.8 % compared to those who apply it for other purposes. There's hardly any difference to the previous situation where we compared against an average applicant and obtained the Delta-p value 0.159. This means that the credit applicants with other purposes than education are very close to the sample average in terms of their credit rating, apparently because they make up 95% of the total sample.

Now we know that applying credit for education purposes has a negative effect on the credit rating. Which column could have a positive effect? Let's check the effect of the other dummy column that we created, the "checking" column that tells if the balance

of the existing account is negative. The coefficient value of checking=some funds or no account is -1.063 with a p-value 0, as you can see in the first row in the coefficient statistics table.

The Delta-p statistics -0.171 in the first row of the output table above show that credit applicants with no negative account balance tend to have a 17.1 % lower probability of a bad credit rating than an average credit applicant. Interestingly, we found two columns, purpose and checking, that have an effect of almost the same size but a different direction. If we look at the odds ratio of these two variables, we wouldn't get the same information at first glance: The odds ratio is 0.345 for checking=some funds or no account and 1.979 for purpose=education.

Conclusions

In this article, we have introduced Delta-p statistics as a straightforward way of interpreting the coefficients of a logistic regression model. With Delta-p statistics, the predictions based on a logistic regression model are easy to understand by non-technical decision-makers.

We used Delta-p statistics to assess the individual effects that make a credit application succeed or fail. Of course, the use cases of Delta-p statistics are many more. For example, we could use Delta-p statistics to determine the individual touchpoints that decrease or increase the customer satisfaction the most, or to find the symptoms with the highest relevance, when detecting a disease. Also notice that not always the whole process from raw data to model training and model evaluation need to be completed, Delta-p statistics can also be used to re-evaluate the coefficients of a previously trained logistic regression model.

Delta-p statistics can only be used to assess the individual effects of predictor columns in a logistic regression model. Logistic regression might not be the best choice when working with high dimensional data, with many correlated predictor columns, and columns not correlated with the target column. The target classes also need to be linearly separable in the feature space.

If you want to replicate the procedure described in the article, one option is to install the open source KNIME Analytics Platform on their laptops and download the KNIME workflow attached to the article for free. A visual representation of the workflow is available on the KNIME Community Hub without installing KNIME Analytics Platform. Other options are to implement the calculations in any another programming tool, or even perform them manually with a calculator.

This chapter was first published in InfoQ.

Topic Index

B

Binary classification inspector..... 23

C

Calculate expected profit 53
Churn prediction 15
Classification on imbalanced datasets .. 25
Cohen's kappa 33
Credit rating 60
Cumulative gain chart 21

D

Delta-p 57
Delta-p statistics..... 58

E

Email classification 1

F

Fraud detection..... 27

L

Lift chart..... 20
Logistic regression 57
Logistic regression model..... 58

M

Mean absolute error 8
Mean absolute percentage error..... 8
Mean signed difference..... 8
Multivariate Classification Model 5

N

Numeric scoring metrics 8

O

Optimal classification threshold..... 50
Oversampling 26
Oversampling (SMOTE)..... 46
Oversampling bootstrap 46

P

Profit matrix..... 53

R

Resampling..... 41
Resampling techniques..... 26
ROC curve 17
Root mean squared error 8
R-squared.....8

S

Sensitivity 4, 5
SMOTE 26
Specificity 4, 5

U

Undersampling 26
Undersampling bootstrap 47

V

Visual scoring techniques..... 15

Scoring Metrics
Evaluating Machine Learning Models

Machine learning models can automate different kinds of processes - prove customer creditworthiness, flag emails as spam, detect fraudulent transactions, forecast weather, optimize the electricity supply, and more! The overarching goal of all these applications is to have accurate predictions. But how do we define “accurate”?

This book explains different model evaluation - or scoring - techniques that show the model’s performance in a broader context, introducing the perspectives of single metrics and their robustness to the properties of data, and demonstrating the concrete value of a careful model validation.

Maarit Widmann is a data scientist at KNIME. She works in the Evangelism team, communicating concepts behind data science and presenting solutions in blog posts, videos, online/onsite presentations, and courses. She’s the author behind the KNIME self-paced courses and the co-author of the time series course. She holds her Bachelor’s degree in social sciences, and the Master’s in Data Science.

Alfredo Roccato is an independent consultant and trainer with a focus on data science. He studied statistics at the Catholic University of Milan. Since the 80s he has been working for companies on projects related to Business Intelligence and Business Analytics.