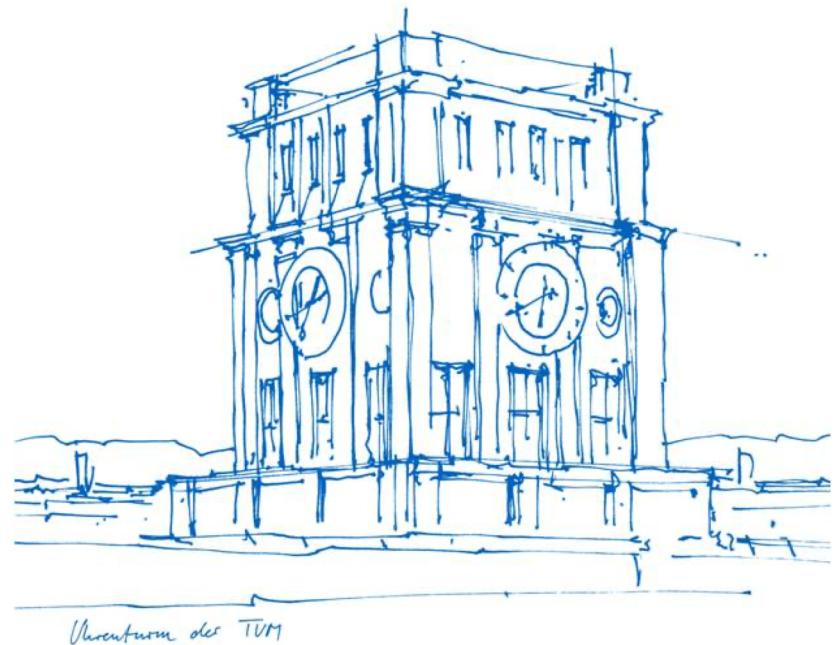


Exercises for Social Gaming and Social Computing (IN2241 + IN0040) – Introduction to

Exercise Sheet 4 Sentiment Analysis in DotA



Exercise Content

Sheet Number	Exercise	Working Time
1	<ul style="list-style-type: none">Introduction to Python: basic Python programming language exercisesGraph Drawing using network, introduction to Steam API	Monday, May 27 - Monday, June 3, 24:00
2	<ul style="list-style-type: none">Centrality measures	Monday, June 3 - Monday, June 17, 24:00
3	<ul style="list-style-type: none">Collaborative Filtering Recommender System using Steam data	Monday, June 17 - Monday, June 24, 24:00
4	<ul style="list-style-type: none">Trying to infer toxic behavior from game data in DotA 2	Monday, June 24- Monday, July 01, 24:00
5	<ul style="list-style-type: none">Clustering EVE and BF2 players according to Radoff's motivational player types using K-means	Monday, July 01 - Monday, July 08, 24:00
6	<ul style="list-style-type: none">analyzing short-term social context using mobile interaction data (Reality Mining)	Monday, July 08 - Monday, July 15, 24:00

Exercise Sheet 4: Sentiment Analysis in DotA

- **goal:** analyse DotA chat logs and use **sentiment analysis** to find out whether a **negative attitude** affects a **player's win rate**
- **DotA**
 - two teams of **5 players** (that usually **don't** know each other)
 - each team tries to **destroy** the enemy base
 - the nature of the game does **provoke negativity** at times
 - and we want to try to **predict** it

The Data: Your friends' Steam library

- information from **1.500 matches** played during December 2016, split in the following files:
 - *chat.csv*: **what** was said in the chat between teams, **when** it was said and **which player** said it
 - *match.csv*: detailed **statistics** for every **match**
 - *players.csv*: detailed **statistics** for every **player**
 - *player times.csv*: **gold & exp.** for every player each minute of every game
 - *labels.xlsx*: a sample of **labeled** chat used for the **sentiment analysis**

Task 4.1: Your first sentiment analysis

Sentiment analysis, sometimes called **opinion mining**, is a method to derive information from the text that allows for a classification as neutral, positive or negative. It is a **semi-supervised process**, meaning that you need a small set of **labeled data** to train your machine learning model on in order to use it on another set of unlabeled data. Without the labeled set it would be difficult for your AI to know what exactly makes a statement positive or negative. Some of its many practical applications are the analysis of customer reviews, social media comments or survey responses.

Note: We will use a **random forest classifier** in this task.

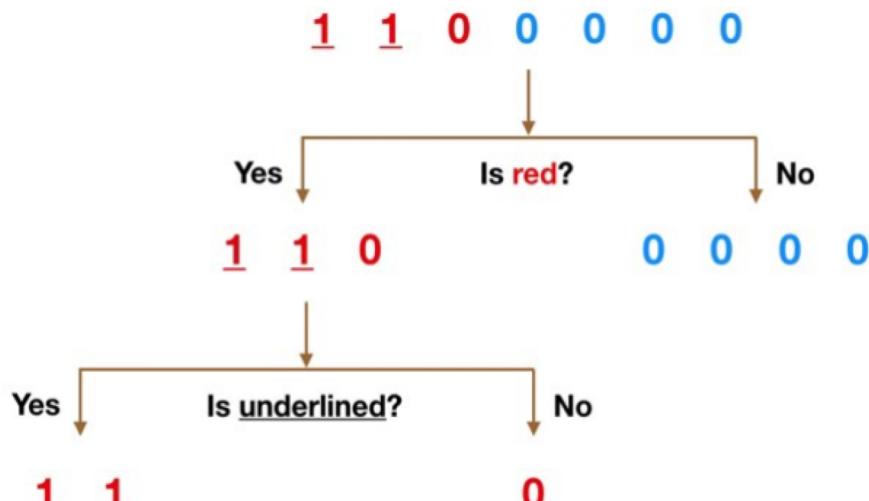
Your task is to **train a model** using the labeled data, then use that model to **predict the sentiments** of the whole chat. Let us start with the basics:

Sentiment Analysis

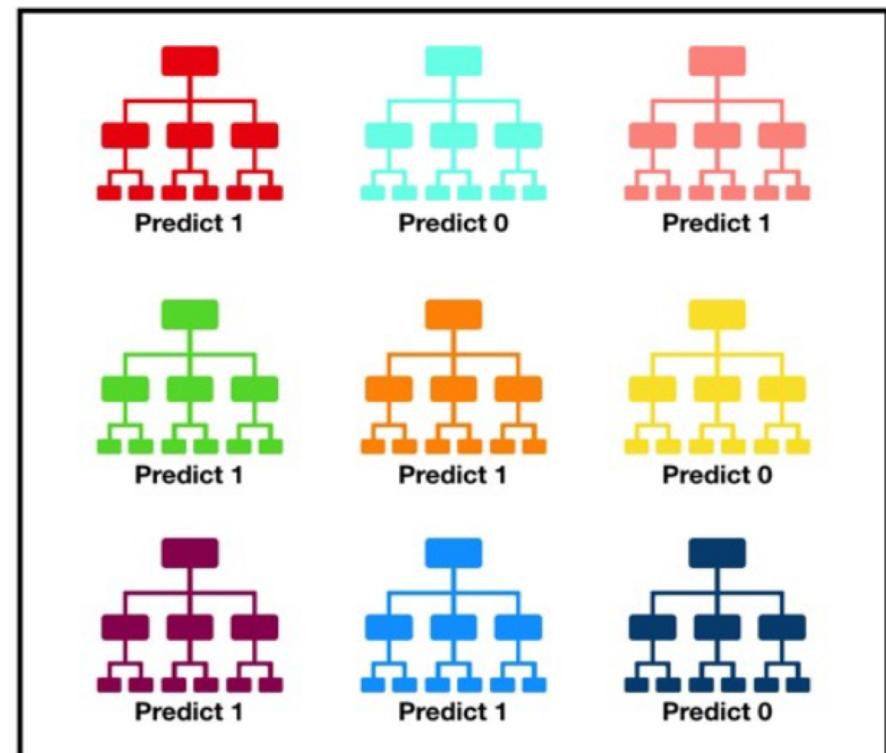
- **input**: text (e.g. tweets, comments, reviews)
- **goal**: classify sentiment of the text (usually neutral, negative or positive)
- **means**: machine learning model
- **requirements**: pre-labeled text (called “train-set”).
 - **labels.xlsx** in our case

Sentiment Analysis: Machine Learning

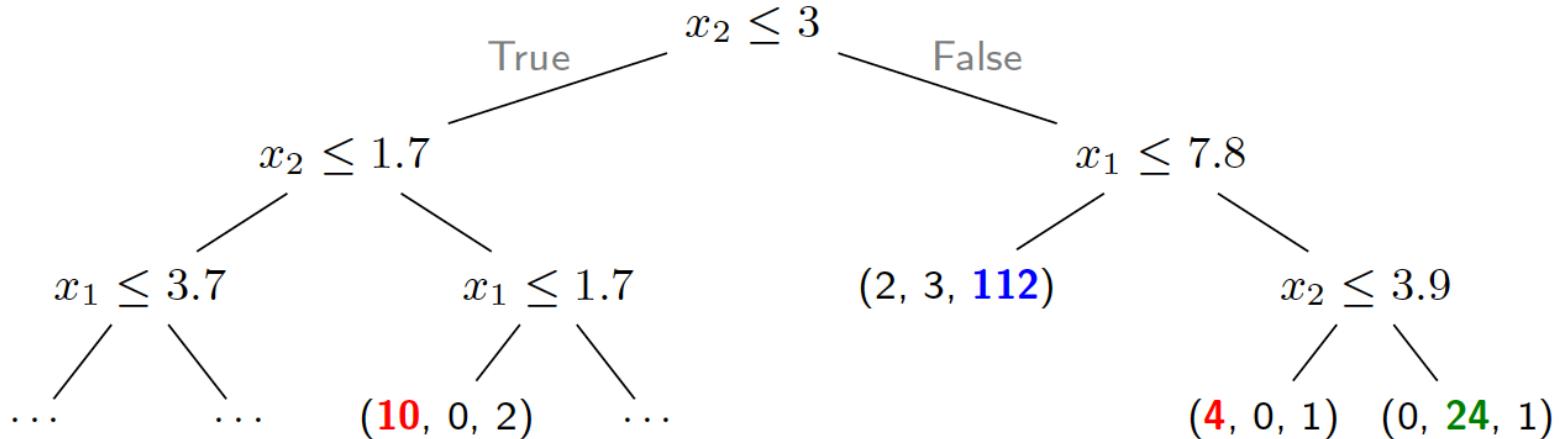
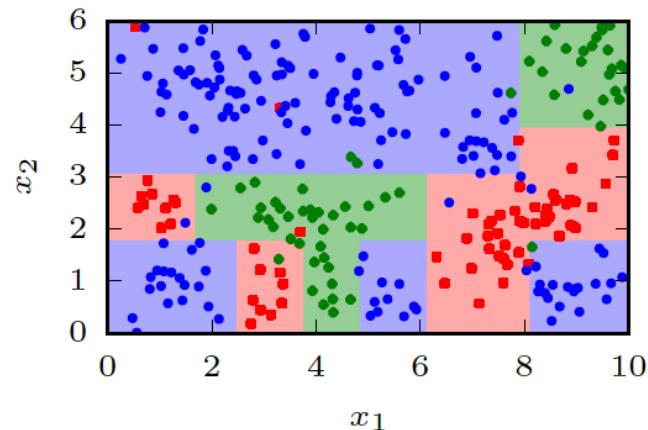
- machine learning algorithm: **random forest classifier**
- general idea:
 - build **decision trees** for classification
 - every tree makes **one prediction**
 - the class with **most predictions** becomes model prediction



Simple Decision Tree Example

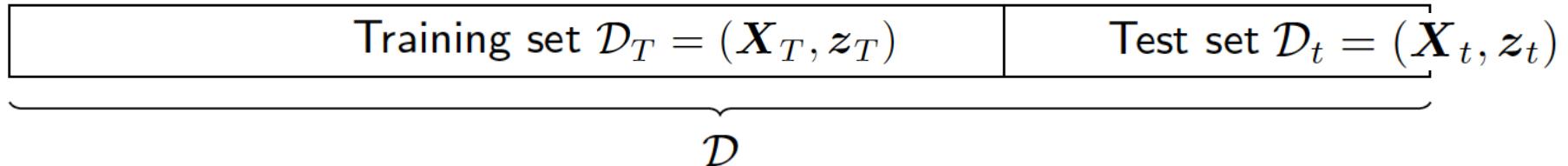


Decision Trees [2], [5]



Distribution of classes in leaf: (red, green, blue)

Decision Trees [2], [5]



- Idea: Build all possible trees and evaluate their performance on \mathcal{D}_t
- All concrete features and values in \mathcal{D}_T can serve as tests in the tree.
- Sadly: „all possible trees“ → combinatorical explosion → finding optimal tree is NP-complete
- → Grow the tree top-down and choose the best split node-by-node using a greedy heuristic on the training data

feature	tests
x_1	≤ 0.36457631
	≤ 0.50120369
	≤ 0.54139549
	$\geq \dots$
	≤ 0.09652214
	≤ 0.20923062
x_2	$\geq \dots$

Decision Trees [2], [5]

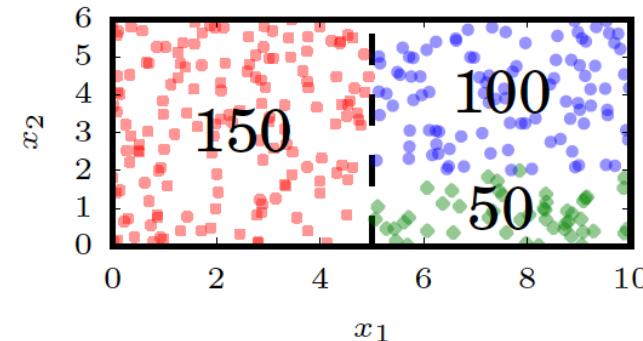
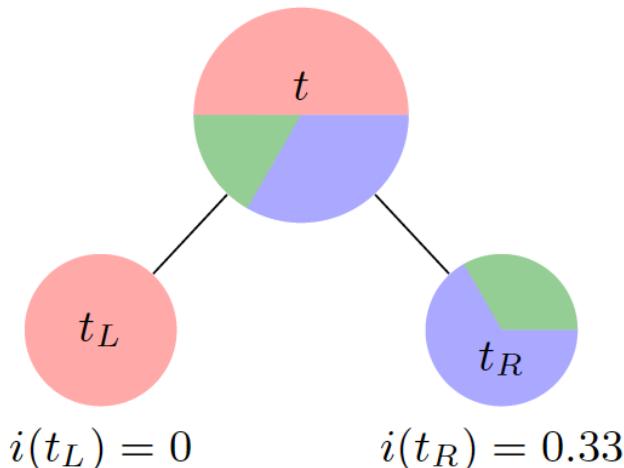
For example: Split the node when it improves the missclassification rate i_E at node t

$$i_E(t) = 1 - \max_c p(z = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is given by

$$\Delta i(s, t) = i(t) - p_L i(t_L) - p_R i(t_R)$$

$$i(t) = 1 - 0.5 = 0.5$$



Impurity measures

With $\pi_c = p(z = c \mid t)$:

Misclassification rate:

$$i_E(t) = 1 - \max_c \pi_c$$

Entropy:

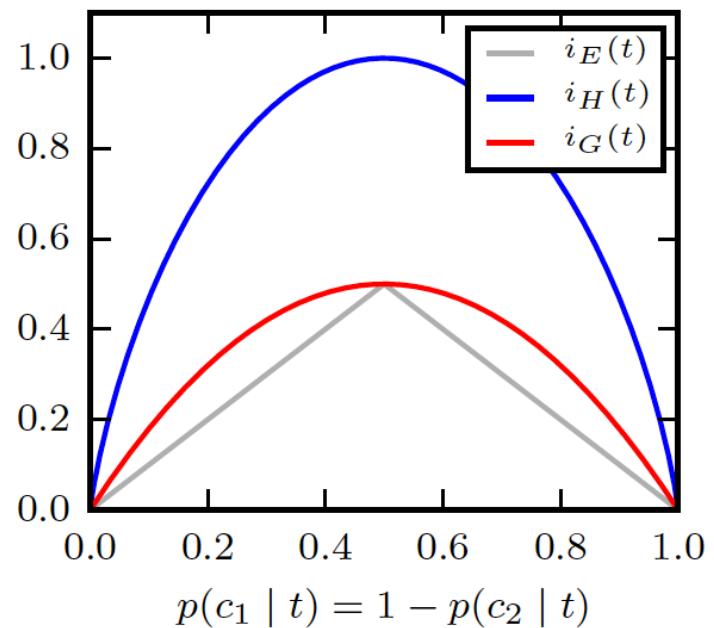
$$i_H(t) = - \sum_{c_i \in C} \pi_{c_i} \log \pi_{c_i}$$

(Note that $\lim_{x \rightarrow 0+} x \log x = 0$.)

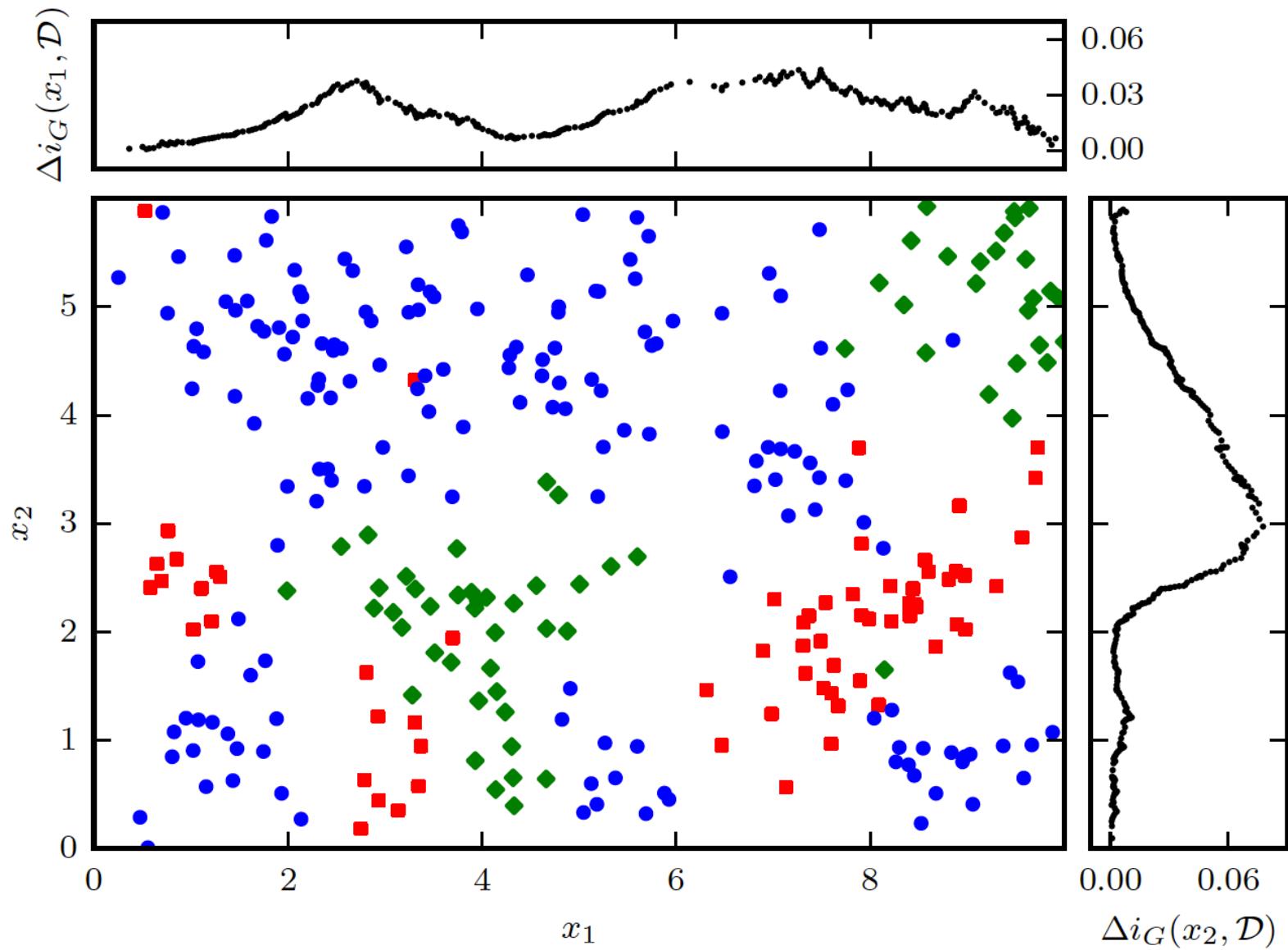
Gini index:

$$\begin{aligned} i_G(t) &= \sum_{c_i \in C} \pi_{c_i} (1 - \pi_{c_i}) \\ &= 1 - \sum_{c_i \in C} \pi_{c_i}^2 \end{aligned}$$

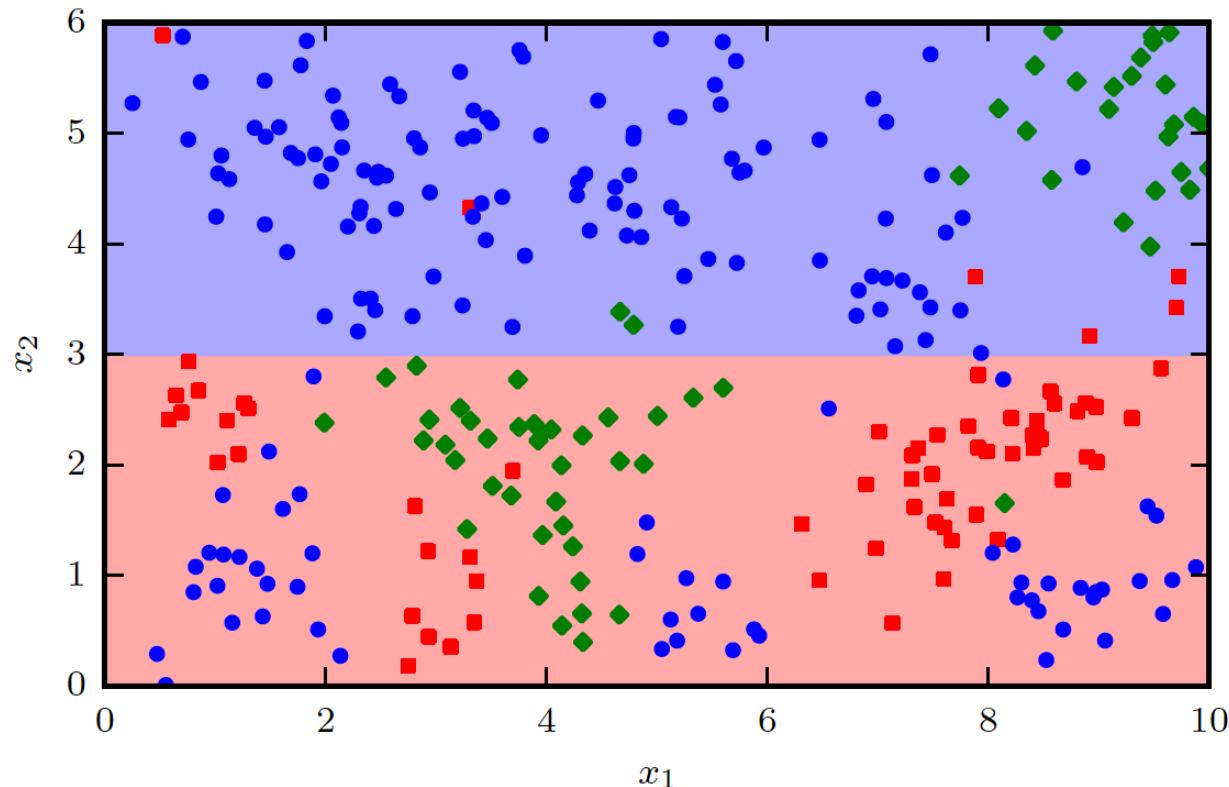
For $C = \{c_1, c_2\}$:



Decision Trees [2], [5]

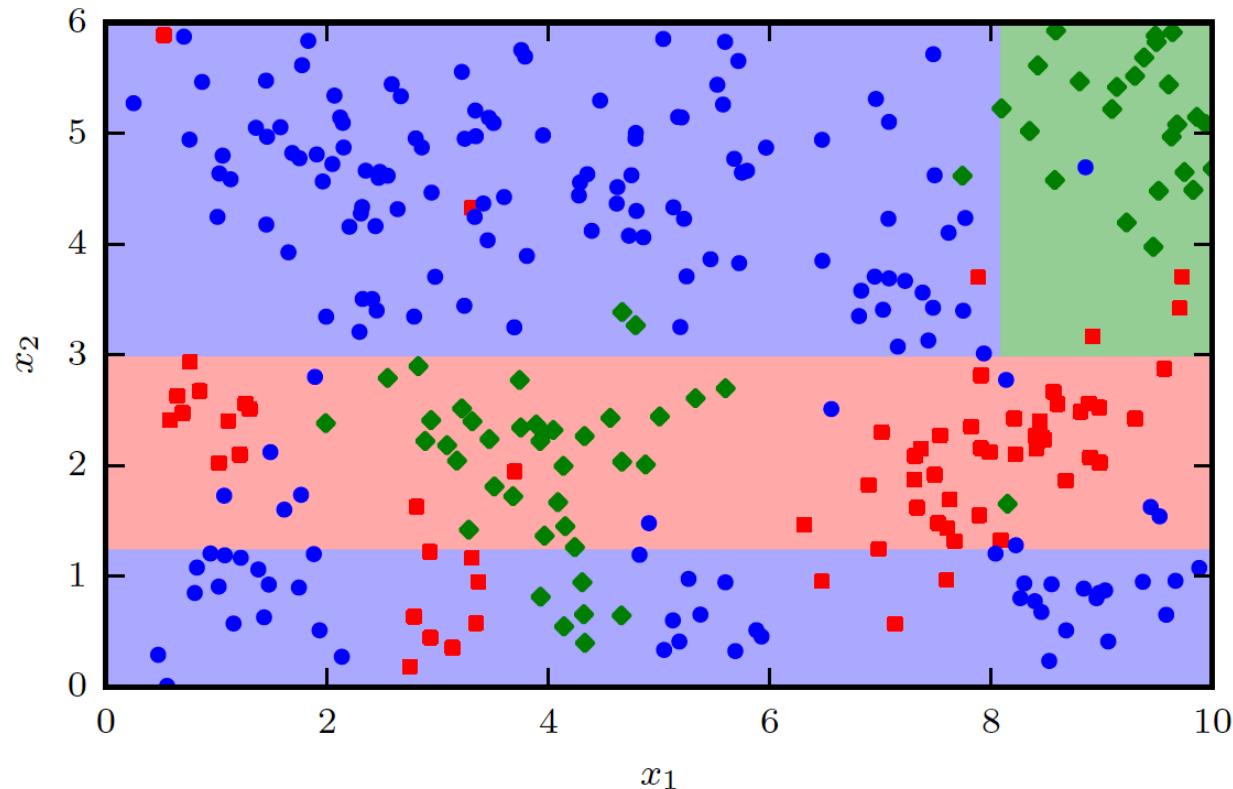


Decision boundaries at depth 1



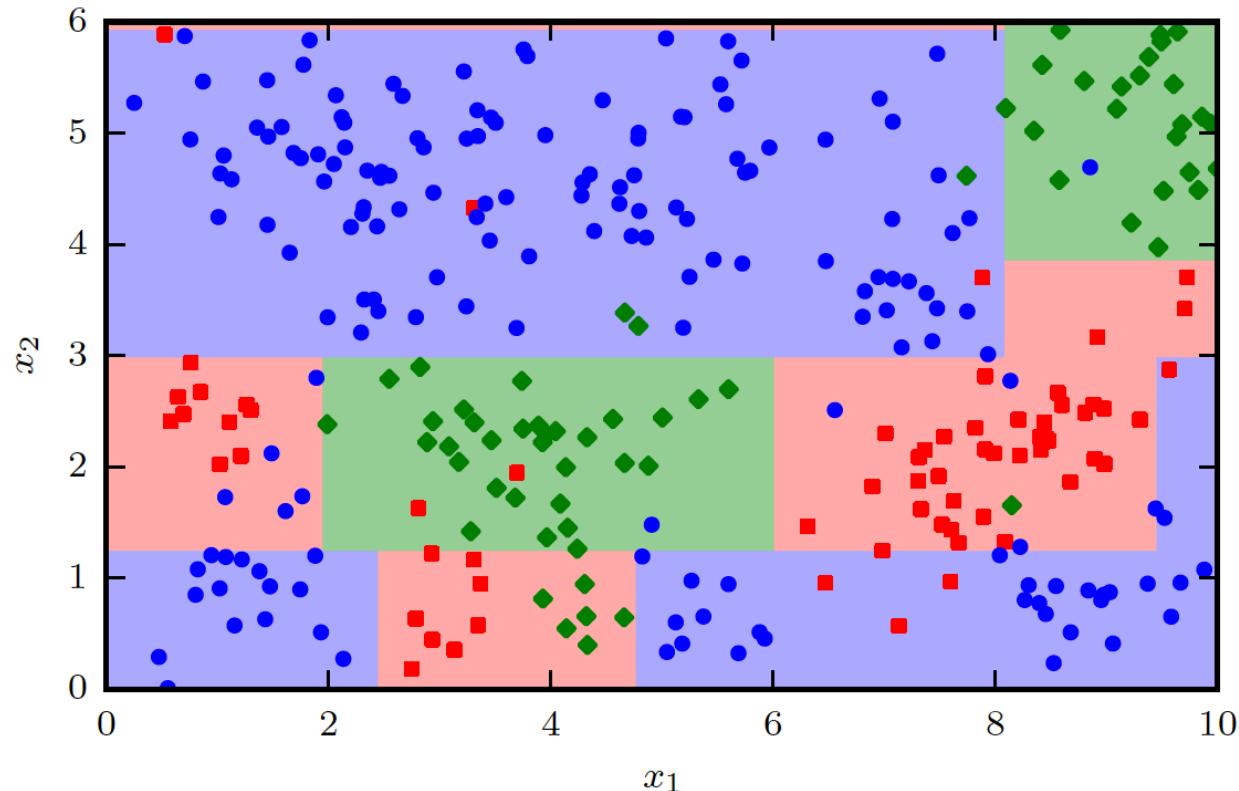
Accuracy on the whole data set: 58.3%

Decision boundaries at depth 2



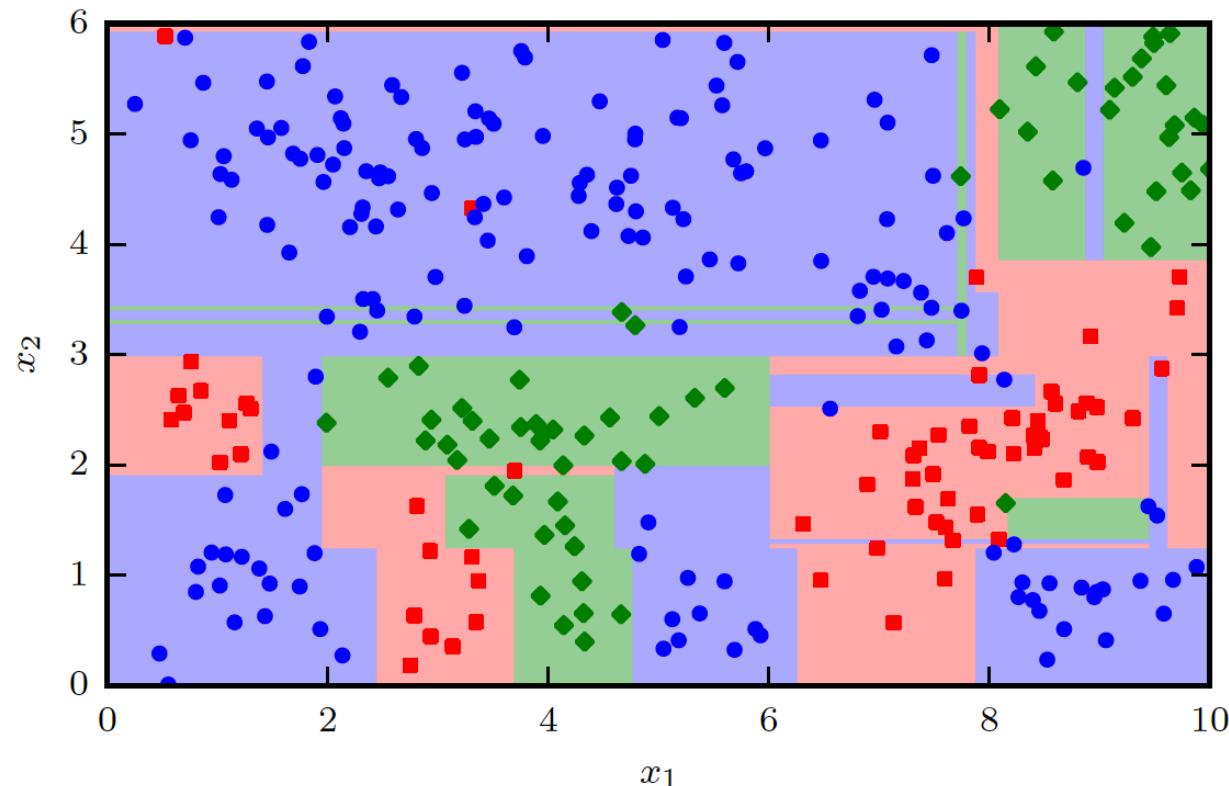
Accuracy on the whole data set: 77%

Decision boundaries at depth 4



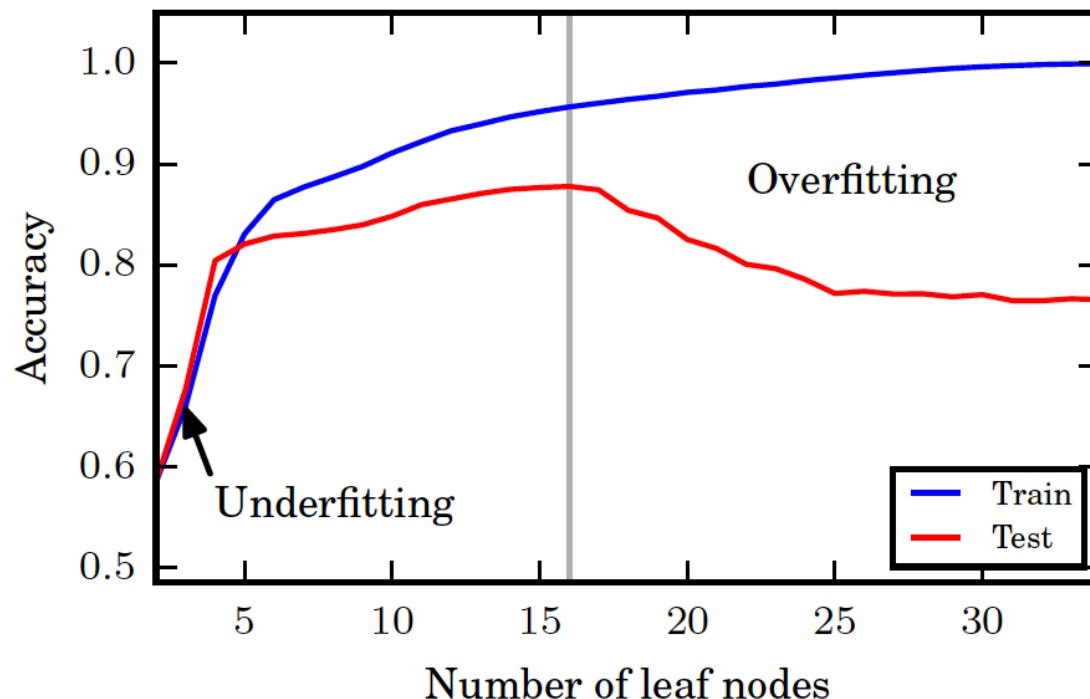
Accuracy on the whole data set: 90.3%

Decision boundaries of a maximally pure tree



Accuracy on the whole data set: 100% \rightarrow generalisation?

Decision Trees [2], [5]



Ideas to counter overfitting:

- limit depth when growing tree
- stop growing when splitting benefit drops below certain threshold
- grow maximally and prune back afterwards

Random Forests: Introducing Variance

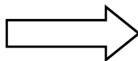
Let N denote the number of samples in the training set \mathcal{D}_T and D denote the number of features (attributes, dimensions).

- Instead of using the whole training set to build the tree, we build each tree T_i from a bootstrap sample:
From \mathcal{D}_T create \mathcal{D}_{T_i} by *randomly* drawing N samples *with replacement*.
By sampling with replacement we only use $\approx 63.2\%$ of \mathcal{D}_T for each tree.
- Build decision trees T_i top-down with a greedy heuristic and a small variation:
At each node, instead of picking the best of all possible tests, *randomly* select a subset of $d < D$ features and consider only tests on these features.
- There is no need for pruning.

Simple TF-IDF Features for Text

- What kinds of features can we use if the input data is text?
→ $|V|$ -dimensional tf-idf vectors:

*"I like to dance samba,
bake pizza, watch tv and
plant trees in the garden. I
also like to bake cakes."*



I	2
like	2
to	2
dance	1
samba	1
bake	2
pizza	1
watch	1
tv	1
and	1
plant	1
trees	1
in	1
the	1
garden	1
also	1
cakes	1

*Often: Instead of term-frequency (tf) alone: use term-frequency * inverse document frequency (idf); idf = - log (#of docs where t occurs / #of docs)*

Sentiment Analysis

- first, the model needs to be **trained**:
 - **feed** the labeled data **into the classifier**
 - **split** the data into **train-set** and **test-set**
 - **train** on the train-set
 - **report performance** with test-set
- the trained model is now **ready** to be used on **unlabeled** data

Tasks (cont.)

a) Preparation

Import the labels and split them into two arrays: the chat itself and the labels.

A label is like a review of a single message:

-1 = negative

0 = neutral

1 = positive

b) How to train your model

In this step you will use the chat and labels to train your random forest classifier. In order to do so, create the random forest classifier, fit it and make a prediction on the test set.

After you are done, print the accuracy score and comment on it.

Hints:

- When creating the classifier, use n_estimators=200, random_state=0 as arguments.
- The test should be 20% of the whole set

Tasks (cont.)

c) Prediction time:

Now you can use the model to predict the sentiments for the whole chat. Import the chat and predict the labels. You will need to use `vectorizer.transform().toarray()` on your data, but **DO NOT** use `fit()` anywhere! The classifier is already fitted, fitting it again effectively erases all it has learned.

Hint: `chat.csv` includes ALL chatlines, including those that have been used in the previous steps and are already labeled. Don't label them again.

Note: The chat table is massive. Labelling all of it may take a while.

```
chatData = pd.read_csv("chat.csv")
unlabeled = chatData.iloc[:,1].values
# TODO:
```

Linear Regression: Theory

- is a technique that fits a function from **input** variables X to a **dependent variable** y :

$$y = \alpha + \beta X + \epsilon$$

- **training data:** $\{(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)\}$
- X is the **predictive vector**, containing the (predictive) variables
- α and β are the model's **parameters**, where
 - α is the **intercept/bias**
 - β the **coefficient vector** containing coefficients for each predictive variable
- And ϵ is an assumed Gaussian **error**

Linear Regression: Theory

- is a technique that tries to find a **correlation** between a set of **input** variables X and a **dependent variable** y :
- our goal:
 - **input** (gold advantage of teams, worst kill-death ratio per team, negativity) **as X**
 - **determine** whether $y = \text{radiant_win}$ can be modeled as a function of the other factors

Tasks (cont.)

Task 4.2: Linear regression

a) Preparation:

1. First, we need to `read the csv` files and `group` them by `match_id`.
2. We will `create a dataframe` containing all relevant information and each row will represent one match. An empty dataframe has already been created for you with all the columns you need to fill. Here is an explanation of what you need to put into each column:
3. `Create a list of tuples` called `full_chatdata`, each tuple has the following structure: label, team.
4. `Create a list` called `goldData` containing the gold advantage for every timestamp of a match (usually every minute).
5. `Create a list` called `KDratios` of kill-death ratios for each player in a match, split into two parts, one for each team, called `ratiosRadiant` and `ratiosDire`.
6. `Add an additional column` called `radiant_win` displaying the winning team using boolean.

Hints:

- you can find the labels in the `chatData` dataframe
- use the `slot` column to determine the team. 0 to 4 is for radiant, 5-9 is for dire
- There is a column in the `match.csv` file called `radiant_win` that displays true if team radiant won, false if not

Tasks (cont.)

b) Per-match analysis:

As you may have noticed, the gold variables are gathered every minute, but the chat times are irregular. We could try to group the chat into 60 second timeframes that would correspond with the gold values, but this would be too tedious. Instead, we will simplify this by looking at the game as a whole:

1. Compute the average negativity for each team by iterating over the list of tuples you created in exercise 4.2.3.
2. Then, compute the average gold advantage for each match, and add a column for the gold advantage at the end of a match. The gold advantage at the end of a match is the last value of the list.
3. Create a new column for the difference in negativity between the two teams.
4. The kill/death ratios aren't very useful in the current format. Take the lowest K/D ratio from each team and create new columns for them. The reasoning behind this is that a low K/D ratio is a sign of underperformance of a player and players who do not perform on an acceptable level are usually harassed more often.

Tasks (cont.)

c) A warm-up regression

Before we test our hypothesis of whether or not the state of the game influences player behavior, we will perform a linear regression with only one input variable: The gold advantage.

You have probably wondered why we just assume that the gold values would represent the state of a game, whether a team is losing or winning. So far, this has only been a theory, and we should test it, as it would not make sense to use it as a representation for the state of the game in the actual regression model, if it wasn't representative at all.

1. Once again, split your data into a train set and test set, create a linear regression model, fit the data and print your score. Try it two times! Your dependant variable should always be radiant_win , your X should be the average gold advantage and the gold advantage at the end.
2. Discuss the score you obtained! What do the results mean for the explanatory power of the gold variables?

Tasks (cont.)

d) Testing our hypothesis

1. Finally, we can do our linear regression. This time, use the gold data, the kill/death ratios and the negativity scores.

```
# TODO 1:  
X = # TODO  
Y = # TODO  
  
# with sklearn  
regr = linear_model.LinearRegression()  
regr.fit(X, Y)  
print(regr.score(X,Y))
```

e) Discussion

What is the score? What does that number mean? Discuss possible reasons for this result.

Hint: Take a peek at the labels.xlsx file and look at some of the most common negative words.

Submitting your solution

- work by **expanding** the .ipynb iPython notebook for the exercise that you **downloaded** from Moodle
- **save** your expanded .ipynb iPython notebook in **your working directory**
- **submit** your .ipynb iPython notebook **via Moodle** (nothing else)
- remember: working in groups is not permitted. Each student must submit **their own** .ipynb notebook!
- we check for **plagiarism**. Each detected case will be graded with 5.0 for the whole exercise
- **deadline**: check Moodle

Citations

- [1] E. Gilbert and K. Karahalios: *Predicting Tie Strength With Social Media*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2009.
- [2] C. Bishop: *Pattern Recognition and Machine Learning*. 2006, chapter 3
- [3] Lecture Machine Learning I, TUM Informatics (Winter 2017)
- [4] <https://degninou.net/2016/02/04/multiple-regression-and-diagnostics-with-python/>
- [5] Lecture Machine Learning I, TUM Informatics (Winter 2016)