Projects INFOB2KI 2018-2019

In partial fulfilment of the requirements to pass the course, two projects have to be executed. The deliverables for each project include a written lab report and a piece of code. The projects are based on a framework provided by UC Berkeley's Pac-Man projects, which we have permission to re-use.

Background The projects are concerned with creating and studying artificial intelligent behaviour in a.o. the well-known game of Pac-Man. You will gain hands-on and in-depth experience with several of the techniques treated in the course, both by coding them (to some extent), and mostly by comparing them and experimenting with their parameters. Coding is only a small part of the effort and provides you with the means to experiment with the techniques. The latter is the main purpose of the projects since it provides you with more insight and understanding of the techniques, their benefits and their limitations. In addition, the projects (especially the second one) will encourage you to research some topics beyond the scope of the basic material treated in the course book and lectures.

Form & guidance The projects are executed in groups of 3 students for the following reasons: [1]

1. the projects involve coding, experimenting, (theoretical) analyses, and writing: the various tasks and the total expected workload fit 2-3 persons
2. the various questions to be answered in the lab-reports assume that you experiment with different parameter settings, figure out what does and doesn't work and come up with a well thought-out and well-motivated theory for your observations. This latter part is one of the main purposes of the projects and is best achieved in groups of 3 where you can discuss and explain your thoughts and observations.
3. we have a responsibility to seriously grade your projects within reasonable time.

Each of you is responsible for finding partners to work with (e.g. in class, or through the discussion forum on Blackboard), and for finding suitable moments and means for communicating and working on the projects together. Since you are expected to spend 20 hours per week on this course, it should be no problem to find time in your schedules to meet (physically or virtually).

During the weekly scheduled sessions (indicated as 'practicum' in the course schedule) teaching assistants are available to answer your questions about the projects. You can also encouraged to discuss problems or issues amongst one another through the discussion forum on Blackboard. Note that to complete the projects, attendance of the scheduled sessions alone will not suffice. Regular attendance will help keep you on track, though, and helps you to meet the deadlines and the requirements. If you feel that assistance is not necessary, attendance is not required.

Software and submission All software required for the projects is listed on the projects' page of the course website. This page includes details on how and when to submit your work. The page also provides access to local copies of the Pac-Man files, if for some reason the Berkeley server is temporarily unavailable. Since failure to meet the project requirements means you fail the course beyond repair, please note the following:

[1] *Deviation of this rule is only allowed under exceptional circumstances*. This could be upon advise of the study-advisor C. de Gee, or the advisor of your own programme; the following arguments are **no** grounds for exception: "I cannot attend the practical sessions", "It is hard to plan time together due to my busy schedule", "I prefer to work alone", "I already did the projects last year",...

1. everyone in the group is responsible for meeting submission deadlines and criteria!
2. since you are working in groups, there is always someone who will be able to submit the projects in time. As a result, **we will not grant any extensions**!
3. only complete submissions, with *all* deliverables in a single file, will be processed.
4. you are responsible for any submission with your name on it, therefore make sure you have seen and approved the submitted version.
5. make sure that the code you submit is the same code that your lab report is based upon. Since you cannot finish the lab report without at least some working code, make sure you submit code that doesn't run into errors under normal operation. Either use version control, or be sure to have back ups of working versions.

Expectations and grading Project A is not graded: you can *autograde* your work, but will not receive any credits for it. For projects B and C the autograder gives an indication of your mark for the code (weight = 30%); the mark for the lab report (weight = 70%) depends on your writing and the answers to the additional exercises (see requirements per project). Please note that the autograder cannot do a complete verification of correctness, so your mark may deviate from the autograder score; moreover, your mark will take the extent to which your code is your own, and is clear and well-documented into account.

- *About the coding part:*
  Since the Pac-Man projects are used in AI courses all over the world, several solutions to the coding parts can be easily found (even though the license doesn't allow publishing of solutions!). Coding is not where the emphasis of these projects lies, although figuring out what to code helps in understanding the subject matter. If you feel you have to partly rely on someone else's code to be able to complete the project, you may do so as long as you
  - explicitly and unmistakably indicate both in the code and the 'Own Code Declaration' (OCD) which parts are not your own and what their true source is (otherwise you are committing both fraude and plagiarism![2];
  - keep the copied code as literal and close to the original as possible;
  - provide *your own* extensive comments for the copied code, demonstrating that you in fact understand what it does.

  Of course you will not receive any credits for code that is not your own. Be sure to verify the correctness of the code: otherwise it will affect the results for the experiments and grade of your lab report as well.

- *About the experimental part:*
  In order to answer the parallel exercises, often experimenting with parameters in your code is required to study the effects of those parameters. Such an empirical study requires that you

[2] **Submissions will be evaluated by a plagiarism detection system**, which compares your code to code submitted by your fellow students and code used by students (here and around the world) that have previously done these projects. If you copy code or solutions from someone else, without due acknowledgement of the source – whether a classmate, friend, vague acquaintance, or total stranger – this is fraud or plagiarism and you will have to suffer the consequences. This also holds for anyone who knowingly and wilfully *enables* fraud or plagiarism (i.e. by posting solutions, leaving them at the printer or on screen, etc). See *art. 5.14 - fraude en plagiaat* of the Onderwijs- en Examenregeling Bacheloropleiding.)

clearly describe the set-up and process of your experiments (in this case typically: what values (or combinations thereof) were chosen for which parameters?), that you gather sufficient data to be able to draw conclusions, and that you describe the obtained results, the conclusions you draw from the results and a justification of the conclusions. This is what you learn in the *Onderzoeksmethoden* course.

For real empirical research enough data must be gathered to be able to draw *statistically supported* conclusions. In this case that means you would need to study many different values of a parameter, and **even more** if you are considering combinations of values of multiple interacting parameters. Since most of you have not yet followed the *Onderzoeksmethoden* course, we do *not* require you to provide statistical proof for your claims. However, you are required to clearly describe exactly what experiments you performed (based upon your description, we should be able to repeat them), and experiment with at least 5 different values **per** parameter (which means a multiple of that for different combinations of 2 parameters, etc) if your conclusions are to be based solely upon experimental results. Note that sometimes it is possible to theoretically support claims about the effect of certain (combinations of) parameters. In that case, a thorough theoretical analysis given in addition to, or instead of, an analysis of the results of a series of experiments will suffice.

Project A: Python and autograder tutorial

Since we are assuming you are not yet familiar with Python, you can use the first course week to (individually) do a Python tutorial described in Project 0 of the Pac-Man projects. Note that this tutorial in addition makes you acquainted with the *autograder* that you can use to get feedback on the coding part of the projects. Project A is not submitted.

Project B: Searching and learning

In the classical Pac-Man game, the player operates Pac-Man, moving him through a maze while collecting food and dodging ghosts (or catching ghosts after eating a power-pill). In the current projects, Pac-Man will be operated by an AI agent. In this project you will implement and experiment with various search algorithms, discussed in class, to let Pac-Man find his way through the maze. In addition, rather than finding his way through a predefined strategy as provided by a standard search algorithm, Pac-Man can learn himself how to best move through the maze, using reinforcement learning. In the course we have discussed the latter at a very abstract level, assuming it can be interpreted as an MDP. In this project you will implement and experiment with value iteration for MDPs and with Q-learning.

Requirements

Project B is based on parts of Pac-Man projects 1 (Search) and 3 (Reinforcement learning).

- You will complete the code for questions 1 (Q1) through 4 (Q4) of project 1 and Q1 through Q5 of project 3.
- You will write a (well-written!) lab report addressing answers to the 13 exercises below.
- You will submit:
    1. all .py files you altered, since those are the ones that will be run and graded. *You are not supposed to edit any files other than the ones you are instructed to edit by the Berke-*

ley description! Your code should be well-structured and include ample comments and documentation.

2. the lab report (in pdf). *Do not forget to include your names and student-numbers in the report!*
3. a filled-out 'Own Code Declaration' (download this from the projects' page of the course website).

Some additional information: Check out Berkeley's lecture slides (also on video) on search, and on MDPs and reinforcement learning:

- Lecture 2: Uninformed Search
- Lecture 3: Informed Search
- Lectures 8 and 9: MDPs I and II
- Lectures 10 and 11: Reinforcement Learning I and II

Exercise 1

The purpose of this exercise is to get acquainted with the Pac-Man framework; answer the questions in enough detail to convince us that your understanding of the framework suffices for successful completion of the project. Note that the mentioned files are present in both the Search and Reinforcement learning archives.

a. In the file `pacman.py` the game state is defined. Describe in your own words what is exactly represented in the game state, and how the game state is updated upon action.

b. In the file `game.py` the state of an agent is defined. Describe in your own words what is exactly represented in the agent state, what actions are available and how the agent state is updated upon action.

c. The file `util.py` defines a number of data structures. Indicate which data structure is best described by which metaphor:

Data structure:	Metaphor:
A. Stack	I. emergency room waiting area
B. Queue	II. huge pile of dirty restaurant dishes
C. Priority Queue[a]	III. roller coaster waiting line

[a] NB carefully inspect the code: is smallest or largest number the one with highest priority?

Points: a: 2; b: 3; c: 2

Exercise 2

a. Describe a generic code structure that you can use for each of the graph-search algorithms requested for questions Q1 ... Q4 of the Search project. Motivate your choice by describing how each of the algorithms can exploit the structure.

b. How can you change your structure into one suitable for tree-search versions?

Points: a: 4; b: 2

Exercise 3

Complete Search Q1 and answer the following questions:

a. What does it mean for a search strategy to be complete?

b. Argue whether or not your implemented search strategy is complete.

c. Argue whether or not your solutions are guaranteed to be least cost solutions.

d. Consider all explored states and the order of exploration. Is the exploration order what you would have expected? And does Pac-Man visit all the explored squares on his way to the goal? Explain why or why not.

Points: a: 2; b, c: 3; d: 2

Exercise 4

Complete Search Q2 and answer the following questions:

a. Argue whether or not your implemented search strategy is complete.

b. Argue whether or not your solutions are guaranteed to be least cost solutions.

c. Does your Pac-Man code return a solution for the eightpuzzel? Explain why or why not.

Points: a, b: 3; c: 2

Exercise 5

Complete Search Q3 and answer the following questions:

a. Explain for each of the three UCS agents *what* their intended behaviour is, and *how* this is achieved by the cost functions.

b. Run the three UCS agents on a (single!) maze of your choice. Report the name of the maze, the path costs you find for the three agents, and explain the differences in path costs by comparing the cost functions.

Points: a: 4; b: 3

Exercise 6

Complete Search Q4 and compare DFS, BFS, UCS, and A* with Manhatten-distance on `OpenMaze`. Clearly describe what each strategy does, and explain the differences or similarities in behaviour (solutions, path costs, number of expanded nodes, ...).

Points: 6

Exercise 7

The following questions are meant to get further acquainted with the Pac-Man framework, and more specifically with the MDP and q-learning code; answer the questions in enough detail to convince us that your understanding of the framework suffices for successful completion of the rest of the project. The mentioned files are available in the Reinforcement learning archive.

a. The files `valueIterationAgents.py` and `qlearningAgents.py` are both set up to implement a type of learning agent. Describe in your own words what the difference between these two types of agent is.

b. In the file `mdp.py` a number of methods is defined. For two of them a remark is made concerning their applicability in reinforcement learning. Describe in your own words why applicability of these methods is restricted or impossible in reinforcement learning.

c. The file `gridworld.py` defines the Gridworld MDP. MDP implementations can differ in how they handle terminal states and when exactly rewards are given. Discuss the implementation of the Gridworld MDP with respect to these issues. Are the Gridworld examples in the course-slides based on a similar implementation? Motivate your answers.

d. Give the definition of a terminal state as can be found in the literature (provide your source), and compare that definition with the implementations of such states in `gridworld.py` and the course slides.

Points: a, b: 2; c, d: 3

Exercise 8

a. The inclusion of one or more terminal states in an MDP is often done to model a certain type of task. How do we call this type of task or the MDP that implements it?

b. The value of a state in an MDP is typically defined as the expected sum of future (possibly discounted) rewards. Should the presence of terminal states in an MDP have an effect on the discounting factor used for computing the value of non-terminal states? Explain your answer.

Points: a: 1; b: 2

The Gridworld MDP has a number of parameters that may influence the computed policy of an agent. In the following two questions we will focus on three of these parameters:

- the discount rate γ for future rewards: option '-d'
- the noise n in the planned movement: option '-n'
- the immediate reward r at each time step: option '-r'

For each experiment you perform in the exercises 9 – 12, you should clearly describe in your lab report which parameter(s) you vary, what values you considered and what the corresponding outcomes were!

Exercise 9

Complete Reinforcement learning Q1 and Q2. For the latter you have to create a value iteration agent that crosses the bridge and answer the following questions:

a. Analyse and report for (at least) 5 different discount parameters whether or not the agent crosses the bridge, or shows evidence of at least attempting to cross the bridge. Repeat your analysis for (at least) 5 different noise parameters. Be sure to specify the parameter values used.

b. Which parameter did you change in `analysis.py` and to what value? Clearly motivate why you chose to change this parameter, and why to this specific value. Does it have the desired effect in that the agent crosses the bridge? If so, can you argue why? If not, can you think of an explanation?

Points: a: 6; b: 3

Exercise 10

Complete Reinforcement learning Q3 and answer the following questions:

a. For each of the 5 policy types (3a,. . . , 3e), give the 3-tuple (γ, n, r) that you set in `analysis.py`.

b. For each of the 5 policy types, provide theoretically backed arguments why this parameter setting has the desired effect, or why your answer is 'NOT POSSIBLE'.

Points: a: 0 (-1 if unanswered); b: **10**

A Q-learner uses a number of learning parameters. In the following questions we consider:

- the learning rate α (option '-l' in gridworld)
- the exploration rate ϵ (option '-e' in gridworld)

For the crawler robot the parameters can be changed in the GUI. You will experiment with different combinations of parameter settings.

Exercise 11

Complete Reinforcement learning Q4 and Q5. You are now able to run the Q-learning crawler robot and experiment with its parameters.

a. Experiment with 5 different values of learning rate α; use the results to hypothesize about how changes in only the learning rate affect the agent's policies and actions. Provide further evidence for your hypothesis using (theoretical) arguments or additional experimental results.

b. Experiment with 5 different values of exploration rate ϵ; use the results to hypothesize about how changes in only the exploration rate affect the agent's policies and actions. Provide further evidence for your hypothesis using (theoretical) arguments or additional experimental results.

c. Experiment with different combinations of α and ϵ; use the results to hypothesize about how changes in both these parameters affect the agent's policies and actions. Provide further evidence for your hypothesis using (theoretical) arguments or additional experimental results.

d. Does the effect of changing α depend on the value of the discount rate γ? If so, how and why? If not, why not? Give clear (theoretical) arguments and/or describe which experiments you have done to arrive at your conclusion.

e. Does the effect of changing ϵ depend on the value of the discount rate γ? If so, how and why? If not, why not? Give clear (theoretical) arguments and/or describe which experiments you have done to arrive at your conclusion.

Points: a, b: 4; c: 6; d, e: 4

Exercise 12

The file `qlearningAgents.py` specifies the PacmanQAgent as a QLearningAgent with default learning parameters tuned to the Pac-Man problem. The Q-learning Pac-Man is run on a tiny grid using the following command:

```
python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
```

The default Q-learning parameter settings of Pac-Man agents can be changed using the option '-a' (agentArgs); see `python pacman.py -h`.

a. Experiment with the learning parameters (ϵ, γ and α) of the PacmanQAgent on `smallGrid`. Clearly describe your experiment and use your observations to motivate why the default settings for Pac-Man ($\epsilon = 0.05$, $\gamma = 0.8$, and $\alpha = 0.2$) seem the most effective.

b. Train PacmanQAgent on mediumGrid for at least 2000 games, using Pac-Man's default learning parameter settings, and test it. Report:

- the number of training games you used
- the average reward over all training games
- the number of test games you used
- the win rate and average score

What was your expectation of Pac-Man's performance? Are the results better or worse than you expected? At hindsight, can you explain that?

Points: a: **8**; b: 4

Exercise 13

The score for your code is based on the correctness of the code, the extent to which you used your own code and the readability of the code. In addition to inspecting your code, we will also run an autograder on it, which is used as a first assessment of its correctness. Sometimes we get different scores than you; to enable an explanation of this, we ask you to list the *autograder* scores for all the projects' questions:

	Q1	Q2	Q3	Q4	Q5	Total
Search	x/3	x/3	x/3	x/3	0	x / 12
Reinf.learn.	x/6	x/1	x/5	x/5	x/3	x / 20
Total:						x / 32

Project C: (Non)-Probabilistic classification

In the course we have discussed different types of classification model, both probabilistic (such as naive Bayes) and non-probabilistic (such as neural nets). In this project you will design, train and experiment with different classifiers, among which classifiers that were *not* treated in class: multi-class perceptrons, a large-margin classifier (MIRA) and a modified perceptron for behavioural cloning. This project therefore asks for a more independent attitude towards the course subjects.

Requirements

Project C is based on Pac-Man project 5 (Classification).

- You will complete the code for questions 1 (Q1) through 6 (Q6) of the project.
- You will write a (well-written!) lab report addressing answers to the 9 exercises below.
- You will submit:
  1. all .py files you altered, since those are the ones that will be run and graded. *You are not supposed to edit any files other than the ones you are instructed to edit by the description!* Your code should be well-structured and include ample comments and documentation.
  2. the lab report (in pdf). *Do not forget to include your names and student-numbers in the report!*
  3. a filled-out 'Own Code Declaration' (download this from the projects' page of the course website).

Some additional information, hints and tips: In this project you will work with data for three types of application (all data is found in `data.zip`). In addition to the Pac-Man application that you already used in the first project and the OCR task briefly described in the introduction to Berkeley project 5 (Classification), you will consider the task of face detection as required for applications such as human computer interaction and surveillance. You will consider a simplified face detection task from images that have been pre-processed by an edge detection algorithm. The task is to determine whether the edge image is a face or not. In order for the provided code to work with the face data, a few additions to the file `dataClassifier.py` will be required.

For the perceptron and MIRA classifiers, familiarise yourself with

- multi-class perceptrons
- the MIRA classifier; NB oncerning the formulas presented for MIRA in Q3: the double bar $||$ is used to indicate the length (norm) of a vector, the subscript 2 indicates that it is a 2-norm (which means regular Euclidean space), and the superscript 2 is an exponent. So if v is the vector $(1, 2, 3)$ then $||v||_2^2$ is simply the length of the vector squared:

$$||v||_2^2 = \left(\sqrt{1^2 + 2^2 + 3^2}\right)^2 = 1^2 + 2^2 + 3^2 = 14$$

- the Berkeley's lecture slides (also on video) on perceptrons:
  - Lecture 22: Perceptron

Exercise 1

Any classifier basically takes as input the values to a number of pre-defined features and returns a label indicating the 'class' it assigns the input to. In this exercise we focus on classifiers with just *basic*[3] features for recognising handwritten digits and for recognising faces. Images have been pre-processed for our purposes and can be found in `data.zip`. Study these files together with the files `samples.py`, `dataClassifier.py`, `mostFrequent.py` and `naiveBayes.py`.

a. Clearly describe, both for digit and face classifiers and data:

   - the number of input features and what they represent
   - the possible values of the input features and what they represent
   - the output labels and what they represent
   - the frequency or probability distributions over the output labels (distinguish between training, validation and testing data)

b. Consider the classification methods for both the 'most frequent' and the 'naive Bayes' classifiers. Clearly describe the difference in decision rules used to determine the class label for a given input.

c. Consider the training methods for both the 'most frequent' and the 'naive Bayes' classifiers. Clearly discuss, for both classifiers, whether or not these methods use *supervised learning*.

Points: a: 8; b, c: 2

Exercise 2

The purpose of this exercise is to get further acquainted with the 'most frequent' and 'naive Bayes' classifiers, and the classification pipeline. Again study the files `dataClassifier.py`, `mostFrequent.py`, and `naiveBayes.py`. Make the following adaptations to `dataClassifier.py`:

- in the function `readCommand` add the range of legal labels for the face data
- in the function `runClassifier` add code to correctly load the face data

a. Run the `dataClassifier` code on the two above-mentioned classifiers using both the digit data (= default) and the face data. Write down, for each combination, the validation and testing accuracy. In addition, provide the value of the smoothing parameter k used by naive Bayes. (These results will be your baseline for various comparisons.)

		accuracy		
classifier	data	validation	testing	k
most frequent	digits			$\times$
	faces			$\times$
naive Bayes	digits			
	faces			

[3] With basic features we refer to those that are directly extracted from input data without any additional domain knowledge; using actual knowledge of the domain we can sometimes define 'smarter' features, which is the topic of Q4 – Q6.

Give an explanation for the difference in performance of the two classifiers (combine both general properties of the classifier and of the data in your explanation; if appropriate, you can refer to your answers for Exercise 1).

Can you also explain the difference in performance of each classifier for the two applications (digits vs faces)?

b. The smoothing parameter in naive Bayes training is used to perform *Laplace smoothing*. Explain *in your own words* what the smoothing does and why it is used (hint: Google).

c. Train the naive Bayes classifier with the `--autotune` option on both the digits and faces data sets. Explain what this option does. In addition, report the validation and test accuracy you find and the value of k to which they correspond:

		accuracy		
classifier	data	validation	testing	k
naive Bayes	digits			
	faces			

Finally, compare the performance of this naive Bayes classifier with that of part a. (is it better/worse/comparable? does this differ per data set?) and explain the differences or similarities.

d. Note that upon training the above-mentioned classifiers, three sets of data are used: a training set, a validation set and a test set. Explain, in your own words, for both classifiers ('most Frequent' and 'naive Bayes') what the *exact* purpose of these three sets is.

e. If you are asked to report *the* accuracy of a classifier for some application, should you use the accuracy on a validation set or on a test set? Clearly explain your answer, including an explanation of the general difference between the two.

Points: a: 6; b: 2; c, d: 4; e: 2

Exercise 3

The purpose of this exercise is to study the effect of implementation choices and bias on the performance of the 'naive Bayes' classifier. To this end, you will have to edit the file `naiveBayes.py`.

a. In the file `naiveBayes.py` you can see that the naive Bayes classifier computes with the $\log$ of probabilities rather than using the probabilities themselves. Use the *number of features* in the digit or face data to explain why using the $\log$ is a wise thing to do.

b. In the file `naiveBayes.py` you can also see that the naive Bayes classifier computes *joint* probabilities $P(\text{label} \wedge \text{datum})$ rather than *posterior* probabilities $P(\text{label} \mid \text{datum})$ as we did in class. Argue why using the $\log$ and joint rather than posterior probabilities does not change the classification.

Consider the faces data. In the current implementation, though perhaps not immediately clear, the naive Bayes classifier basically classifies an input case as a 'face' iff $P(\text{face} \mid \text{input features}) \geq 0.5$. We call 0.5 the *decision threshold* for concluding 'face'. Suppose that for our purposes the classifier returns too many false positives ('face' is concluded for a non-face). Adapt the code in `naiveBayes.py` such that for the faces data you can work with an adaptable decision threshold.

c. Train the naive Bayes classifier with $k = 0.1$ on the faces data. Experiment with different values of the threshold and report the associated accuracy on the faces test data :

	decision threshold					
	0.55	0.60	0.65	0.70	0.75	0.80
accuracy						

What conclusions would you draw from the above results about the classifier's ability to separate faces from non-faces?

d. Now consider the digits data. Suppose the training data were strongly biased in the sense that even numbers occurred twice as often as odd numbers. How would this effect the training of the naive Bayes classifier?[4] If you are aware of the specific bias in the data, how could you correct the classifier for this, without training it again on new data?

Points: a: 2; b: 3; c: 6; d: 2

Exercise 4

Complete Q1 and answer the following questions:

a. Report validation and testing accuracy of the perceptron on the digit data after i training iterations, for $i = 1, 2, 3, 4, 5, 6, 10, 50$. Also mention in which order your code goes through the training examples. Is your perceptron better than the naive Bayes classifier? Provide a clear argument to support your claim.

b. Based upon the above numbers, when would it be safe to stop training (if safe at all)? Clearly motivate your answer.

c. Give a drawing or clear description of the multi-class perceptron learnt for the digit data. Your description or image should show, if applicable: input nodes, output nodes, hidden nodes, their connections, weights, activation functions, thresholds/biases, etc. (you just don't have to provide actual numbers for the weights: those are only available after training).

Points: a: 4; b: 3; c: 5

Exercise 5

Complete Q2 in which you analyse your learnt perceptron on digits data, then answer the following questions:

a. Clearly motivate your choice for weight sequence 'a' or 'b' in `answers.py`.

b. Given the type of functions a perceptron is capable of learning, is it in theory a suitable instrument for classifying digits? Argue why or why not.

Points: 2 each

[4] You should be able to answer this question theoretically, but as inspiration you can easily construct such a data set to see what the effect is.

Exercise 6

Complete Q3 in which you train, tune and analyse a MIRA classifier on the digits data, then answer the following questions:

a. In class we included a learning rate α in the perceptron learing rule; in the perceptron learning rule implemented for Q1 you in fact used $\alpha = 1$. The MIRA classifier uses a *dynamic* 'margin-dependent' learning rate, called τ, which limits the amount of change applied to the weights. Explain why limiting the weight-change is a wise thing to do, especially for domains with a large number of features.

b. Clearly explain what the option `--autotune` does in relation to the constant C used by MIRA. In addition, report the validation and test accuracy you find and the value of C to which these correspond. Finally, compare the performance of your MIRA classifier with the perceptron and the (best) naive Bayes classifier: is it better/worse/comparable? Provide clear arguments to support your claims.

c. Similar to the perceptron in Q2 you can determine the 100 highest weight features for each label and visualize them. Present images of the results for the MIRA classifier, and compare them to those of the perceptron in Q2. What kind of differences and similarities do you see? Can you explain these? (Note that you have to copy a piece of code from the perceptron; this adjustment of mira.py is allowed.)

Points: a: 2; b: 6; c: 3

Exercise 7

Complete Q4 in which you reconsider a naive Bayes classifier on the digits data, then answer the following questions:

a. Is a naive Bayes classifier with its underlying independence assumptions in theory a suitable instrument for classifying digits? Provide clear arguments to support your claim.

b. Execute

```
python dataClassifier.py -d digits -c naiveBayes -o -1 4
-2 2.
```

Clearly explain what the command does, what *odds* are and how to interpret the output.

c. Design and describe (at least) four features that can help distinguish between (and therefore classify) digits (note that you do not necessarily have to implement them all!). Clearly explain why you believed at design time that your features could improve classification.

d. Which features did you in fact implement? And what is the test performance of your classifier upon executing

```
python dataClassifier.py -d digits -c naiveBayes -f -a -t
1000?
```

Has the performance of your naiveBayes classifier improved as a result of the implemented features? Provide clear arguments to support your claim.

Points: a, b: 2; c: 12; d: 3

Exercise 8

Complete Q5, in which you train a perceptron for pacman, and Q6 in which you attempt to improve the perceptron by trying to clone the behaviour of a number of predefined agents. Then answer the following questions:

a. In Q5, what is the validation and test performance of your baseline perceptron classifier given by executing

```
python dataClassifier.py -d pacman -c perceptron?
```

b. For each of the following agents, design and describe at least the number of indicated features (note that you do not necessarily have to implement them all!). In addition, clearly explain why you believed at design time that these features would capture the intented behaviour of the corresponding agents (to at least some extent).

   a. **StopAgent**: 1 feature
   b. **FoodAgent**: 1 feature
   c. **SuicideAgent**: 2 features
   d. **ContestAgent**: 3 features

c. Which features did you in fact implement? Discuss, for each of the agents individually, whether or not the use of the implemented feature(s) increases the performance of your classifier. Provide clear arguments to support your claims.

Points: a: 1; b: 12; c: 8

Exercise 9

The score for your code is based on the correctness of the code, the extent to which you used your own code and the readability of the code. In addition to inspecting your code, we will also run an autograder on it, which is used as a first assessment of its correctness. Sometimes we get different scores than you; to enable an explanation of this, we ask you to list the *autograder* scores for all the project's questions:

Q1	Q2	Q3	Q4	Q5	Q6	Total
x/4	x/1	x/6	x/6	x/4	x/4	x / 25 = . . .