**Warmup**

(CAN YOU GIVE ME this one first once it’s done? I need this warmup asap! Rest project can take yout time) will be using the Keras and TensorFlow systems to do some exploratory work with convolutional neural networks.\_

1, add the TensorFlow framework for neural networks, and the Keras front-end, which provides ease of use, so use the following steps for installation: 1; pip install --upgrade pip; 2, install tensorflow library: pip install tensorflow; 3, install Keras frontend pip install Keras.

Data used: CIFAR-10 object recognition

The CIFAR-10 dataset consists of 60000 images, equally divided among 10 classes, including

cats, dogs, frogs, airplanes, and automobiles. It is a popular dataset for use in image classification, and can be autoloaded using Keras itself.

There are a number of online tutorials to get started with CIFAR-10. Start by

working through all of the steps in the following http://tinyurl.com/cifar10-exercise, for both the basic model and the more complex one.

Add Markdown cells between the various major pieces of code. Each Markdown cell need you to contain explanatory text, indicating what each piece of the code is doing and why. Must have high quality of explanations

Note: Training the various models can be a time consuming task, so if you have some machines that have fast speed, that’s great. Note also that while testing, you may want to limit the number and size of the training epochs, to speed things up.

The tutorial, as given, works through building a classifier that gets approximately 80% accuracy. At the end of the online exercise are some suggestions on how these results might be improved. Work through some of these suggestions, or try other techniques of your own, and see if you can improve the classifier performance. In notebook Markdown sections, document details you have tried, and the results (good or bad). Must be very clear in explanation and show detailed of evidence that is interesting or creative.

Should contain at least one figure analyzing model performance, either during training or overall. One example could be classifier accuracy over the training period, analysis of particular errors, etc.

**Main project: We did this many months ago. This time is kind of redoing it because I am not satisfied with results. So, you need to use previous code and do some editing to make it perfect. Don’t redo all. Just use old written code and do editing stuff and make all parts 100% correct. The pages seem long, but most are my explanation to make things clear. I know I am wordy. In the separate file called “need to specify in details” I list specific part I was not satisfied, which needs focus on and needs clear explanations on them.**

Natural language processing we did last time together. You will explore and analyze a range of model

hyperparameters.

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 classifier 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 2400 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 on leaderboard list we did before. Folder also includes 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 gooodd.

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.

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 xn of some common length n. For each of the two feature

representations, you will build and compare a number of different types of models.

**P1: Classifying Review Sentiment with Bag-of-Words Features**

First, 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 ith 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. 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 could produce a model identical to yours if they wished. You should be able to explain what you did to me clearly. Your justification for why you made the decisions you did needs to explain to me.
2. Generate a logistic regression model for your feature-data and use it to classify the training data. Write details:
   * 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. (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. )

* + 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?

1. 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.
2. Generate a third model, of whatever type you choose; you could use, for instance, SVM classifiers, or try ones whatever you want: (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 al- lowed, of course.
3. Summarize which classifier of the three you built performs best overall on your labeled data, and give 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?

1. Apply your best classifier from the previous steps to the text data in x\_test.csv file, storing the outcomes as a probabilistic prediction as before. Write to describe the performance that you see there. How does that match up to the performance you saw during training and crossvalidation? If it is as expected, what does that mean? If it is not as expected, what does that mean?

**P2: Classifying Review Sentiment with Word Embeddings**

The basic idea of a word embedding is that for each possible vocabulary word, you area given 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 input to a classifier.

Some background on word embeddings can be found in the following linked articles: • Shane Lynn: Get Busy with Word Embeddings: <https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/>

• Jason Brownlee: What Are Word Embeddings for Text? <https://machinelearningmastery.com/what-are-word-embeddings/>

Folder has archived files 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: <https://nlp.stanford.edu/projects/glove/> at Stanford

* Pennington, Socher & Manning, GloVe research paper EMNLP-14 <https://www.aclweb.org/anthology/D14-1162/>

1. 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?
* Write your feature-generation pipeline, as you did for the BoW model. Again, you should clearly describe what you did, sufficient that a reader could duplicate your process, and giving some justification for the choices you made.

1. 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.
2. 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.
3. 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

1. 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.
2. 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.
3. Generate the predictions that each makes as before on the unlabeled x\_test.csv file, meaning 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 you should be able to load it into a 1-dimensional NumPy array using:

np.loadtxt('yproba1\_test.txt')