Introduction to Machine Learning (67577)
Hackathon 2021

Teacher: Dr. Matan Gavish

TAs: Gadi Mintz, Asaf Yehudai, Gilad Green, Yuval Lahav

Tzars: Varda Zilberman, Eliran Elisha

Second Semester, 2021

Contents

1 General Instructions

Welcome to the IML Hackathon 2021! Please read carefully the entire document before selecting a challenge to work on.

- On Wednesday 02.06.2021 7pm we all meet in the auditorium for the reveal of this years challenges. The hackathon ends on Friday 04.06.2021 at 11am.
- During the hackathon you will be working in teams of three or four students only.
- The data of both challenges is available in the following link or on the course Moodle website.
- After reading through the entire document select **one** of the chllenges below, and solve it as best as you can. All the materials for the tasks are available on the course Moodle.
- To ensure we are able to run your code without errors we require all submissions to define and use a virtual environment. Submit the environment's `requirements.txt` file. Instructions on the virtualenv and the freeze command can be found here and here. The virtual environment will allow you to install via the command `pip install <package_name>` and use packages that are not installed on the CS computers.
- Solutions must be written in Python3 (the tester will be using Python 3.7, so please ensure your code works with this version). To use Python 3 on CS computers, use python3 instead of python.
- Submission of your solution is done through the course Moodle. Submit a single zip file named `id_number.zip`, where `id_number` is a 9-digits ID of one team member. The file should include:
  - A folder named `task1` or `task2`, depending on the challenge you choose. This folder should include all your code and supplementary files. Follow those carefully, as there will be automatic tests.
  - `README.txt` — contains a file list and a brief description of each file.
  - `USERS.txt` — contains the logins and IDs of all the team members. Use one line per student, in the format login, ID.
  - `project.pdf` -– a written description of your project, up to 2 pages long, as a PDF file. Explain your solution, describe your work process, and do your best to justify any design decisions you have made. Feel free to include supporting figures if you want.
  - `requirements.txt` — the required packages for your virtual environment, as described above.
- When submitting on Moodle, we will run a presubmission script on your project. The goal of this script is to check that all required files exits, that we are able to install all your requirements, and that we are able to run your code. You **must** pass the presubmission in order to be graded on your work.
- Make sure to write good, clean and documented code. Our ability to understand what you are performing and why is important for receiving a high grade.
- We advise to begin with finding a basic solution that works. Then try to improve it as much as you can in the given time.

1.1 Evaluation

For both challenges your grade will be determined by two factors (in descending order of importance):

1. The quality of your prediction on the test set. This will be based on a ranking between the performance of all participants, where any team that will produce a model that has some

reasonable performance ('reasonable' depends on which task you chose) will receive a grade of 80 and the top ranking teams will receive a grade of 100, and there will be some distribution between these two extremes.

2. The quality and depth of your written description of your work that you wrote in the PDF. We will grade your written description based on the following:
   - Did you describe the dataset, and any challenging characteristics it has?
   - Did you describe (briefly) the data cleaning and preprocessing?
   - Did you describe the considerations that guided your design of learning systems?
   - Did you describe (briefly) various methods you tried and the results you obtained?
   - Did you describe the learning system you ended up using?
   - Did you provide a prediction (and explanation) of the generalization error you expect your system to have?

Good Luck!

1.2 Verify Before Submission

IMPORTANT: Our tests have **zero tolerance** hence you have to follow the requested formats. Submission that fail to comply with the requested format will receive zero points.

- You are obligated to submit files in the defined file folder structure as below.
- Output data has to be in the matching format.
- Your code should only call files present in the submitted folder. When doing so use a relative path. For example : `"weights.txt"` is good while `"C:\Users\name\IML\weights.txt"` is bad.
- You are required to submit a `requirements.txt` file.
- Your code cannot throw any exception or error when running.
- Your code has to finish prediction (including any necessary loading or initialization) in a time frame of 5 minutes when running on the CS computers.
- After zipping your project, it should have a maximum size of 20 MB.
- Submit only **one** of the challenges (see directory structure).
- Submit the zip file through the Moodle of only **one** of the team members.
- You must use the following directory structure in your submission zip file:

File or directory	Description
<student-id>.zip	Submission zip file
├── USERS	
├── README	
├── project.pdf	
├── task1	***Only if submitting task 1***
│ ├── requirements.txt	
│ ├── regerssion.py	
│ └── *<other source files>*	
└── task2	
├── requirements.txt	***Only if submitting task 2***
├── classifier.py	
└── *<other source files>*	

2 Hackathon Challenges:

Please note: both challenges are equally easy (or equally hard). Don't choose the challenge that looks easier - choose the challenge that looks more interesting and will be more fun to work on. You are far more likely to have good results on a challenge you find fun and interesting; Also, performance you can hope to achieve are different between the two challenges.

2.1 Challenges 1: Movies!

The movie business is a huge business with a global box-office revenue of billions of dollars a year. However, not all movies are money makers, and in reality many movies end up with a loss. The cost of making a movie is very high and includes costs of script, development, salaries of the actresses and actors, production, special effect, marketing, and so on. Can we say something about the revenue of a movie, and the viewer ranking of the movie, before it is officially released? Are there parameters which may help us answer this question?
Your task is to learn to predict the box-office revenue of a movie and its viewer ranking, given some details about it.

Dataset:

The dataset contains around 5,000 movies, each with 22 features, as detailed below. Note that part of the features are in a JSON format.

Features Descriptions:

- **id** - Unique identifier for the movie - int
- **belongs_to_collection** -description of a collection this movie belongs to (if there is one) - JSON {id - int, name - string}
- **budget** - the movie budget - int
- **genre** - the movie genres - list of JSONs {id - int, name - string}
- **homepage** - link to movie homepage - string
- **original_language** - original language of the movie - string (2 letters)
- **original_title** - title of the movie in the original language - string
- **overview** - movie overview - string
- **vote_count** - number of viewers ranked the movie - int
- **production_companies** - list of production companies - list of JSONs {id - int, name - string, orig_country - string}
- **production_countries** - list of production countries - list of JSONs {iso_3166_1 - string, name - string}
  - **iso_3166_1** - codes for the representation of countries names (2 letters).
- **release_date** - date of official release - date
- **runtime** - the time (in minutes) between the starting of the movie up to the end of the credits scene - int
- **spoken_languages** - languages that are spoken in the movie - list of JSONs {iso_639_1 - string, name - string, english_name - string}
  - **iso_639_1** - codes for the representation of languages names (2 letters).
- **status** - is movie released / in production / planned - string
- **tagline** - description of the movie in a few words - string
- **title** - title of the movie (in English) - string
- **keywords** - keywords in the movie - list of JSONs {id - int, name - string}

- **cast** - list with the actresses details - list of JSONs {gender - int, id - int, known_for_department - string, name - string, cast_id - int, character - int }
  - **gender** - 0 = not set, 1 = Female, 2 = Male
- **crew** - list of the crew details - list of JSONs {gender - int, id - int, known_for_department - string, name - string, department - string, job - string}

Response:

- **revenue** - the movie box-office revenue (in dollars) - int
- **vote_average** - average rank of the movie (ranked by viewers) - float between 0 and 10, with one digit after decimal point

Code Requirements:

You need to implement the module `regression.py` (which can be found in the Moodle). This module has a method `predict` that receives a csv file with movies details (same format as the training data), and predicts for each movie its revenue and average ranking. You should return two python lists - one with the predicted revenues and one with predicted rankings. Do not change the signature of this method. **NOTICE:** Since we are not going to train your model, you are required to load your trained model in this method. Please verify this method works as expected (with all the additional files it is required - s.a. the weights file)

Performance Measure:

Evaluating your model will be with respect to the RMSE function:

$$RMSE(\mathbf{y},\hat{\mathbf{y}}) = \sqrt{\frac{1}{m}\sum(y_i - \hat{y}_i)^2}$$

Tips:

In order to represent text as a feature vector you will need to be creative. We suggest to start by examining the 'sklearn.feature_extraction.text' package. One simple way of representing a document as a vector is:

- **Bag of Words (BoW):** In this representation, a text is represented by a vector $\mathbf{x} = \{0,1\}^n$, where n is the number of words we 'know' and $x_i = 1$ iff the word corresponding to index i appears in the text.

2.2 Challenges 2: Help the Chicago Police Prevent Crime!

Police departments around the world are building machine learning systems, which attempt to predict time, place and type of crimes. With an effective learning system, the police might be able to direct patrols to certain locations at certain times that are more prone to occurrence of a crime, and hopefully prevent it from happening in the first place. You'll receive a dataset of 35,000 crimes that happened in Chicago. A sample contains location of the crime (x,y), time, type, and more - see below.

Challenges:

In this task there is a primary challenge and a secondary challenge. The primary challenge is classification of crime type: you are required to build a learning system that, given crime feature vectors (see below description) predicts which kind of crime it is, from 5 classes. The secondary challenge is crime prevention. Every day you can send 30 police cars to the city - you direct a car to

a specific location and a specific time of day. If a crime was about to happen up to 500m from the location you specified and up to 30 minutes before or after the time you specified, you prevented a crime! You are required to build a learning system that, given a date in the future, will output 30 (x,y,time) combinations - where time is during that day from midnight to midnight. Police cars will be sent to these locations at these times.

Dataset:

This dataset reflects reported incidents of crime that have occurred in Chicago over the past year, minus the most recent seven days of data. The data includes features such as the time (date and time of the day) and location (in several spatial resolutions) of each crime. The labels in this task are the Primary Types of each crime.
The five crime Primary Types in the given dataset are:
'BATTERY', 'THEFT', 'CRIMINAL DAMAGE', 'DECEPTIVE PRACTICE', and 'ASSAULT'.
The dict is :
{0: 'BATTERY', 1: 'THEFT', 2: 'CRIMINAL DAMAGE', 3: 'DECEPTIVE PRACTICE', 4: 'ASSAULT'}
recommend to map between index and Primary Types create a dict and use with this command:
list(map(dict_crimes.get,res_training))
for reverse operation create a reverse dict
Each crime also has a time and location.

Code Requirements:

You need to implement the module `classifier.py`. This module has two methods. One method is `predict` that receives a `csv` file with the feature columns (as in the training set) of crimes and predicts for each one which crime has occurred. More specifically, during test time it will return a list (or a one dimension numpy array) of labels (ints between {0-4} for the 5 classes). The second method is `send_police_cars` that receives a date (in the same date/time format as the dataset - the time is ignored) and outputs a list with 30 entries. Each entry is a tuple of the form (x,y,time) where "x" and "y" are of the same format as the dataset, and time is a date/time stamp of the same format as the dataset, where "time" falls in the specified day. Do not change the signature of these methods. NOTICE, since we're not going to train your model, you are required to load your trained model in this method. Please verify this method works as expected (with all the additional files it is required - s.a. the weights file)

Label descriptions:

- **Primary Type** - The primary description of the IUCR code indicating which type of crime took place - string

Feature descriptions:

feature name - description - type

- **ID** - Unique identifier for the record - int
- **Date** - Date when the incident occurred. This is sometimes a best estimate - Date and Time
- **Year** - Year the incident occurred - int
- **Updated On** - Date and time the record was last updated.

- **Block** - The partially redacted address where the incident occurred, placing it on the same block as the actual address - string
- **Location Description** - Description of the location where the incident occurred - string
- **Arrest** - Indicates whether an arrest was made - boolean
- **Domestic** - Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act - boolean
- **Beat** - Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has a dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a police district. The Chicago Police Department has 22 police districts - string
- **District** - Indicates the police district where the incident occurred - string
- **Ward** - The ward (City Council district) where the incident occurred - int
- **Community Area** - Indicates the community area where the incident occurred. Chicago has 77 community areas - string
- **X Coordinate** - The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block - float
- **Y Coordinate** - The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block - float
- **Latitude** - The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block - float
- **Longitude** - The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block - float

Performance Measure:

For the primary task (classification), we will use simple misclassification loss to evaluate your performance on new data. For the secondary task (crime prevention) we will simply count the number of crimes you prevented - where, again, you prevented a crime if you sent a police car to within 500m (in space) and 30 minutes (in time) from where a crime was about to happen. >

Good Luck and Have Fun!