# Introduction to Machine Learning (COMP 170)
## Project 02 (95 points)
***Due on Gradescope by 5:00 PM, Monday, 27 April 2020***

---

For this assignment, you will explore a variety of classifiers, applied to some real-world data-sets for natural language processing. In each case, you will explore and analyze a range of model hyperparameters. You will hand in a PDF containing results and analysis, along with the usual collaborators file; you ***do not*** hand in code for this assignment.

***Note***: for full points, the PDF should consist mostly of results, figures, and discussion; code snippets are fine, but large blocks of code are not desirable, as they will be harder to understand than careful plain language explanations. If you want to include a PDF generated from a Python notebook as a separate appendix, you can, but a "report" that consists only of that is incomplete.

You will also submit two separate text-files containing predictions for classifiers that you will build; each will be scored as before on a leaderboard, based upon its accuracy on a provided testing input set (for which you do not know the appropriate outputs).

### Data for the Project

You have been supplied with several thousand single-sentence reviews, collected from three domains: `imdb.com`, `amazon.com`, and `yelp.com`. Each review consists of a sentence, and has been assigned a binary label indicating the ***sentiment*** (`1` for positive and `0` for negative) of that sentence. Your goal is to develop binary classifiers that can generate the sentiment-labels for new sentences, automating the assessment process. While the reviews were collected from websites where much of the content is in English, the reviews may well contain slang, spelling errors, foreign characters and the like, all of which make natural language data challenging, albeit fun, to try to classify like this.

The provided data consists of $2,400$ training examples in the usual CSV `x` and `y` format.* Input data has two columns, for the source-website and review text; outputs are given as binary values, where `1` indicates a positive review. There are also 600 testing inputs, for which no `y`-values are given; these will be used for validation against the Gradescope leaderboards. The Project download also contains a short script, `load_train_data.py`, that will give you guidance as to how you might load the data using Pandas (of course, you can load it in other ways as well if you so choose).

Examples of ***positive*** reviews include:

- `(amazon) #1 It Works - #2 It is Comfortable.`
- `(imdb) "Gotta love those close-ups of slimy, drooling teeth!  "`
- `(yelp) Food was so goooddd.`

Examples of ***negative*** reviews include:

- `(amazon) DO NOT BUY DO NOT BUYIT SUCKS`
- `(imdb) This is not movie-making.`
- `(yelp) The service was poor and thats being nice.`

*Data comes from work by D. Kotzias, M. Denil, N. De Freitas, and P. Smyth, as described in their paper From Group to Individualized Labels Using Deep Features (KDD 2015). Thanks to the authors for making it available.

It is recommended that you ***preprocess*** your data, removing punctuation, non-English and non-text characters, and unifying the case (i.e., setting everything to be either upper- or lower-case). You will then investigate two types of ***feature representations*** for converting strings of words in sentence form into feature vectors $\mathbf{x}_n$ of some common length $n$. For each of the two feature representations, you will build and compare a number of different types of models.

**Part Zero: Collaborators file** (5 points)

Provide the usual file containing your name, the amount of time you worked on the assignment, and any resources or individuals you consulted in your work. Note that for this assignment, there are ***no restrictions*** on the Python libraries you may use (you won't be handing in code), along with the usual `sklearn` models and functions; you can consult other sites and guides to those libraries, but should include any such resources in your collaborators file as well.

**Part One: Classifying Review Sentiment with Bag-of-Words Features** (42 points)

As briefly discussed in Lecture 09, when we were talking about nearest-neighbor models, the "Bag-of-Words" (BoW) model of a document (i.e., in this case, a single review) involves determining a known fixed vocabulary, $V$, in advance, imposing an order on those words, and then representing each document with a vector of length $|V|$ that has a non-zero value at position $i$ if the $i$th word in $V$ is part of that document, and is `0` otherwise. You will build such a representation for your input data (train and test). Your first step will be to make some design decisions with respect to how your BoW model works; questions you will need to answer may include:

- How big is the vocabulary, and what order to you place those words into?
- Do you exclude very rare words (and what does "very rare" mean)?
- Do you exclude very common words (and what does "very common" mean)?
- Do you count the occurrences of a word in the document, or only record if it is there or not (producing a binary vector)?
- Is it worth using something other than word counts, like the *inverse document frequency* idea described in lecture.
- Do you use single features only, or do you try counting word-pairs instead? What about counting $n$-tuples of words?

Whatever you decide (and you may want to experiment) you want a representation whereby each feature of the resulting input vector corresponds to a single word (or $n$-tuples of words, if you go that route). Once you have decided upon your feature representation, you will investigate three distinct classifier models on the data, seeking one that gives you best performance.

**Resources**: there are several tools available in `sklearn` for creating BoW representations:

https://scikit-learn.org/stable/modules/feature_extraction.html

1. (*10 pts.*) In your report, include a paragraph that explains the "pipeline" for generating your BoW features. This should include a clear description of any pre-processing you did on the basic text, along with the sorts of decisions you made in generating your final feature-vectors. You should present this in complete enough form that someone else (another student, say) could produce a model identical to yours if they wished, based upon reading your report. As

we have said before, keep code samples to a minimum; ideally, you should be able to explain what you did in plain language. Your paragraph should also contain some justification for why you made the decisions you did.

2. (*7 pts.*) Generate a ***logistic regression*** model for your feature-data and use it to classify the training data. In your report:

   - Give a few sentences describing the model you built, and any decision made about how you set is parameters, trained it, etc.
   - Choose at least one hyperparameter that controls model complexity and/or its tendency to overfit. Vary that hyperparameter in a systematic way, testing it using a cross-validation methodology. Explain the hyperparameter(s) you chose, the range of values you explored (and why), and describe the cross-validation testing in a clear enough manner that the reader could reproduce its basic form, if desired.
   - Produce at least one figure that shows, for at least one tested hyperparameter, at performance on at least 5 distinct values—this performance should be plotted in terms of average error for both training and validation data across the multiple folds, for each of the values of the hyperparameter. Include information, either in the figure, or along with it in the report, on the *uncertainty* in these results.[†]
   - Give a few sentences analyzing these results. Are there hyperparameter settings for which the classifier clearly does better (or worse)? Is there evidence of over-fitting at some settings?

3. (*7 pts.*) Generate a ***neural network*** (or MLP) model for you feature-data. Produce the same sort of description and analysis for it as you did for the previous model, including variation of one or more hyperparameters, cross-validation testing, and at least one figure that shows how performance on training and validation data is affected as the hyperparameters change.

4. (*7 pts.*) Generate a third model, of whatever type you choose; you could use, for instance, SVM classifiers, or try ones that we have not yet explored directly (`sklearn` has its own decision-tree and decision-forest classifiers, for example). Whatever you choose, produce the same analysis as for the prior models, including a description of what you did, how hyperparameter variation affected results, and so forth. A figure is expected showing training/validation performance relative to hyperparameter variation; additional figures are allowed, of course.

5. (*7 pts.*) Summarize which classifier of the three you built performs best overall on your labeled data, and give some reasons why this may be so. Does it have more flexibility? Is it better at avoiding overfitting on this data?

   In addition, look at the performance of your best classifier and try to characterize the mistakes that it makes. Are there common features to the sentences that it gets wrong (e.g., are they mostly from one of the three source websites)? Are there other features that you can identify? Can you hypothesize why you see the results you do?

[†]This can be measured in terms of simply standard deviation across the $k$-fold cross-validation tests, or in more detail by showing exact performance metrics on each fold. The idea is to help the reader understand if the average performance is typical and stable, or if there is a lot of difference from one cross-validation test to another.

6. (*4 pts.*) Apply your best classifier from the previous steps to the text data in `x_test.csv` file, storing the outcomes as a probabilistic prediction and then submitting them to the Bag-of-Words leaderboard, as described below. In your report, describe the performance that you see there. How does that match up to the performance you saw during training and cross-validation? If it is as expected, what does that tell us, do you think? If it is not as expected, what does *that* tell us?

**Part Two: Classifying Review Sentiment with Word Embeddings** (42 points)

The basic idea of a ***word embedding*** is that for each possible vocabulary word, we have a complex feature-vector for that specific word (generated by some other process), something typically much more complex than a single number giving occurrence counts or other frequency measures. You can then take a document, and then combine the feature-vectors of each word in it in some way, to generate the final features that are used an inputs to a classifier.

Some background on word embeddings can be found in the following linked articles:

- Shane Lynn: Get Busy with Word Embeddings
- Jason Brownlee: What Are Word Embeddings for Text?

In the download for the project, we have provided an archived file containing pre-trained embedding vectors for 400,000 possible vocabulary words. Each line of that file consists of a word, followed by a 50-value embedding vector for it. There is also a Python code example that shows one way you might load the data.[‡]

The vectors you have been given have been generated by the GloVe (Global Vectors) technique; you can read more about it, and find some more pre-trained vectors in:

- The The GloVe project site at Stanford
- Pennington, Socher & Manning, GloVe research paper, *EMNLP-14*

1. (*10 pts.*) Given a set of pre-trained embedding vectors (from GloVe or not), we can represent one of our reviews by taking all of the vectors for each of the words that occur in it, and combine them together to produce the final input vector for our classifier. Decisions you need to make now are how you will do this process of combining vectors. Things you may consider, among others:

   - How do you aggregate the vectors: do you average them, sum them, concatenate them, or do something else entirely?
   - What do you do when a word appears many times: do you use multiple copies of its vector, or just use it once?
   - Do you ignore rare or common words?

   In your report, explain your feature-generation pipeline, as you did for the BoW model. Again, you should include a paragraph that clearly describes what you did, sufficient that a reader could duplicate your process, and giving some justification for the choices you made.

[‡]Thanks to Dr. Mike Hughes for that sample code.

2. (*7 pts.*) As you did for BoW, train a logistic regression model, analyze its performance when you very one or more hyperparameters, and give at least one figure with related analysis of cross-validated performance. The length and substance will be similar to what you did for the BoW regression model.

3. (*7 pts.*) As you did for BoW, train a neural network model, analyze its performance when you very one or more hyperparameters, and give at least one figure with related analysis of cross-validated performance. The length and substance will be similar to what you did for the BoW neural network model.

4. (*7 pts.*) As you did for BoW, train a third model of your choice, analyze its performance when you very one or more hyperparameters, and give at least one figure with related analysis of cross-validated performance. The length and substance will be similar to what you did for the corresponding BoW model.

   ***Note***: if you want, you can choose a different model for this part of the project than you did for the first part; as long as it is not just another logistic regression or neural network model, anything is fair game.

5. (*7 pts.*) As for the BoW models, summarize and analyze the best model you found when using embedding vectors. Analyze the sorts of mistakes it makes. This should be comparable in length and substance to the corresponding section of the BoW portion.

6. (*4 pts.*) Apply your best model from the previous three steps to the unlabeled test data and submit the predictions to the GloVe leaderboard. Report performance and analyze how it looks relative to what you saw during the cross-validation process.

**Part Three: Prediction submissions** (6 points)

To test your various classifiers, you can submit the predictions that each makes—on the unlabeled `x_test.csv` file—to a leaderboard (one for Bag-of-Words features, and one for word embeddings). You can submit multiple classifiers, of multiple types, and simply re-submit whichever did the best at the end for your final graded score. The leaderboard code will compare your predictions to known correct examples, scoring them relative to the correct answers. Each of the leaderboard scores will count for 3 points of the overall project score.

As for Project 01, the submission should be in the form of a plain text-file, named `yproba1_test.txt`, containing one probability value (a floating-point number giving the probability of a positive binary label, `1`) per example in the test input. Each line will be a single number, and we should be able to load it into a 1-dimensional NumPy array using:

```
np.loadtxt('yproba1_test.txt')
```

(It would be a good idea to verify that this will work as expected.) These numbers will be thresholded at a probability of 0.5 for scoring purposes.

### Submission details

There are ***four*** Gradescope submission links for this assignment. You will upload the following:

- **Collaborators information**: Submit the usual `COLLABORATORS.txt` file, containing your name, amount of time spent on the project, and persons/resources consulted.
- **PDF**: Submit the PDF containing all the required figures and discussion.
- **Predictions**: Submit the text-files containing your two sets of predictions on the test data to the relevant leaderboards.