CPSC 340 Machine Learning Take-Home Final Exam (Fall 2020)

Answer: CSID: c6n1b. Student Number: 32356362.

Instructions

This is a take home final with two components:

1. an individual component
2. a group component for groups of up to 5. Note that your final and midterm groups will not be allowed to have any overlap in membership besides you.

You may work on the group components as an individual, but it is to your advantage to team up with others. There will be no leniency in grading for smaller groups or individual work.

Submission instructions

Typed, LaTeX-formatted solutions are due on Gradescope by **Wednesday, December 16 at 11:59pm PST**.

- Please use the `final.tex` file provided as a starting point for your reports.
- Each student must submit question 1 individually as a pdf file named `question1.pdf`. Include your CS ID and student ID. Upload your answer on Gradescope under **Final Exam Question 1**.
- Each group should designate one group member to submit their solution to question 2 to Gradescope using its group feature (`https://www.gradescope.com/help#help-center-item-student-group-members`). Submit a zip file for question 2 under **Final Exam Question 2**. Include each group member's CS ID and student ID.

Question 1 - Individual [60/100 points]

Recall the MNIST data set from assignment 6 which could be downloaded at `https://github.com/mnielsen/neural-networks-and-deep-learning/blob/master/data/mnist.pkl.gz`. Go ahead and download this dataset, since we will be using it for this question.

MNIST contains labelled handwritten digits (i.e. 0 to 9) with 60,000 training examples and 10,000 test examples. It is a widely used dataset and with known error rates for several machine learning methods encountered in class. We will be using `http://yann.lecun.com/exdb/mnist/` as a reference for test errors.

For this question, you will implement 5 machine learning methods from class and apply them to the MNIST dataset in order to do supervised classification of digits, with the goal of minimizing the test error. The approaches to be implemented and employed are one example from each of the following types:

1. k-nearest-neighbours (KNN)
2. linear regression

3. support vector machine (SVM)
4. multi-layer perceptron (MLP)
5. convolutional neural network (CNN)

This question will be answered in a report format, provided at the end of the exam LaTeXfile `final.tex`. You will have to provide test errors achieved using your implementations, calculated as the percentage of incorrectly labeled test examples (using the default test set provided in the MNIST dataset partition). As an example, results from `http://yann.lecun.com/exdb/mnist/` for each of the above models (with particular hyper-parameter settings) are shown below:

Model	Error (%)
KNN	
linear regression	
SVM	
MLP	
CNN	

Assignment 6 provides code that will load the MNIST dataset into a training set and a test set (if you stored the dataset in a separate directory called `./data/`). The rest of the code (model, training, and testing procedures) must be written by you. You are not permitted to use built-in models (e.g. from PyTorch or scikit-learn), but we encourage you to use code from your assignments. Remember that in past assignments, you have had to implement all of the models listed except for CNNs.

Bundle your code along with a `.pdf` generated from the filled in LaTeXreport into a `.zip` file and submit it to Gradescope. Marks may be taken off for very messy or hard to read code, so make sure to use descriptive variable names and include comments where appropriate. Since we are also marking based on test error, you are expected to only evaluate performance on the test set in the partition provided.

Question 2 - Group [40/100 points]

This part of the final is a group project that takes place on Kaggle at `https://www.kaggle.com/t/b1d0feabe2b94cc19c85a5694b606fe9`. You can sign up for a new account or re-use the account for your midterm; Again, note that the Kaggle servers may be in the US, so bear this in mind. We recommend that for data protection purposes you use a non-identifiable (but ideally hilarious) team name. You will link your group members to your team name in your submission document.

You are not allowed to use any software that you did not develop yourself. There is one exception to this: you may use homework support code and code that you wrote yourself for your homework.

Similar to the midterm, your mark for this part of the final will be based on the score from Kaggle for your test set predictions (see below and the Kaggle competition pages), a written report that explains your findings, and your code. Your report must LaTeXformatted and follow the format given in the answer template for Question 2 below.

Imagine you are an intern working at a self-driving car company and you are writing a behavior prediction algorithm that predicts the trajectory of a car of interest. Your task is to create an ego-centric predictive model of vehicle motion conditioned on past positions (represented as coordinates ($x_{j,t}$, $y_{j,t}$) for a time step t) of both the ego vehicle and other agents moving about the same intersection. Specifically, given one second of vehicle position data for the ego vehicle and (up to) the ten nearest agents to the ego at the point in time where prediction starts, you need to predict the future position of the ego vehicle 3 seconds into the future.

Data

This dataset we are providing you contains large amounts of road user trajectories (e.g., vehicles, pedestrians) around an unsignalized intersection in the US (right-of-way for motorists, bicyclists, and pedestrians; not controlled by a traffic signal). The data we are providing for this task is a small subset of the data from the Interpret Challenge at `http://challenge.interaction-dataset.com/prediction-challenge/intro`.

`Training/validation data:` You will be provided with 2307 + 523 pairs of `CSV` files (i.e. `X_001.csv` and `y_001.csv`) as your training and validation data respectively, where $\mathbf{X}_j$ is a snapshot of up to the ten nearest vehicles at an intersection over one second in the past, and $\mathbf{y}_j$ is the trajectory of the ego vehicle over the next three seconds. Note that each ($\mathbf{X}_j$,$\mathbf{y}_j$) pair is a training/validation example on one ego vehicle, so it must contain exactly one ego vehicle in each file.

$\mathbf{X}_j$: Each training/validation $\mathbf{X}_j$ contains a column corresponding to the time step (the first column) and 60 more columns that contain information regarding up to 10 agents in the intersection. We have included a table below that mimics the format of each $\mathbf{X}_j$ file (in the table i ranges from 0 to 9 and numbers in tuples are randomly chosen) and instructions on how to read it.

Sorry for this notational clash, but note that ($\mathbf{X}_j$,$\mathbf{y}_j$) are different from the coordinate ($x_{j,t}$,$y_{j,t}$).

time step t	id$_i$	role$_i$	type$_i$	x_i	y_i	present$_i$	**...**
-1000	**1**	**agent**	**car**	**-4.44**	**0.025**	**1**	**...**
...	**1**	**agent**	**car**	**...**	**...**	**1**	**...**
0	**1**	**agent**	**car**	**0.0**	**0.0**	**1**	**...**

Table 1: Table of $\mathbf{X}_j$ Files

Instructions:

- **time step(ms)** represents the time the agent appears in the snapshot. There are in total ten time steps in the past from -1000ms (one second ago) to 0ms (current).
- id$_i$ represents the unique ID of the agent starting from 1.
- role$_i$ is used to identify the ego vehicle, if role = agent: ego vehicle ; if role = others: other agents.
- type$_i$ represents the types of tracked agents. For example, it can be a car, a pedestrian and so on.
- **(**x_i**,** y_i**)** represents the ($x_{i,t}$, $y_{i,t}$) position of the agent i at the current time step t. Each ($x_{i,t}$, $y_{i,t}$) has been transformed to lie in the ego vehicle's reference frame at time 0 with forward towards the "engine" being the positive "y-axis" and towards the passenger door (in a left hand drive vehicle) as positive "x-axis". As a result, you will find that ($x_{i,0}$, $y_{i,0}$) for the ego vehicle (role = agent) is always (0, 0) at time 0. You may find Figure 1 below to be helpful in visualizing this coordinate system.
- present$_i$ is a binary value that represents whether an agent is in the intersection at that time step.

Note that we pick 9-nearest neighbours as other agents (role = others) around an ego vehicle. If there are less than 9 neighbours, the extra columns will be filled with 0s.

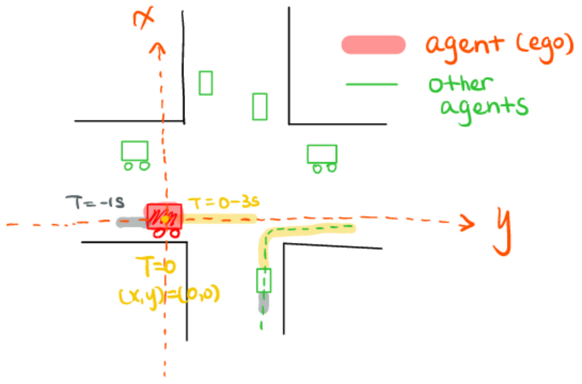


Figure 1: Coordinate System Description

$\mathbf{y}_j$: Each training/validation $\mathbf{y}_j$ is the trajectory of the ego vehicle over the next three seconds in the same coordinate system, so it only contains **x**, **y** and **time step** (0ms to 3000ms).

`Testing data:` The testing data format is exactly the same as the training/validation format, except you are not given $\mathbf{y}_j$, as that is what you need to predict. Specifically, you are given 20 test $\mathbf{X}_j$ `CSV` files, each of which contains the information of an ego vehicle and its surrounding agents over the past one second, and your task is to predict the future trajectory of the ego vehicle over the next three seconds. Note that the final result you hand in shouldn't be separated $\mathbf{y}_j$ files, but should be combined into one table according to the sample submission given on Kaggle.

Finally, there exists a python script to visuzalize the dataset at `https://github.com/interaction-dataset/interaction-dataset`, and instructions on how to use it can be found on Kaggle.

Prediction Task

For each testing data `X_###.csv`, you only need to predict the ego vehicle's future position over the next three seconds. You are not required to make predictions for other nearby agents, but you may find it helpful to do so. You are free to construct any feature set you wish and train as many models as you want in order to solve this task. Your test accuracy will be measured (and compared) against the actual positions using Root Mean Square Error (RMSE) as described in the Kaggle competition.

Submitting Your Results

Similar to the midterm Kaggle competition, your final grade for this question depends on multiple things. Rubrics will grade for test accuracy (i.e. your Kaggle ranking), a description of your pipeline (e.g. data preprocessing, feature engineering, hyper-parameter tuning, evaluation such as cross validation, ...), report writing, code readability, and reflections.

`Kaggle submission`: You will upload your predictions to the Kaggle server. Please refer to the sample submission file on Kaggle for the correct format.

`Gradescope submission`: Bundle your code along with a `.pdf` generated from the filled in LaTeXreport skeleton into a `.zip` file and submit it to Gradescope. Marks may be taken off for very messy or hard to read code, so make sure to use descriptive variable names and include comments where appropriate. Since we are also marking based on test error, you are expected to only evaluate performance on the test set in the partition provided.

Skeleton for Question 1 Answer

1 Introduction (3 *points*)

Three sentences describing the MNIST classification problem.

Answer: Each training example (X) is the flattened pixel value of a hand-written digit picture. Each y is the corresponding digit (0-9) in the picture. The classification task is to recognize the digit in a new hand-written picture. There are 50000 training data, 10000 validation data and 10000 test data.

2 Methods (40 *points*)

2.1 KNN (8 *points*)

Three to four sentences describing the particulars of your KNN implementation, highlighting the hyperparameter value choices you made and why.

Answer: For the implementation, I used the code from previous assignment. I speed up the algorithm by replacing the nested for-loop in cosine distance with numpy array operations. I used validation set to choose the hyperparameter k. I also used the validation set to compare between the cosine distance and the euclidean distance and found that cosine distance performs better generally. I chose $k = 4$ by testing on the validation set. Choosing the hyperparameter that gives the lowest validation error will probably result in over-fit on the validation set though.

2.2 linear regression (8 *points*)

Three to four sentences describing the particulars of your linear regression implementation, highlighting the hyperparameter value choices you made and why.

Answer: For the implementation, I used the multi-class softmax loss function from previous assignment. Compared with one-vs-all logistic loss which train each class' W independently, softmax encourages calibrations among all columns. I used the log-sum-exp trick from the Piazza post (https://piazza.com/class/kenhpni6f94x4?cid=804) to deal with the overflow issues for both the exponential and the logarithm functions. For the hyperparameters, I used the validation set to tune the max iterations. I chose max iteration = 300, which gives the lowest validation error.

2.3 SVM (8 *points*)

Three to four sentences describing the particulars of your SVM implementation, highlighting the hyperparameter value choices you made and why.

Answer: For the implementation, I used the multi-class SVM. The main changes are in funObj function. I used a nested for-loop to add up the loss function and the gradient at the same time. For the calculation of the gradient, I used the sub-gradient method from the Piazza post (https://piazza.com/class/kenhpni6f94x4?cid=770). And then I used the validation set to tune the hyper parameters lambda and max iterations. As always, hyperparameters that give low validation errors tend to have larger Eapprox. In the end, using the method from the previous questions, I chose lambda = 0.01 and max iteration = 500.

2.4 MLP (8 *points*)

Three to four sentences describing the particulars of your MLP implementation, highlighting the hyperparameter value choices you made and why.

Answer: For the implementation, I used the neural net from a6 which has one layer and uses sigmoid activation function. From a6 I found that sigmoid performs the best among some commonly used activation functions by using scikit learn library so I'll keep using it here. I used the validation set to tune the hyperparameters lambda, maximum number of iterations and hidden layer size. Generally, validation error decreases with larger max iteration and hidden layer size. In the end, I chose lambda = 0.025 , max iteration = 1000 and hidden layer size = 150.

2.5 CNN (8 *points*)

Three to four sentences describing the particulars of your CNN implementation, highlighting the hyperparameter value choices you made and why.

Answer: A CNN generally has the following components: several convolution layers, with specified filters with different sizes and strides. There are also different methods to deal with the edges of the images, such as filling with zeros. And then there are usually one or more max pooling layers that capture the most important features per sub-images of a certain size. We can then flatten out the images into one array. Now we can use regular neural networks to classify the images. In my opinion, this is a composite of the methods we've implemented above: A CNN consists of MLPs, and each layer of the neural network has a classification model, like SVM.
There are many hyperparamters that we can tune. I would use the validation set as before and observe the how the error rate changes. But training one CNN model takes too long so I just used the default ones.
Citation: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

3 Results (10 *points*)

Model	Their Error	Your Error (%)
KNN	0.52	2.87
linear regression	7.6	7.43
SVM	0.56	8.12
MLP	0.35	1.84
CNN	0.23	1.84

4 Discussion (7 *points*)

Up to half a page describing why you believe your reported test errors are different than those provided (and "detailed" on the MNIST website).

Answer: First of all, I did not perform any pre-processing on the image, such as normalization, reducing noise etc. So there could be messy parts of the images influences the learning.

Also, the distance function, loss function, regularization term and the number of layers I used in my implementation might be different from the ones that give the error rates on the website. I might be also missing some functions that can improve the performance, such as kernel for SVM. For CNN, there are more choices could be done, such as the filters, strides, filter sizes, pooling layer, activation function, classification functions, the number of the layers and all the hyper-parameters of those models.

I believe doing a cross-validation with a larger range for the hyperparameters could also help with finding the best ones. Also, I'd like to try using a different method to choose hyperparameters. However, since the dataset is pretty large, it takes too much time to iterate over all possible combinations.

Skeleton for Question 2 Answer

1 Team

Team Members	*all team member names and csids here*
Kaggle Team Name	*your Kaggle team name here*

2 Introduction (3 *points*)

A few sentences describing the autonomous driving prediction problem.

Answer: In the training data, there are data for objects around an intersection of 1000ms in total. For each 100ms, there are a position of an ego car and the positions of up to ten nearest objects (nearest wrt. the start of the 1000ms time frame) to the ego car. We need to predict how the ego car will move in the next 3000ms in total, one position per 100ms.

3 Summary (12 *points*)

Several paragraphs describing the approach you took to address the problem.

Answer: The thing that our model needs to learn is the behaviour of the ego car given the movement of itself and that of the surroundings. I believe there are 3 factors that influence the decision made by the ego car the most:

1. The type of the n nearest objects. The type matters because a car's movement is noticeably different from that of a bicycle or a pedestrian. Therefore depending on the types, the weight of the impact on the ego car's movement is different.
2. The distance of the objects from the ego car. The distance matters because intuitively the closer the object is to the ego car, the more weights it gives to the decision of the movement of the ego car.
3. The speed and accelerations of the ego car and the objects. This two values can largely describe the physical movement, including the moving directions, of the objects. At the end of the day, if the objects move very fast towards the ego car, this factor should be more important for the decision made by the ego car than a pedestrian that's just standing still. What's more, the speed and acceleration of the ego car itself is a crucial factor for its future movement. For example, if the ego car is moving very fast, even if there's an emergency happening, it's still quite impossible to slower itself to a speed of 0 in the next second.

So in summary, the question for the model is WHAT the decision is given the situation and HOW MUCH can the ego car change its movement depending on its original speed and accelerations.

After understanding what the question is, we can now zoom into more details. To make one prediction, we use the information of the past $1100ms$. In other words, our time frame is always $1100ms$. For example, to predict the position of $t = 100ms$, we use the data in the range $[-1000ms, 0ms]$; to predict the position of $t = 200ms$, we use the data in the range $[-900ms, 100ms]$. So when we have a new prediction at $t = t_p$, we would update the feature spaces in the following way: remove the ego car data at $t = t_p - 1100$, and add the ego car data at $t = t_p$ at the end of the matrix. Then we re-train the model with new data and make a new prediction at $t = t_p + 100$.

4 Experiments (15 *points*)

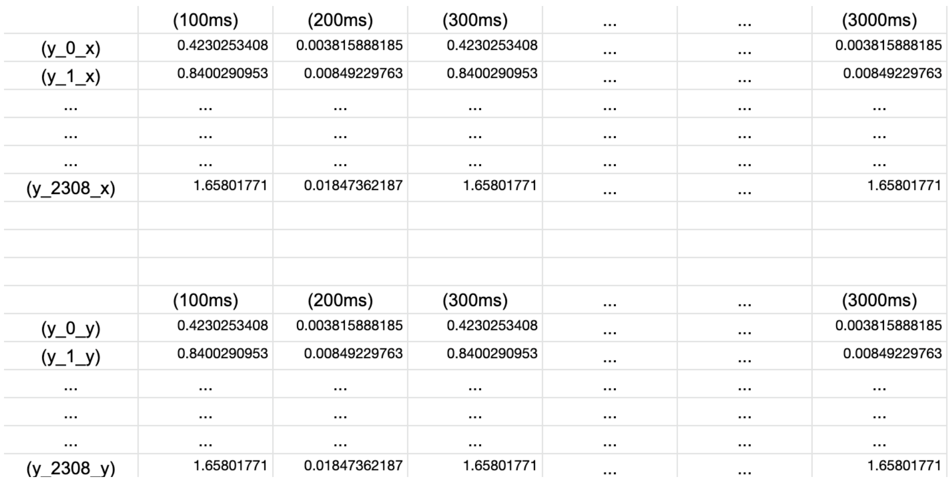
Several paragraphs describing the experiments you ran in the process of developing your Kaggle competition final entry, including how you went about data prepossessing, feature engineering, model, hyper-parameter tuning, evaluation, and so forth.

Answer:

1. Data pre-processing:

   (a) Loading data: each X_n.csv is a training example and y_n.csv is the corresponding prediction value. For each training example, we merge the 11 rows (time step -1000ms to 0ms, one row per 100ms) into one rows. We then concat 2308 rows into one dataframe to form our X. We duplicate it to have one X_x for model of the x coordinate and one X_y for model of the y coordinate.

   For each training example, there are 30 predicted coordinates (x, y) from time 0s to 3000ms. Since we predict the x and y separately, we keep one y_x and one y_y for each training example and each time step. And then we transpose the column into a row. Lastly, we concat 2308 rows to form a big dataframe. In this way, we can easily access the prediction value of all training data at time step $= t$ by $y[t]$. See the below chart.



	(100ms)	(200ms)	(300ms)	...	...	(3000ms)
(y_0_x)	0.4230253408	0.003815888185	0.4230253408	...	...	0.003815888185
(y_1_x)	0.8400290953	0.00849229763	0.8400290953	...	...	0.00849229763
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
(y_2308_x)	1.65801771	0.01847362187	1.65801771	...	...	1.65801771

	(100ms)	(200ms)	(300ms)	...	...	(3000ms)
(y_0_y)	0.4230253408	0.003815888185	0.4230253408	...	...	0.003815888185
(y_1_y)	0.8400290953	0.00849229763	0.8400290953	...	...	0.00849229763
...	...	...	...	...	...	...
...	...	...	...	...	...	...
...	...	...	...	...	...	...
(y_2308_y)	1.65801771	0.01847362187	1.65801771	...	...	1.65801771

   (b) Sorting the feature space X_x and X_y: we need to be consistent on the feature space across all training example. For example, the first entry for each training example should be the coordinates of the ego car at the start of the time frame, which is not necessary the case for the raw data. We adjust the feature spaces using the following rule: first entry is the ego car information, and then sort the objects in descending distance to the ego car, and repeat this for all time step in a given time window.

   (c) Dropping some training examples. It's possible that some nearest objects to the ego car is not around the intersection (present = 0). There are also some objects that have all entries being zero, which are just place holders for the sake of dimension. I will regard these two kind of objects as invalid object, and if a training example has too many invalid objects, I will just NOT use that training example.

   (d) Drop unneeded columns, only keeping all (x, y).

   (e) Construct time series matrix. For each intersection, there are 30 rows by the the following construction: 1. keep all the other objects' positions the same; 2. move the time window rightwards per row. For example, in the first row of an intersection, there are positions of : all nearby objects

+ ego car in the range of -1000ms - 0ms; Then the next row has positions of : same nearby objects + ego car in the range of -900ms - 100ms; Repeat this and the last row of this intersection has positions of: same nearby objects + ego car in the range of 1900ms - 2900ms

2. Feature engineering: I used the forward selection method taken from my implementation from midterm. A hyper-parameter *num_feature* is passed in to define how many features I want to select.
3. Model: I used the most basic linear model (least square).
4. Hyper-parameter tuning: Although a validation set is given, only using validation set to adjust the hyperparameter will probably results in overfitting the validation set. So we use a 5-fold cross validation to adjust the hyperparameter $num_{f}eature$ first and then do a slight adjust using the originally given validation set.
5. Evaluation: Use the RMSE formula.

5 Results (5 *points*)

Team Name	Kaggle Score
the name of your team	*your kaggle score*

6 Conclusion (5 *points*)

Several paragraphs describing what you learned in attempting to solve this problem, what you might have changed to make the solution more valuable, etc.

Answer: From this project, I learnt that pre-processing is probably the most important part for machine learning, especially when the data are from real life. It's because usually the data is messy and in consistent. It would be crucial to make sure that all data we are using are valid before moving on with any manipulations. Visualization is also important because sometimes we can eyeball which model suits better from the graphs or animations.

If given more time, I'd like to try out different feature selection methods and machine learning models. Also I'd like to do a more comprehensive cross-validation on all the hyper-parameters. For the code itself, I'd like to change it to first create a large CSV for all the data instead of iterating through and reading each file every time we run the model.

7 Code

Include all the code you have written for the autonomous driving prediction problem.