MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2015

Project 1: Automatic Answer Grading. Issued: Fri., 2/20 Due: Fri., 3/6 at 9am

Project Submission: Please submit a .zip file of your project on the Stellar web site by 9am, 3/6. The zip file should contain: a PDF of your project report, your python code, and file with the labels for the answer test set in OUR FORMAT.

Introduction

The goal of this project is to design a classifier that automatically evaluates student-written answers to exam-type questions. Our training set consists of short answers written by high-school students to specific question prompts on different topics, and their evaluations produced by teachers. For the purpose of the project, we weed to discriminate between good and bad answers. Table 1 shows an example of two answers from our dataset and their corresponding labels.

Student's Answer	Label
When translation begins, the mRNA goes with a ribosome and attaches itself to a start codon. It then goes with a ribosome to start adding the anticodons to the codons added on to the genetic code; the anticodons are added by the tRNA. The tRNA attaches to the P site first then moves to the A site when the it attaches an amino acid to the amino acid chain, then this process continues until a stop codon is reached. The result is the newly created protein.	1
The mRNA goes to a ribosome. Then tRNA come and attach themselves to the mRNA. The DNA is replicated. Then they unzip and there are two DNA strands.	-1

Table 1: Example entries from the short-answer dataset. The two answers were written by two different students answering the following prompt: "*Starting with mRNA leaving the nucleus, list and describe four major steps involved in protein synthesis.*"

In order to create an automatic answer grader, you will tackle the following tasks:

- Implement and compare two types of linear classifiers: the perceptron and passive-aggressive algorithm (Section 1).
- Use your classifiers on the short answer dataset, using some simple text features (Section 2).
- Experiment with additional features and explore their impact on the performance of your classifier (Section 3).

In addition, as part of the project we will run a competition on classifying a set of examples for which you are not provided with labels. You will receive extra credit if your method performs particularly well in this dataset (see below for details).

0. Setup

For this project and throughout the course we will be using Python with some additional libraries. We strongly recommend that you take note of how the NumPy numerical library is used in the code provided, and read through the on-line NumPy tutorial. NumPy arrays are much more efficient than Python's native arrays when doing numerical computation. In addition, using NumPy will substantially reduce the lines of code you will need to write. Please also make use of `Matplotlib` when asked to produce figures and plots.

0. Download the necessary libraries and tools.
   Note on software: For the all the 6.036 projects, we will use python augmented with the **NumPy** *numerical toolbox, the* **matplotlib** *plotting toolbox, and the* **scikit_learn** *machine learning toolbox (though you are not permitted to use* **scikit_learn** *until project two). We strongly recommend (and do our testing with) the Enthought Canopy Python distribution, as it includes the* **IPython** *environment, the matplotlib and NumPy toolboxes, and a package manager that can be used to add the scikit_learn toolbox. Enthought canopy is available free for students and staff (but you have to use your* `.edu` *email address), and installers for all major operating systems can be found at:*
   `https://www.enthought.com/products/canopy/`.

1. Download `project1.zip` from Stellar and unzip it in to a working directory. The zip file contains the various data files in `.tsv` format, along with a two python files:. `project1_code.py` contains various useful functions and it has function templates for you to and you will implement your learning algorithms. `main.py` is a script skeleton from where you will call these functions and run your experiments. You will use `best_learner.py` to label the data for the challenge.

1. Implementing classifiers

We will implement three linear classifiers: perceptron, average perceptron, and the average passive-aggressive (PA) algorithm. Since the data may not be linearly separable, we we will make two modifications to these algorithms. First, all the algorithms will run only a fixed number T of iterations through the training set. As a result, the algorithm will update the parameters at most nT times. The average perceptron and average passive-aggressive algorithms will add another modification. Since the basic algorithms continue updating, nudging parameters in possibly conflicting directions, it is better to take an average of those parameters as the final answer. For example, for perceptron, every step of the algorithm is the same, updating the parameters θ as before. However, the final θ parameters that the algorithm returns are an average of the θs across the nT steps:

$$\theta_{\text{final}} = \frac{1}{nT}\left(\theta^{(1)} + \theta^{(2)} + \cdots + \theta^{(nT)}\right)$$

The same idea defines the average passive-aggressive algorithm.

2. Implement the perceptron, average perceptron and average passive-aggressive algorithms in the corresponding functions in the project1_code.py script.

3. We have included a synthetic 2D dataset in `toy_data.txt` which you should use to qualitatively verify your implementations. Use `read_toy_data` to import the data, train your models using $T = 5, \lambda = 10$ and initializing the weight vector an offset with $\theta = 0, \theta_0 = 0$. Plot the resulting model or boundary for each function using `plot_2d_examples`. Which of the three algorithms

performed the best and which performed the worst? Why is this? Hand in: the plots obtained for each algorithm, and a paragraph answering the question.

2. The automatic grading task

Now that you have verified the correctness of your implementations, you are ready to tackle the main task of this project: building a classifier that labels answers as "good" or "bad", using text-based features and the linear classifiers that you implemented in the previous section.

The Data

The data consists of several answers, each of which has been labeled with -1 or 1, corresponding to a "bad" or "good" answer, respectively. The approximate class proportion is 75% negative vs 25% positive examples. The original data was split into four files: `train.tsv` (1800 examples), `tion.tsv` (350 examples), `test.tsv` (350 examples) and `submit.tsv` (500 examples). Only the first three sets are labeled, and you will use them to train, tune and evaluate your classifier. Once you have optimized your method, you will use it to predict labels for the `submit` data, which you will submit along with your code and answers.

All the data files have the same format: they are tab separated and contain a label in the first column and the raw text of the answer on the second.

Translating answers to feature vectors

We will convert answers into feature vectors using a *bag of words* approach. We start by compiling all the words that appear in a training set of answers into a *dictionary*, thereby producing a list of d unique words. We can then transform each of the answers into a feature vector of length d by setting the i^{th} coordinate of the feature vector to 1 if the i^{th} word appears in the answer and setting the coordinate to zero otherwise. For instance, consider two toy documents "Mary loves apples" and "Red apples". In this case, the dictionary is a list $(Mary, loves, apples, red)$, and the documents are represented as $(1, 1, 1, 0)$ and $(0, 0, 1, 1)$. A bag of words model can be easily expanded to include phrases of length m. In the project, we will be using unigrams ($m = 1$) and bigrams ($m = 2$).

For this section, your tasks are the following:

6. Import the four data sets with the `read_data` function. The function `extract_feature_vectors` takes the raw text as an input and computes some basic features: unigram and bigram occurrence, along with some statistics about text and word length.

7. Train your classifiers of the feature vectors obtained from the `train` data using $T = 5$ and $\lambda = 1$, and evaluate their performance on the `test` set. To do this, you will need to implement the function `classify` to assign labels to new data, after which you can assess the accuracy of your predictions using `score_accuracy`. Report the training and testing accuracy that you obtain with each of the three methods. These should be in the $83 - 90\%$ range.

8. To improve the performance of your methods, you will need to *tune* (e.g. optimize) the parameters T and λ. For this purpose, you will train your train your classifiers with several values of λ and T, each time using the `validation` data to assess the performance. Reasonable values to try are $\lambda = [1, 10, 20, 50, 100]$ and $T = [1, 5, 10, 15, 20]$. For each of these values, compute the training and validation accuracy and plot your results with the function `plot_scores`. Hand in: your plot and your solutions to the following questions:

(a) Do the training and validation accuracies behave similarly as a function of λ and T? Why or why not?

(b) What are the optimal values of T and λ?

9. After you have chosen your best method (perceptron or passive-agressive) and parameters, use this classifier to compute testing accuracy on the `test` set. Report this value.

3. New Features and the challenge

Frequently, they way the data is represented can have a significant impact on the performance of a machine learning method. Try to improve the performance of your passive-aggressive classifier by augmenting the feature set. Some features that you might want to explore are:

- Number of words of length greater that k
- Occurrence of misspelled words
- Occurrence of "difficult" words
- Number of stop words (e.g. functional words such as "the", "to", "for")

The file `SAT_words.txt` contains a list of $\sim$ 5000 words commonly found in the SAT exams, which you can use as prototypical "difficult" words. The file `stopwords.txt` contains a list of stop words. To verify spelling, we are providing you with a file `words.txt` which contains a list of approximately 200K words.

10. Experiment with at least one of these additional features and compare the performance of your method to your results with the original features. Again, you should use the validation set to choose values for the parameters and report the error on the test set. Your report should contain:

    - A clear explanation of how you changed the feature set, and why you think the features you chose might be useful. Include the code you use for this purpose in `extract_feature_vectors`.
    - A description of the experiment you conducted to compare it to the original feature-computation method, and the results of that experiment. Include plots if applicable.

    It's okay if it turns out that your new method performs worse than the original one—just explain and document your results.

11. Modify `best_learner.py` to call your best performing method. For example, if you obtained the best results with the PA method using values λ^* and T^*, you need to replace line 13 with a call to `p1.avg_passive_aggressive` using λ^* and T^*. We will run this script to verify your method, so make sure it works as expected. In addition, `best_learner.py` labels the answers in `submit.tsv` and writes them to `challenge_predictions.txt`. **Remember to include this file in your .zip**.

12. **(Extra Credit)** You will receive 3 points extra credit if the accuracy of your submitted labels is above the staff baseline (the staff is busy and probably won't spend as much time as you optimizing parameters and trying new features, but will use the best of the suggestions listed above). You will receive 10 points extra credit and looks of jealousy from all of your friends if your predictions are the best in the class. The top performing scores will be posted on Stellar.

REMEMER Submit a .zip file containing your source code, the label vector for the answer test set, and your report in PDF format to Stellar.