CSI344 End-of-Course Assessment

Multiclass Perceptron Learning and Agents

Assigned: Monday 2020-06-22

Due: Monday 2020-07-06 11PM via Moodle End-of-Course (SUBMIT HERE) Link

[Maximum mark: 100]

This assessment is in two parts, Part A and Part B. Part A requires submission of a single Python file, eca.py, and a data file, and Part B requires submission of a pdf file. All three should be zipped together in a single .zip archive and submitted via the End-of-Course (SUBMIT HERE) link on Moodle.

PART A [60 MARKS]

You are to implement a Machine Learning Evaluation system based on the Multiclass Perceptron Learning algorithm in Python. Your are provided a file with data representing apples. There are 120 apples altogether. Each apple has two features and a class ("A", "B", or "C"). Your system must read in the data file, split such that 70% of apples are used for training and 30% for testing. Use training apples to learn the model for the problem, then use the 30% apples for testing, and output an evaluation (percentage of test set which are correctly classified). Download the template file eca.py. Do not modify any code already present in the template; provide additional code as specified below. Your implementation must not make use of any libraries outside of those discussed in this document.

You are to work individually: group work is not allowed. Any evidence of copying will earn a zero. Failure to follow the specification described below and in the template will earn you a zero. The following are the requirements. Your file name should be eca.py.

1. Implement VectorAdd(vectorA,vectorB) as in Lab 5. It takes in two vectors as parameters and returns a vector produced by adding vectorA to vectorB.
2. Implement VectorSub(vectorA,vectorB) as in Lab 5. It takes in two vectors as parameters and returns a vector produced by subtracting vectorB from vectorA.
3. Implement VectorMult(vectorA,vectorB) as in Lab 5. It takes in two vectors as parameters and returns a value produced by multiplying vectorA with vectorB.
4. Implement `LoadFile(fileName)` which should read the data file and return a tuple of two lists; one list for training data, the other one for testing data. Each apple is represented by a list. The file name of the data file is **data.txt**. The fields are separated with spaces. The first row has headers (field names). The first column is for Color, the second column is for Size, the last column is Class (A, B, or C). The file should first of all be read and before separation into two lists the complete list should be randomly shuffled, to ensure no bias related to the order of apples in the data file. **Hints:** To read from a file: `myfile=open(fileName,"r")`. To skip the first line, use `next(myfile)`. To go through an open file line by line: for line in myfile. To split a line from the file into a list of items, `linesplit=line.split()`. To convert a string into a float: `float(mystring)`. To randomly shuffle a list: `random.shuffle(mylist)`. Before you use random shuffle, seed the random number generator with: `random.seed(2)`. You need to import random first. To extract the first 10 items of mylist: `mylist[:10]`, the last part after the first 10: `mylist[10:]`.
5. Implement MulticlassPerceptronLearning(trainingvectors) described in Appendix A at the end of this document (and in the slides on Moodle), which should return a dictionary of weight vectors. The dictionary should have the following structure:

```
weights={'A': [list of weights for A],
'B':[list of weights for B],'C':[list of weights for C]}.
```

The initial weight vectors should be `[0,0,0]` for all classes (this includes the bias, which takes the role of a threshold). The threshold to be used is zero (0). A message should be output which contains the iteration number and each incorrectly classified apple, and the weights at the end of each iteration. **Further requirements:** a function named `argMax()` has been provided. This function takes in a dictionary of scores for each class, and returns the class with the highes score. The input dictionary has the following structure:

`{'A': score for A, 'B': score for B, 'C': score for C}`.

Therefore, in MulticlassPerceptronLearning(), you must repeat the following until all apples are correctly classified or until a maximum of 1000 iterations has been executed.

Consider each apple, one at a time, and calculate a score for that apple for each possible class, using each of the 3 weight vectors.

You then have to use `argMax()` to get the class with the highest score. If the class is the correct class for the apple, no change to weight vectors is necessary. If it is not, you have to subtract the apple feature vector from the weight vector for the incorrect class, and add the apple feature vector to the weight vector for the correct class. This serves to nudge both weight vectors in the correct direction.

6. Implement `Evaluate(weight,appleVectors)`. This function takes each apple from the test set and calculates a score for each class, which is stored in a dictionary of weights. argMax() is then called, which yields the predicted class. If the predicted class is the right class, it is counted as a correctly classified apple. If not, the count of correctly classified apples is not increased. It returns the percentage of correctly classified apples.
7. Study how `RunExperiment()` is implemented in order to know the parameters and return values for the various functions. This function calls `LoadFile()`, which returns two lists, one for training and one for testing, calls MulticlassPerceptronLearning(), which returns a dictionary of weight vectors, then calls `Evaluate()`, which returns an accuracy score. `RunExperiment()` then prints evaluation and exits.
8. Write a comment at the end of your file discussing the performance of your Machine Learning model.
9. Submit both files together with the Part B pdf file, in a single .zip archive.

PART A ADDITIONAL INFORMATION

Multi-Class Perceptron

The multi-class perceptron algorithm is a supervised learning algorithm for classification of data into one of a series of classes.

Algorithm Summary

This algorithm, like most perceptron algorithms is based on the biological model of a neuron, and it's activation. In the case of a normal perceptron (binary classifier), the data is broken up into a series of attributes, or features, each with a specific value. When this feature vector is received by the artificial neuron as a stimulus, it is multiplied (dot product) by a weight vector, to calculate the activation value of the specific data point. If the activation energy is high enough, the neuron fires (the data meets the classification criteria).

In the case of a multi-class perceptron, things are a little different. The data comes in the same way, but instead of the respecting feature vector being multiplied by a single weight vector (for a single class), it is multiplied (dot product) by a number of weight vectors (a separate vector of weights for each unique class). Whichever weight vector that yields the highest activation energy product is the class the data

belongs to. This decision process is known as the Multi-Class Decision Rule.

Training Process

To train the algorithm, the following process is taken. Unlike some other popular classification algorithms that require a single pass through the supervised data set, the multi-class perceptron algorithm requires multiple training iterations to fully learn the data. The iteration count can be easily set as a parameter.

During each iteration of training, the data (formatted as a feature vector) is read in, and the dot product is taken with each unique weight vector (which are all initially set to 0). The class that yields the highest product is the class to which the data belongs. In the case this class is the correct value (matches with the actual category to which the data belongs), nothing happens, and the next data point is read in. However, in the case that the predicted value is wrong, the weight vectors are corrected as follows: The feature vector is subtracted from the predicted weight vector, and added to the actual (correct) weight vector. This makes sense, as we want to reject the wrong answer, and accept the correct one.

After the final iteration, the final weight vectors should be somewhat stable (it is of importance to note that unlike the assumptions of the binary perceptron, there is no guarantee the multi-class perceptron will reach a steady state), and the classifier will be ready to be put to use.

PART B [40 MARKS]

You are to write a report on design of an Intelligent Agent for a system. This has to be described in sufficient detail in order to address all the points listed below. This must be your own work, not a copy of work done by someone else or downloaded from the Web. You must be creative. Every student is given his/her own system, listed in Table 1 on Page 4. You are to write:

1. Section 1 (Introduction). Give an overview of the problem the intelligent agent is supposed to solve. At least half a page.
2. Section 2. Describe the task environment (performance measure, external environment, actuators, sensors) with respect to your given system, in as much detail as possible. At least one page.
3. Section 3. Explain and classify the task environment along the following dimensions: fully or partially observable, single-agent or multi-agent, deterministic or stochastic, episodic or sequential, static or dynamic, discrete or continuous, and known or unknown. For each dimension, explain your choice. At least one page.
4. Section 4. Determine which agent design is appropriate: simple reflex agent, model-based reflex agent, goal-based agent, utility-based agent (and integration into a learning agent, if necessary) and explain the role all components work, specific to your agent, with reference to your given system. At least one page.
5. Section 5. Give an overview of the operation of the intelligent agent, describing how it will be used and its operations within the given environment. You must demonstrate as much creativity as possible. At least one page.
6. Section 6 (Conclusion): Conclude by explaining the usefulness of your agent. At least half a page.
7. References. These should be listed in APA style (at least 3). Online encyclopedias must not be used as references, but you could use them to learn more about your area. References include descriptions of how the parent systems work e.g. basic function of a helicopter. Use a bibliographic tool to create a database of references and insert them in your word processing document.
8. Formatting and appearance of report:
9. Submit your file as a pdf (at least 6 pages, excluding the cover page), together with the two files

from Part A in a single .zip archive.

Table 1: Assignment of Agent Topics to Students

ID	Agent
201404472	Intelligent Software Installation Agent (Installs software and tests it in order to ensure it is correctly installed).
201501341	Intelligent Drone (Self-flying drone that also handle cargo deliver and pick and and battery exchange).
201503605	Hear Disease Prediction System (diagnoses heart problems)
201506116	Home Robot Assistant (learns about users, interacts with them, monitors house and pets, etc.)
201600689	Kids Robot (interacts with children and helps to improve their social, emotional, and cognitive skills)
201601976	Virtual Travel Assistant (discover prices for trips, suggest travel destinations, books transport and accommodation)
201604466	Robotic Assistant Surgeon (Assists surgeon to perform surgery)
201604516	Go-Playing Agent (agent plays go and improves its performance by playing against itself)
201700108	Humanoid Robot (chats, has facial expressions, learns about users, etc.)
201700174	Drowsy Driving AI (Detects when driver is not paying attention to the road and takes action).
201700590	Cargo Bot (Robot that moves around on its own and carries cargo such as shopping)
201700939	Automated Excavator (Detects objects on site, operates excavator machinery, displays information.)
201701145	Security Drone (Self-flying drone that can detect and monitor suspicious activity).
201701407	System Performance Agent (Provides insight on various bottlenecks in computer systems, advice on how to resolve them)
201701426	Conversational Marketing System (communicate with customers and include them in marketing campaigns)
201701769	AI Assistant with Personality (communicate with user, learns about user, and becomes like the user)
201702201	Automated Retail Agent (Has hardware to support automated retail stores, with self-service checkout, etc.)
201702517	AI Dietician (acts as diet consultant for humans, recommends diets and related advice)
201703230	Automated Crane System (Detects objects on construction site, operates crane, displays information.)
201703498	Personal Assistant (play music, does shopping, ask about weather, etc.)
201704757	AI Desktop Partner (chats with user when he is bored)
201803471	Student Information Chat Bot System (answers enquiries about college related activities)

~~~~~~ END OF END-OF-COURSE ASSESSMENT ~~~~~~
~~~~~~