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

Instructions

This is a take home midterm 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 **Monday, October 26**.

- You must use the latex skeletons provided later in this document to format your reports. You may include additional, terse, comments in additional sections beyond those specified.
- 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 **Midterm 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.ca/help#help-center-item-student-group-members`). Please hand in your work separately for question 2 on Gradescope. Submit a zip file for question 2 under **Midterm Exam Question 2**. Include each group members' CS IDs and student IDs.

Question 1 - Individual [70/100 points]

In this question, you are provided with a dataset consisting of tweets from the two U.S. presidential candidates - Donald Trump, and Joe Biden. The features of each word in each tweet were extracted using the word2vec algorithm (see `https://en.wikipedia.org/wiki/Word2vec`; word2vec will be covered later in the term, understanding how it works is not required to be able to complete this question) then combined via averaging to form a fixed-length feature vector per tweet. Your task is to design a number of binary classifiers which, given a tweet vector, predict whether the tweet was authored by Donald Trump or Joe Biden. You are given the following files.

- `wordvec_train.csv` - dataset of word2vec features. Each data entry has 200 numerical features, and a label (1 for Donald Trump, and 0 for Joe Biden). This is your training data.
- `tweet_train.csv` - dataset of tweets prior to feature extraction (you don't need to use this, it is provided just for own amusement). These correspond to features in your training data.
- `wordvec_test.csv` - dataset of word2vec features. This is your test set.
- `tweet_test.csv` - tweets corresponding to test set features.

You will implement the following classifiers.

1. A random forest classifier.
   (a) Which uses the Gini index as a splitting score.
   (b) Where the K-means algorithm is used to quantize each feature, determining the thresholds to search over for each feature.
2. Naive Bayes for continuous features.
3. K nearest neighbours with cosine similarity as the distance metric.
4. A stacking ensemble consisting of your classifiers from parts 1-3, where a decision tree is used as a meta-classifier.

All of these concepts are familiar to you and have been introduced in class. As the details of these specific models are "new" to you, we elaborate on them in the following sections.

Gini Impurity and Gini index

Gini Impurity measures the probability of misclassifying an observation. Consider a classification problem with J classes. Let p_i be the fraction of items labeled with class i at a given node. Then the Gini impurity at the node is given by,

$$\textbf{Gini Impurity} = G(p_1, \ldots, p_J) = \sum_{i=1}^{J} p_i(1 - p_i). \tag{1}$$

The **Gini Index** is the weighted sum of Gini Impurity for a split in our data. Let p_i^ℓ be the fraction of items labeled with class i in the left side of our split, and p_i^r be the fraction of items labeled with class i in the right side of our split. Moreover, if we take N_t to be the total number of data, then N_r is the number of data in the right side of the split, and N_ℓ is the number of data in the left side of the split. In the end, the Gini Index of the split is given by,

$$\textbf{Gini Index} = \frac{N_\ell}{N_t} G(p_1^\ell, \ldots, p_J^\ell) + \frac{N_r}{N_t} G(p_1^r, \ldots, p_J^r). \tag{2}$$

Finally, we can use the Gini index as a splitting criteria in a decision tree. To do that, first we need to find the split with the minimum Gini index between the set of all the possible splits for each feature. If the minimum Gini index was less than the Gini impurity of the data before the split, then we select the split with the minimum Gini index as the decision rule, otherwise we stop splitting. For more information please see the appendix at the end of this file.

Using K-means to determine thresholds

K-means can be used as a binning algorithm for continuous data. For each feature, you must train a separate instance of k-means. When you are creating your decision tree splits, instead of searching over every possible unique value in the data, you should only consider splitting at each of the k means-derived thresholds.

Continuous Naive Bayes

In our description of Naive Bayes thus far we have only seen it used for problems with discrete features. However, we can use Naive Bayes for continuous features by assuming that for each class, each feature comes from some continuous distribution. For now, we will make the assumption that each of the features come

from an univariate normal distribution. Note that the class conditional independence assumption made in the lectures resulted in computing the product of Bernoulli likelihoods in the decision rule. Here the class conditionally independent likelihoods will be univariate Gaussian likelihoods.

The univariate Gaussian likelihood, or normal distribution, has two parameters - the mean μ, and the standard deviation σ, of the distribution,

$$p(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right). \tag{3}$$

In Naive Bayes, given a data-label pair (x_i, y_i), we construct and then make predictions using the quantity,

$$p(x_i \mid y_i = y_c), \tag{4}$$

where y_c is an arbitrary class label. In the continuous case, for the class c, for each feature d, we would have a normal distribution with parameters μ_c^d, σ_c^d. We also let x_i^d denote the dth feature of data entry x_i. Thus, we make predictions with the following quantity,

$$p(x_i \mid y_i = y_c) = \prod_d p(x_i^d \mid \mu_c^d, \sigma_c^d), = \prod_d \left(\frac{1}{\sigma_c^d\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_i^d - \mu_c^d}{\sigma_c^d}\right)^2\right)\right). \tag{5}$$

In order to estimate $\mu_{c,d}, \sigma_{c,d}^2$ from the data, we make a maximum likelihood assumption. For the normal distribution, this amounts to estimating the mean and the variance of each feature for both classes, and using these parameters to make predictions.

Recall that the maximum likelihood estimators for the mean and variance of the univariate Gaussian distribution are

$$\mu_c^d = \frac{1}{n_c}\sum_{i=1}^{n_c} x_i^d$$
$$(\sigma_c^d)^2 = \frac{1}{n_c}\sum_{i=1}^{n_c}\left(\mu_c^d - x_i^d\right)^2.$$

Hint: When multiplying many small probabilities together as in (5), the product can go to zero very quickly. Instead, you should make your predictions in log space,

$$\log\left(p(x_i \mid y_i = y_c)\right) = -\sum_d \left(\frac{1}{2}\left(\frac{x_i^d - \mu_c^d}{\sigma_c^d}\right)^2 + \log\left(\sigma_c^d\sqrt{2\pi}\right)\right). \tag{6}$$

Cosine similarity

Cosine similarity measures the similarity between two vectors. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction,

$$Similarity(x_i, x_q) = \frac{x_i^T x_q}{\|x_i\|\|x_q\|}. \tag{7}$$

Stacking classifier

Stacking is an ensembling technique that combines multiple classifiers via a meta-classifier. The individual classifiers are trained on the original training data with their individual losses and learning algorithms. Then, the meta-classifier uses the predictions of the classifiers in the ensemble as features and is trained with those as inputs and the ultimate output as its output. For more information see slide 26 in Lecture 7.

Question 2 - Group [30/100 points]

This part of the midterm is a group project that takes place on Kaggle. There are two phases to this competition which come in the form of two essentially identical (but spaced in time) Kaggle competitions. The Phase 1 Kaggle competition is at `https://www.kaggle.com/t/dd5029ad34d14b1eb9791aaeba0a09fd`. The Phase 2 Kaggle competition is at `https://www.kaggle.com/t/b94ffde05677429c848b6517c9e37d2b`. You can sign up for a new account or use an existing one; however, 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 only allowed to use linear regression techniques to solve this problem. No neural networks, support vector machines, etc. 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 you wrote yourself for your homeworks.

Your mark for this part of the midterm will be based on the score from Kaggle for your Phase 2 test set predictions (see below and the Kaggle competitions 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.

The problem you are tasked with in the Kaggle competitions is predicting "forward in time" five days of Covid-19 daily death counts for the entire country of Canada. The ultimate evaluation of your model's performance will only happen *after* the test submission deadline.

Here's how it is going to work. The competition will proceed in two phases. In the first phase competition we will provide you with an *initial training dataset* dataset consisting of 4 different Covid-19-related counts for effectively all countries in the world for all days starting from Dec. 31st, 2019 up to Oct. 5th, 2020 when the initial prediction window starts. The Phase 1 Kaggle competition will be configured to allow you to upload *initial test predictions* consisting of the Canadian daily Covid-19 death totals for the days Oct. 6-16th, 2020. In the second phase competition we will provide you, *on the day before the midterm is due*, with a *second training dataset* that runs up to the 25th of Oct. You will then be tested, after the exam is handed in (Oct. 26th is the due date) on 5 days of predictions (Oct. 26th, 27th, 28th, 29th, and 30th) of daily Canadian Covid-19 daily death counts. Your test accuracy will be measured (and compared) in terms of means squared error against the actual deaths that happen on those days.[1]

You are free to construct any feature set you wish and train as many linear regression models as you want in order to solve this task. Here and below we provide some useful guidance about how to use linear regression models to learn one-step-ahead prediction models that "eat their own predictions."

Linear Autoregressive Models

To start, consider a one-dimensional time series dataset D consisting of values $[d_1, d_2, d_3, \ldots, d_t, \ldots, d_T]$. A linear regression model for one-step-ahead prediction can be learned by constructing

$$y = \begin{bmatrix} d_K \\ d_{K+1} \\ \vdots \\ d_T \end{bmatrix} \qquad X = \begin{bmatrix} x_K^T \\ x_{K+1}^T \\ \vdots \\ x_T^T \end{bmatrix} = \begin{bmatrix} 1 & d_1 & d_2 & \cdots & d_{K-1} \\ 1 & d_2 & d_3 & \cdots & d_K \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & d_{T-K} & d_{T-K+1} & \cdots & d_{T-1} \end{bmatrix}.$$

Here we have implicitly defined feature vectors x_i which correspond to the vector of K previous values in D used to predict y_i. If you learn a regression model with parameter vector w with this data you can use it to

[1] yes, this means that this question and your performance won't be known until several days after your midterm is handed in and your *static* predictions have already been made

predict $\hat{y}_{T+1}$ given $\tilde{x}_{T+1} = [1,\ d_{T-K+1},\ d_{T-K+2},\ \ldots,\ d_T]^T$. This is what is known as an "autoregressive" model. What is more, such a model can be used to make multi-step predictions into the future by continuing the following recursion

$$\hat{y}_{T+2} \approx [1,\ d_{T-K+2},\ d_{T-K+3},\ \ldots,\ d_T,\ \hat{y}_{T+1}]w$$

where the latest prediction is fed into the feature vector in the appropriate position.

Prediction Task

In this competition you only need to make predictions about Canadian daily death counts. However, the performance of your submission will largely be dictated by how you make use of the other features in the data, particularly the counts in other countries at the same time. The data you receive we call D. It consists of a number of $\{\text{country\_id}, \text{date}, \text{cases}, \text{deaths}, \text{cases\_14\_100k}, \text{cases\_100k}\}$ tuples. An exact explanation of these features is available online.[2]

The simplest thing you could try is the regression model just described, trained and operating only on the Canadian daily death counts. Of course it is also clear that you could build independent country- and "feature"-specific regressors (by feature here we mean colloquially the cases, deaths, cases_14_100k, cases_100k values). This, unfortunately, would not leverage count values from other countries in making the country and feature specific predictions.

This challenge problem gets much more interesting when you think about ways to leverage all the additional data available from countries around the world, particularly by designing feature matrices for each regressor that leverages data from other countries, either in aggregate or in specific.

Here is one kind of thing that you can do. Call the output that we wish to predict for a particular feature (take cases for example), day (take 8/21/20 for example), and country (take AD for example) combination y_i. The feature vector you construct for this particular instance can carry information from all countries on any days prior to 8/21/20. These can be aggregated features, can be restricted to neighboring countries, or countries with high flight connectivity, etc.

Moreover, depending on how you construct your feature space, you can build models that are not specific to a particular country. So long as the first, say, L entries of your feature vector are extracted from the data from the country of the required prediction on any days prior and the remaining $d - L$ entries are features extracted from the data from all countries on any days prior the resulting regression model will be country-agnostic.

See Fig 1 for helpful hints about how one might construct such complex and informative feature spaces.

Submitting Your Results

You need to be prepared to do several things. First, as usual, you must bundle your code along with a `.pdf` generated from the filled in LaTeXreport skeleton into a `.zip` file and submit it to Gradescope. Again, 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.

Also, Phase 2 of the Kaggle competition will be released the day before the midterm is due. You may retrain your model on the additional data or you may simply use it as input to predict five days into the future past the midterm due date. You will upload these predictions to the Kaggle server and your model's relative test performance will be made available in 5 days time.

[2] `https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide`

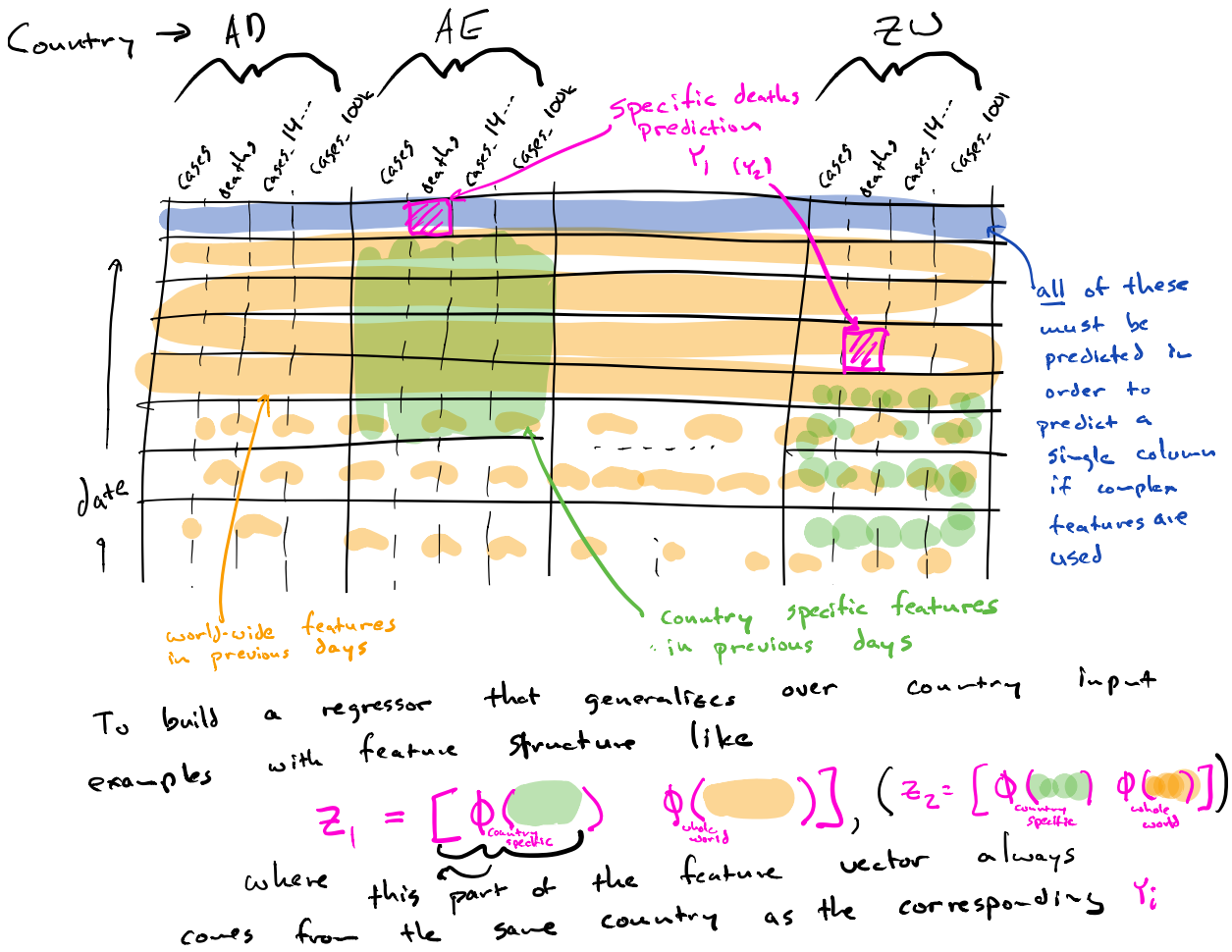

Figure 1: Feature Matrix Hints Diagram

Template for Question 1

1 Individual Classifiers

1.1 Result

Report the train and test result for each classifier in the given table. You should use the following hyperparameters,

1. Random Forest: no max cap on depth, and a forest size of 15 trees.
2. KNN: $k = 3$.
3. Continuous Naive Bayes: has no hyperparameters.

Model	Your Train Error (%)	Your Test Error (%)
Random Forest KNN Naive Bayes		

Explain in one paragraph why you think a particular classifier works better on this dataset.

1.2 Code

Include the code you have written for each particular classifier.

1. Random Forest

2. KNN

3. Naive Bayes

2 Stacking

2.1 Result

Report the test error and training error of the stacking classifier.

2.2 Code

Include all the code you have written for stacking classifier

Template for Question 2

1 Team

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

2 Solution Summary

In no more than several paragraphs summarize the approach you took to address the problem.

3 Experiments

In this section report, in less than two pages, describe in technical terms the training procedures you used, including how you went about feature selection, hyperparameter value selection, training, and so forth. Plots related to hyperparameter sweeps and other reportable aspects of your training procedure would be appreciated.

4 Results

Team Name	Kaggle Phase 1 Score	Kaggle Phase 2 Score
the name of your team	*your Phase 1 Kaggle score*	*your Phase 2 Kaggle score*

5 Conclusion

Describe what you learned and what you would have done were you to have been given more time in a few paragraphs.

Appendix

Gini Index

In this example, we have a dataset with two features x and y. Each data entry belongs to either the blue class or green class.

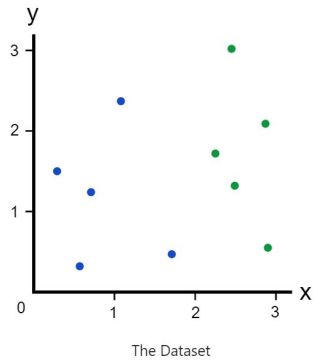

Figure 2: A given dataset with two features

lets define
p_b = probability of blue class,
p_g = probability of green class.

To compute the Gini impurity before splitting,
$G(p_g^{NoSplit}, p_b^{NoSplit}) = p_g^{NoSplit}(1 - p_g^{NoSplit}) + p_b^{NoSplit}(1 - p_b^{NoSplit}) = \frac{5}{10}(1 - \frac{5}{10})$,
$G(p_g^{NoSplit}, p_b^{NoSplit}) = \frac{1}{2}$.

In order to find the best split in the x-axis, we should search over the set of possible splits. One arbitrary choice is shown in figure 2.

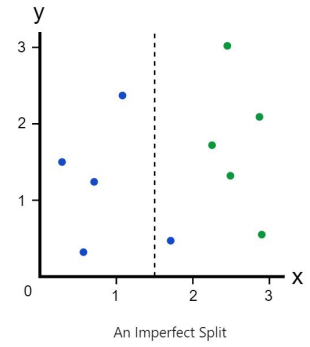

Figure 3: split feature x, where x = 1.5.

let's compute the Gini impurity for the right side.

$G(p_g^r, p_b^r) = p_g^r(1 - p_g^r) + p_b^r(1 - p_b^r)$, where $p_g^r = \frac{5}{6}$, $p_b^r = \frac{1}{6}$,

$G(p_g^r, p_b^r) = \frac{10}{36}$

And for the left side,

$G(p_g^l, p_b^l) = p_g^l(1 - p_g^l) + p_b^l(1 - p_b^l)$, where $p_g^l = \frac{0}{4}$, $p_b^l = \frac{4}{4}$.
$G(p_g^l, p_b^l) = 0$.

In the next step, we compute the Gini index for the current split, as follosw

$Gini\ Index = \frac{N_l}{N_t} * G(p_g^l, p_b^l) + \frac{N_r}{N_t} * G(p_g^r, p_b^r) = \frac{4}{10} * 0 + \frac{6}{10} * \frac{10}{36} = \frac{1}{6}$.

Another possible split is demonstrated in the figure 3.

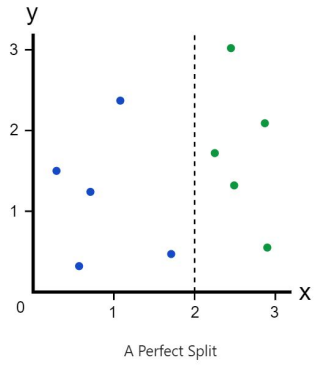

Figure 4: split feature x, where x = 2.

To compute the Gini impurity for the right side, we have

$G(p_g^r, p_b^r) = p_g^r(1 - p_g^r) + p_b^r(1 - p_b^r)$, where $p_g^r = \frac{5}{5}$, $p_b^r = \frac{0}{5}$
$G(p_g^r, p_b^r) = 0$.

And the Gini impurity for the left side is

$G(p_g^l, p_b^l) = p_g^l(1 - p_g^l) + p_b^l(1 - p_b^l)$, where $p_g^l = \frac{0}{5}$, $p_b^l = \frac{5}{5}$
$G(p_g^l, p_b^l) = 0$.

In the next step, we compute the Gini index for the current split:

$Gini\ Index = \frac{N_l}{N_t} * G(p_g^l, p_b^l) + \frac{N_r}{N_t} * G(p_g^r, p_b^r) = \frac{5}{10} * 0 + \frac{5}{10} * 0 = 0$.

In the end, we find the minimum Gini index between the splits. The minimum Gini index for this example is when x = 2. Also, the minimum Gini index is less than Gini impurity of no split. Therefore, we select x = 2 as the splitting rule.