Technische Hochschule Rosenheim

Modul - Introduction to AI - part II (AI2)

Bachelor Programme AAI

03 - Classifier Evaluation

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Vers. 10.4.2022

Goals

Learn different performance metrics for classification

- Confusion Matrix
- Accuracy
- Sensitivity
- Specificity
- Precision
- F score
- Informedness
- Markedness
- Mathews Correlation Coefficient

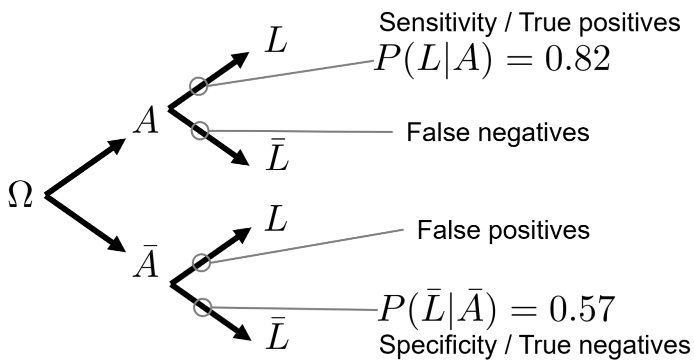
Performance Metrics

Technische Hochschule Rosenheim

- Large variety of metrics available
- Choose the right metrics depending on:
  - Dataset attributes
  - Classifier goal (Regression, Classification,...)
- For binary classification:
  - **P**: All positive samples in the dataset
  - **N**: All negative samples in the dataset

True positive/negative and False positive/negative

- The probability of increased white blood cells when a person has appendicitis is $P(L|A) = 0.82$ (Sensitivity of the test)
- The probability of normal white blood cell concentration if a person has no appendicitis is $P(\neg L|\neg A) = 0.57$ (Specificity of the test)



Can be measured during experiments, universal value for the test

Confusion Matrix

- For Classification problems
- Basis for advanced metrics
- False Positive: Type 1 error
  - The test erroneously classifies a sample as positive
- False Negative: Type 2 error
  - The test erroneously classifies a sample as negative

Example:

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

Sensitivity, recall, hit rate, True Positive Rate (TPR)

- Q: How many P were found?
- Only evaluates P-class

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

Example:

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

Lets say a perfect test would result in this confusion matrix

10	
	90

But we use a test that always classifies as **true** instead.

10	90

$$TPR = \frac{TP}{P} = \frac{10}{0 + 10} = 100\%$$

Specificity, selectivity, True Negative Rate (TNR)

- Q: How many N were found?
- Only evaluates N-class

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$$

Example:

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

Lets say a perfect test would result in this confusion matrix

$$\begin{array}{c|c} 10 & \\ \hline & 90 \end{array}$$

But we use a test that always classifies as **false** instead.

$$\begin{array}{c|c} & \\ \hline 10 & 90 \end{array}$$

$$TNR = \frac{TN}{N} = \frac{90}{0 + 90} = 100\%$$

Sensitivity vs. Specificity

- Sensitivity and Specificity belong together and need to be combined for a judgement

Lets say a perfect test would result in this confusion matrix

10	
	90

$$TPR = 100\%$$

$$TNR = 100\%$$

But we use a test that always classifies as **true** instead.

10	90

$$TPR = \frac{TP}{P} = \frac{10}{0+10} = 100\%$$

$$\boldsymbol{TNR} = \frac{\boldsymbol{TN}}{\boldsymbol{P}} = \frac{\mathbf{0}}{\mathbf{0+90}} = \mathbf{0\%}$$

Lets say a perfect test would result in this confusion matrix

10	
	90

But we use a test that always classifies as **false** instead.

10	90

$$TNR = \frac{TN}{P} = \frac{90}{0+90} = 100\%$$

$$\boldsymbol{TPR} = \frac{\boldsymbol{TP}}{\boldsymbol{P}} = \frac{\mathbf{0}}{\mathbf{0+10}} = \mathbf{0\%}$$

Accuracy

- Q: How many correct classifications?
- Simple but widespread
- Heavily influenced by the dataset□very misleading

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$$

Example:

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

$$ACC = \frac{correctly\ classified}{all\ samples} = \frac{TP + TN}{P + N}$$

Accuracy, easy to trick

Technische Hochschule **Rosenheim**

- Example HIV-Test
- ~90.000 Infected in GER
- 80.000.000 Healthy people

$$ACC = \frac{correctly\ classified}{all\ samples} = \frac{TP + TN}{P + N}$$

```python
def hiv_classifier(data):
    return False
```

A very bad classifier, but with great accuracy!

90k	80M

$$ACC = \frac{0+80.000.000}{80.000.000+90.000} = 0.998876$$

So, accuracy combines true positives and true negatives, but can be misleading if the dataset is imbalanced.

Precision, Positive/Negative Predictive Rate (PPR/NPR)

Technische Hochschule Rosenheim

- Q: How pure is the positive/negative result?
- Only evaluates positive/negative predictions

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

$$PPR = \frac{TP}{TP + FP} \qquad NPR = \frac{TN}{FN + TN}$$

Lets say we use a coin flip for classification

25	25
25	25

But we test on a set of only **positives**

50	
50	

$$PPV = 100\%$$

F score, F1 score, F measure (F1)

- Combines sensitivity (TPR) and precision (PPV) in one value
- Can show similar issues as Accuracy on imbalanced datasets

$$F_1 = 2\frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$$

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

We classify always true and have 99 cats and one dog in the test-set

99	1
0	0

$$F_1 = \frac{2*99}{2*99+1+0} = 0.995$$

$$TNR = \frac{TN}{N} = \frac{0}{1} = 0\%$$

$$TPR = \frac{TP}{P} = \frac{99}{99} = 100\%$$

Informedness, Bookmaker Informedness (BM)

- Combines sensitivity (TPR) and specificity (TNR) in one value
- Avoids problems on imbalanced datasets

$$BM = TPR + TNR - 1$$

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

$$TNR = \frac{TN}{N} = \frac{0}{1} = 0\%$$

$$TPR = \frac{TP}{P} = \frac{99}{99} = 100\%$$

We classify always true and have 99 cats and one dog in the test-set

99	1
0	0

$$BM = 1 + 0 - 1 = 0\%$$

Markedness (MK)

Technische Hochschule Rosenheim

- Combines precision (PPR) and NPR in one value
- Avoids problems on imbalanced datasets
- „Informedness“ of the negative class

$$MK = PPR + NNR - 1$$

$$PPR = \frac{TP}{TP + FP} \; and \; NPR = \frac{TN}{FN + TN}$$

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

We classify always true and have 99 cats and one dog in the test-set

99	1
0	0

$$MK = 99\% + 0\% - 100\% = -1\%$$

Matthews correlation coefficient (MCC)

- Correlation between prediction and observation
- Works well on imbalanced datasets
- Somehow mixes all together.

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

$$MCC = \frac{TP \cdot TN + FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

What to do for multiple classes?

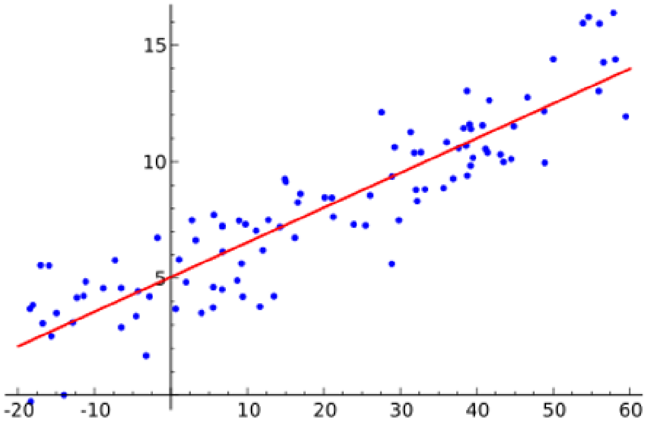
- MPCA: Mean Per Class Accuracy
- MPCE: Mean Per Class Error
    - Calculate metric per class
    - Calculate mean over n classes

		Actual Class	
		Cat (P)	Not a cat (N)
Predicted Class	Cat	True Positive (TP)	False Positive (FP)
	Not a cat	False Negative (FN)	True Negative (TN)

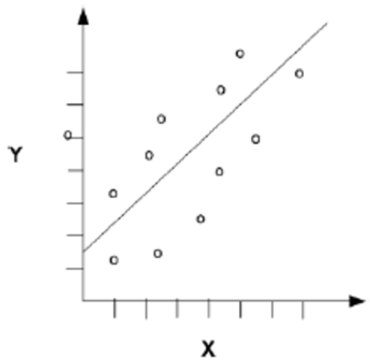
$$MPCA = \frac{1}{n} \sum_i ACC_i$$

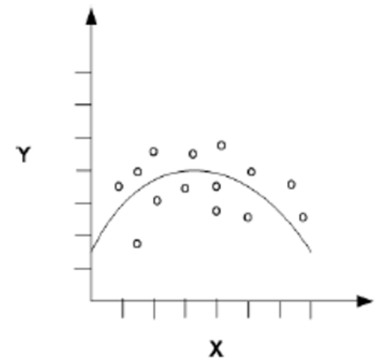
What to do without classes?

- Continuous values
- E.g. when using linear Regression
  - Predicting the rent for a flat from other flats using m² and rent.
- Or forecasting in general
  - Predicting the temperature tomorrow from weather data of tody.



Regression





Objective: find relationship between (correlated) input variables x_i & output variables y_i

The dataset is $D = \{(\vec{x_i}, y_i)\}_{i=1}^{n} \subset X \times Y$ where each data point is a pair of input variables $\vec{x_i} \epsilon\ R^N$ & the corresponding output $y_i \epsilon\ R$

The hypothesis space is the set of all functions from the input space X to the output space Y; i.e.,

F = {f | f : X → Y}

A common restriction is to just consider the set of linear mappings from X to Y as parametrized by w and b

Regression Types

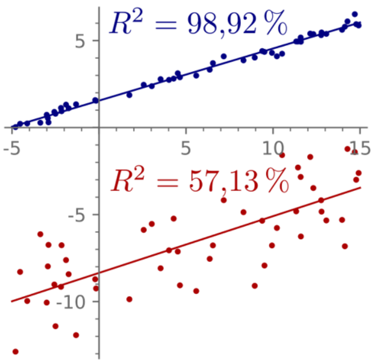
- Simple regression: If there is a single independent variable x with the help of which a numerical, dependent variable y can be predicted (*predict*), then we speak of a **Simple Regression**.
    - If there is a linear relationship, we speak of a **Simple Linear Regression**.
- **Multivariable regression**: If there are several independent variables X with the help of which a numerical, dependent variable y can be predicted (*predict*), then we speak of **Multivariable Regression** (in the linear case of **Multiple Linear Regression**).

Coefficient of determination (R^2)

Technische Hochschule Rosenheim

- How well does the model predict/describe the dependent variable

  - $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ *Mean*
  - $SQR = \sum_{i=1}^{n}(x_i - y_i)^2$ ***S**um of **S**quares **R**esidual*
  - $SQT = \sum_{i=1}^{n}(x_i - \bar{x})^2$ ***S**um of **S**quares **T**otal*



  - $R^2 = 1 - \frac{SQR}{SQT}$ Deviations from prediction *Divided by* Deviations from mean in reality
  - Interpretation: $R^2 = 0.67$ → 67% of the variability is fitted well to the model.

Solve the follwoing equation...

$$y = m * X + b$$

- X is Independent Variable of size i
- Y is Dependent Variable
- b is intercept (mnemonic : ‘b’ means where the line begins)
- m is slope (mnemonic : ‘m’ means ‘move’)

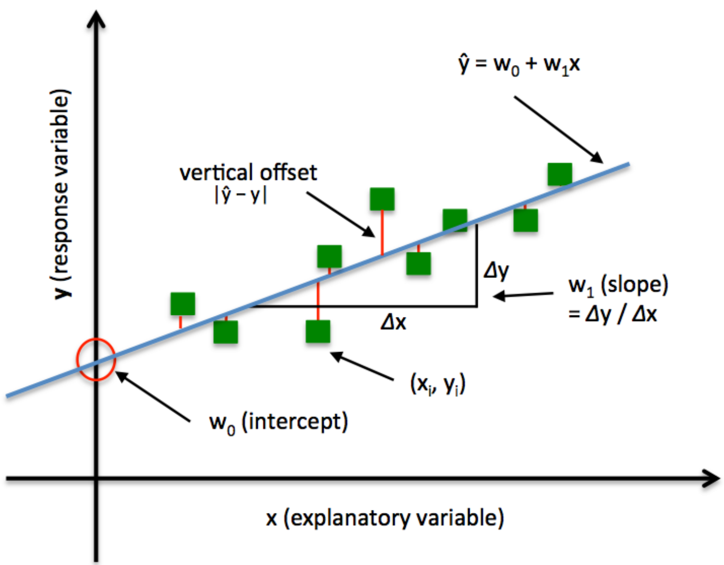
Use:

- **Cost** or **Loss** Function
  - Mean Squared Error - MSE
  - Mean Absolute Error - MAE
  - Mean Absolute Percentage Error - MAPE
- Gradient Descent

Regression Goal

Goal: Minimize the errors of residuals (vertical offset)

- To define and measure the error of our model we define the cost function MSE as the sum of the squares of the residuals.



Cost-Function: MAE

Mean Absolute Error (MSE)

$$MAE = \frac{1}{n} \sum_{i} |\tilde{y}_i - y_i|$$

- n is the total number of observations (data points)
- y_i = expected value (the original value)
- $\tilde{y}_i$ = the calculated value with random b and m

Sum the error difference over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points.

Cost-Function: MAPE

Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{1}{n} \sum \left| \frac{y_i - \tilde{y}_i}{y_i} \right|$$

- n is the total number of observations (data points)
- y_i = expected value (the original value)
- $\tilde{y}_i$ = the calculated value with random b and m
- Multiply by 100 for percentage values

Cost-Function: MSE

Mean Square Error (MSE)

$$MSE = \frac{1}{n} \sum (\tilde{y}_i - y_i)^2$$

- n is the total number of observations (data points)
- y_i = expected value (the original value)
- $\tilde{y}_i$ = the calculated value with random b and m

Square the error difference and sum over all data points and divide that value by the total number of data points. This provides the average squared error over all the data points.

> Weighted error: Many small errors become irrelevant and few large errors are heavily weighted

Task

Find the **mean square error (MSE)** of the following two sets of numbers:

```
Yo = [2, 5, 9, 2]  # original values
Yc = [6, 3, 6, 1]  # calculated values
```

Try it in Python!

Open a Jupyter Notebook and do the math!

Solution

First we calculate the differences between these numbers:

```
D = [2 - 6, 5 - 3, 9 - 6, 2 - 1]
=>  D = [-4, 2, 3, 1]
```

Now we square them:

```
D = [-4 * -4, 2 * 2, 3 * 3, 1 * 1]
=>  D = [16, 4, 9, 1]
```

next we find the mean of these numbers:

```
mse = (16 + 4 + 9 + 1) / 4
mse = 30 / 4
mse = 7.5
```

... or in Python

```python
import numpy as np

Yo = [2, 5, 9, 2]  # original values
Yc = [6, 3, 6, 1]  # calculated values

mse = np.sum([(o-c)**2 for o,c in zip(Yo,Yc)])/len(Yc)

print(mse)
```

Gradient Descent

Technische Hochschule Rosenheim

- We have gotten an error value using the cost function (MSE)
- The next important concept needed to understand linear regression is **Gradient Descent (GD)**
- Gradient descent is a method of updating *m* and *b* to reduce the cost function(MSE)
- The idea is:
    1. We start with some values for *m* and *b*
    2. and then we change these values iteratively to reduce the cost (MSE)
- To update *m* and *b*, we take the gradients from the cost function
- To find these gradients, we take partial derivatives with respect to *m* and *b*

$$b = b - \eta \frac{1}{n} \sum_{i=0}^{n} (y_i - \tilde{y}_i)$$

$$m = m - \eta \frac{1}{n} \sum_{i=0}^{n} (y_i - \tilde{y}_i) x_i$$

with $\eta < 1$ is learning rate

Gradient Descent

$$\tilde{y}_i = m X_i + b \qquad i = \#\ \text{data points}$$

$$\Rightarrow \tilde{y} = \underset{\overset{\uparrow}{n}}{b}\,\underset{\overset{\uparrow}{n}}{X} + b_0 = \underset{\overset{\uparrow}{n+1}}{B}\,\underset{\overset{\uparrow}{n}}{\tilde{X}} \qquad \text{mit } B = \begin{pmatrix} b_0 \\ b \end{pmatrix} \text{ und } \tilde{X} = \begin{pmatrix} 1 \\ x \end{pmatrix}$$

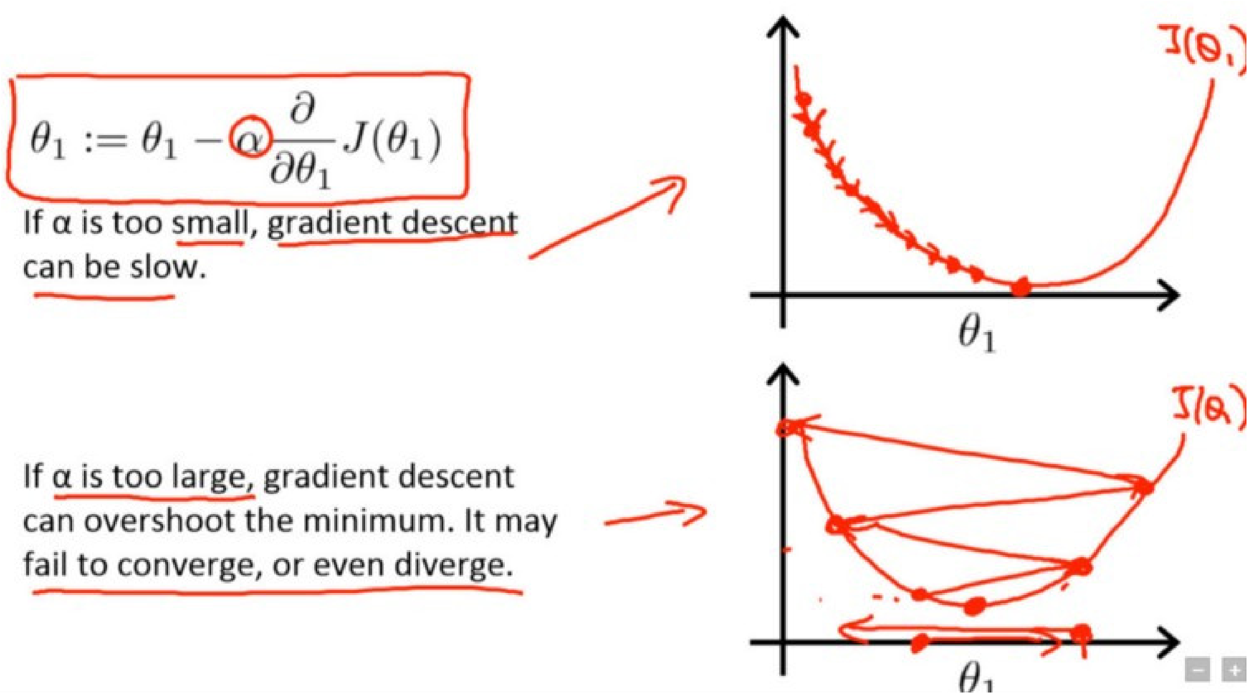
$$= b_0 \cdot 1 + b_1 \cdot x_1 + b_2 \cdot x_2 + \dots + b_n x_n$$

Aus $$MSE(B) = \frac{1}{m} \sum_{i=0}^{m} (y_i - \tilde{y}_i)^2$$

$$\frac{\partial MSE(b_j)}{\partial b_j} = 2 \cdot \frac{1}{m} \sum_{i=0}^{m} (y_i - \tilde{y}_i) \cdot \tilde{x}_i^j$$

$$\Rightarrow \boxed{b_j = b_j - \frac{\alpha}{m} \sum_{i=0}^{m} (y_i - \tilde{y}_i)\, \tilde{x}_i^j}$$

GD-Explanantion



Nice web page: https://developers.google.com/machine-learning/crash-course/fitter/graph

... or in Python

```
import numpy as np

def linear_regression(X,Y, iterations=1000, eta=0.01 ):

    m = 0.0
    b = 0.0

    for j in range(iterations):
        b = b + eta * np.sum([(o-(m*x+b)) for o,x in zip(Y,X)])/len(Y)
        m = m + eta * np.sum([(o-(m*x+b))*x for o,x in zip(Y,X)])/len(Y)

    return m,b

Y = [1,4,3,4,9,21]
X = [1,2,3,4,10,20]

m,b = linear_regression(X,Y)

print("Slope {:.2f} and intersection {:.2f}" .format(m,b))
```

Summary

Technische Hochschule Rosenheim

- What are the classic evaluation techniques for classification
- The more the values of a confusion matrix are aggregated, the harder is an interpretation
- The application domain is required to evaluate a classifier
- How to calcuate an error?
  - Cost functions: MSE, MAE and MAPE
- Linear Regression
  - Check also: Linear_Regression.ipynb
  - or on Colab: https://drive.google.com/file/d/1SgKxD6bvo5LAVNRMlQ6rMvr-k_XGo_ZT/view?usp=sharing