CIS519: Applied Machine Learning	**Spring 2018**
Homework 2	
Handed Out: Februray 14^{th}*,* 2018	*Due: February* 26^{th}*,* 2018*, 11:59 PM*

- Feel free to talk to other members of the class in doing the homework. We are more concerned that you learn how to solve the problem than that you demonstrate that you solved it entirely on your own. You should, however, write down your solution yourself. Please record the names of the people you consulted with in the course of working on this assignment at the top of your solution. Please try to keep the solution brief and clear.
- Please use Piazza if you have questions about the homework. Also, come to the recitations and the office hours.
- Please, no handwritten solutions. You will submit your solution report as a single pdf file and the code package as detailed below. Follow the instructions in the "What to submit" section for more details.
- While we encourage discussion within and outside the class, cheating and copying code is strictly discouraged. Copied code will result in the entire assignment being discarded at the very least.
- The homework is due at 11:59 PM on the due date. We will be using Canvas for collecting the homework assignments. Please submit your solution manuscript as a pdf file via Canvas Please do NOT hand in a hard copy of your write-up. Contact the TAs if you are having technical difficulties in submitting the assignment.
- **Comment:** You are highly encouraged to start on this early because the experiments and graphs may take some time to generate.

1 Algorithms

In this problem set, you will implement and experiment with several linear learning algorithms: Peceptron, Winnow and Adagrad-Perceptron, along with their averaged versions and SVM. You will implement the first three update rules, and their averaged version (making it six algorithms) and will use an existing implementation for SVM. You will then experiment with these algorithms by comparing their performance on synthetic and real-world datasets.

The goal is to understand the differences and similarities between the algorithms, and the impact of the data characteristics on the algorithms' learning behavior and performance.

All three key algorithms are slight variations of each other. You will implement them based on the code we provide for the basic version of Perceptron. You will then code the Averaged scheme as suggested below and use it on top of all three variations. For SVM you will use existing library functions.

You will evaluate your algorithms on a synthetic dataset and observe the difference in behavior of learning algorithms when the target function is sparse or dense. Finally, you will use these algorithms to solve a (simplified version of a) real-world problem, named entity recognition (NER). For the NER you will also implement the feature extraction part, based on a scheme for extracting contextual features from the raw text that is given to you. Your model will be trained on a news dataset that we will give, and tested on both news and email datasets, which will illustrate the difficulty of adapting to new datasets.

Next we provide the details of the learning algorithms to be used in this problem set.

1.1 Variations of Perceptron

Perceptron is an online and mistake driven algorithm. We will consider different variations of the Perceptron – basic Perceptron (code provided), Winnow, and Adagrad. In addition, we will consider the averaged version of these. Since the code for the basic Perceptron is provided, you only need to slightly change the update rule for the two other versions.

a. **Basic Perceptron:** This the basic version of the Perceptron Algorithm. In this version, an update will be performed on the example (x, y) if $y(w^\intercal x + \theta) \leq 0$.

   There are two things about the Perceptron algorithm that should be noted.

   First, the Perceptron algorithm needs to learn both the bias term θ and the weight vector w. When the Perceptron algorithm makes a mistake on the example (x, y), both w and θ has to be updated as follows:

$$\begin{aligned} w_{\text{new}} &\leftarrow w + \eta y x \\ \text{and} \quad \theta_{\text{new}} &\leftarrow \theta + \eta y \end{aligned}$$

   where η is the learning rate. See the lecture notes for more information.

   Second (and more surprising), if we assume that the order of the examples presented to the algorithm is fixed, and we initialize $\begin{bmatrix} w & \theta \end{bmatrix}$ with a zero vector and learn w and θ together, then the learning rate η, in fact, does not have any effect[1].

   Parameters: Given the second fact above, we can fix $\eta = 1$. So there are no parameters to tune.

   Initialization: $w^\intercal = \{0, 0, \cdots, 0\}, \theta = 0$

b. **Winnow**: We describe the basic version of Winnow. Notice that all the target functions we deal with (in the synthetic data part of the assignment) are monotone functions, so we can use this basic version of Winnow.

   When the Winnow algorithm makes a mistake on the example (x, y), w will be updated in the following way:

$$w_{t+1,i} \leftarrow w_{t,i} \alpha^{y x_i}$$

   where α is promotion/demotion parameter and $w_{t,i}$ is the ith component of the weight vector after t mistakes.

   Parameters: Promotion/demotion parameter α (α should be greater than 1)

   Initialization: $w^\intercal = \{1, 1, \cdots, 1\}, \theta = -n$ (n is the number of features) (θ is fixed here, we **do not** update it)

   Parameter Recommendation: Choose $\alpha \in \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$.

[1] In fact you can show that, if w_1 and θ_1 is the output of the Perceptron algorithm with learning rate η_1, then w_1/η_1 and θ_1/η_1 will be the result of the Perceptron with learning rate 1 (note that these two hyperplanes give identical predictions).

c. **Perceptron with AdaGrad**: AdaGrad adapts the learning rate based on historical information, so that frequently changing features get smaller learning rates and stable features get higher ones. Note that here we have different learning rates for different features. We will use the hinge loss:

$$Q((x, y), w) = \max(0, 1 - y(w^\intercal x + \theta)).$$

Since we update both w and θ, we use g_t to denote the gradient vector of Q on the $(n + 1)$ dimensional vector (w, θ) at iteration t.

The per-feature notation at iteration t is: $g_{t,j}$ denotes the jth component of g_t (with respect to w) for $j = 1, \cdots, n$ and $g_{t,n+1}$ denotes the gradient with respect to θ.

In order to write down the update rule we first take the gradient of Q with respect to the weight vector (w_t, θ_t),

$$g_t = \begin{cases} 0 & \text{if } y(w_t^\intercal x + \theta) > 1 \\ -y(x, 1) & otherwise \end{cases}$$

That is, for the first n features, that gradient is $-yx$, and for θ, it is always $-y$.

Then, for each feature j $(j = 1, ..., n + 1)$ we keep the sum of the gradients' squares:

$$G_{t,j} = \sum_{k=1}^{t} g_{k,j}^2$$

and the update rule is

$$w_{t+1,j} \leftarrow w_{t,j} - \eta g_{t,j}/(G_{t,j})^{1/2}$$

By substituting g_t into the update rule above, we get the final update rule:

$$w_{t+1,j} = \begin{cases} w_{t,j} & \text{if } y(w_t^\intercal x + \theta) > 1 \\ w_{t,j} + \eta y x_j/(G_{t,j})^{\frac{1}{2}} & otherwise \end{cases}$$

where for all t we have $x_{n+1} = 1$.

We can see that AdaGrad with hinge loss updates the weight vector only when $y(w^\intercal x + \theta) \leq 1$. The learning rate, though, is changing over time, since $G_{t,j}$ changes with time.

Parameters: η

Initialization: $w^\intercal = \{0, 0, \cdots, 0\}, \theta = 0$

Parameter Recommendation: Choose $\eta \in \{1.5, 0.25, 0.03, 0.005, 0.001\}$.

d. **Averaged Perceptron**: Averaged Perceptron returns a weighted average of a number of earlier hypotheses. **You need to implement Averaged Perceptron for the three versions described above. (it's basically the same implementation)**

We provide the pseudo code for Averaged Perceptron (following the class notes):

Algorithm Averaged Perceptron

Training:
[**m: #(examples); k: #(mistakes) = #(hypotheses);** c_i**: consistency count for** v_i]
Input: a labeled training set (x_1, y_1),...(x_m, y_m), Number of epochs T
Output: a list of weighted perceptrons $(v_1, c_1), ..., (v_k, c_k)$
Initialize: k=0; $v_1 = 0$, $c_1 = 0$
Repeat T times:
for $t = 1 \rightarrow m$ **do**
    Compute prediction $\hat{y} = sgn(v_k \cdot x_i)$
    if $\hat{y} = y_i$ **then**
        $c_k = c_k + 1$
    else
        $v_{k+1} = v_k + y_i x$
        $c_{k+1} = 1$
        $k = k + 1$
    end if
end for
Prediction:
Given: a list of weighted perceptrons $(v_1, c_1), ...(v_k, c_k)$, a new example x
Predict: the label(x) as follows:
$y(x) = sgn[\sum_1^k c_i \; v_i \cdot x]$

Note: While the prediction of the Averaged Perceptrons is done using a weighted average of Perceptron decisions made with the weight vectors $w_1, ..., w_{T+1}$, there is no need to keep *all* these weight vectors. This average **should be implemented** by keeping only two weight vector. A cumulative weight vector computed during the training, and the current one. **You need to think about this and implement this yourself.**

1.2 SVM

While we have not covered SVM in class yet, you will use given libraries for that, so this should not be a problem. See Part 3.2.1 for information on how to convert example formats. We consider soft margin SVMs for non-linearly separable data. You can directly use the library functions in sklearn for the SVM algorithm. Given training sample $S = ((x_1, y_1), ..., (x_m, y_m))$, the objective for SVM is as following:

$$min_{w,b,\xi} \; \frac{1}{2}||w||_2^2 + C \sum_{i=1}^{m} \xi_i$$

subject to

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \; i = 1, ..., m$$

$$\xi_i \geq 0, \; i = 1, ..., m$$

2 Experiments

You will run two experiments, one with synthetic data, and one with real data (the latter will use only a two of the seven algorithms: the averaged version of the basic Perceptron, and SVM.). In each experiment, you will use several algorithms and, for each one, run through these steps:

a. Training and parameter tuning (leading to the final model you will use in this part) on train and development set.

b. Generating learning curves using the models you learned in (a), This will necessitate training each of your algorithms, with the optimal parameter setting chosen in (a) multiple times, each time on a different number of examples, to produce a learning curve.

c. The final model of your learning curve was learned on the training data and will be your final model. You will then compute it's performance on development data and use it to make predictions on the test data given to you.

2.1 Parameter Tuning

One of the goals of this homework is to understand the importance of parameters in a machine learning algorithm.

We will ask you to tune and report the best parameters you chose. You will do it only on the synthetic data set, once on the sparse data and once on the dense data. Only two algorithms, Winnow and Perceptron with AdaGrad, are to be tuned. Basic Perceptron need not be tuned (see above). Similarly, the SVM will not be tuned; use the default sklearn parameters. The Averaged versions will use the same parameter settings as the non-Averaged versions.

The input to the parameter tuning is a training set, a development set and an algorithm (that you develop).

2.1.1 Parameter Tuning Procedure

- You are provided with training and development sets; let's call them Train and Dev respectively. For each parameter value, as given in the algorithms description above, train your algorithm on Train. Then, evaluate the resulting model on Dev and record the accuracy.

- Choose the parameters that results in the highest accuracy on Dev. This is the setting that you will use in the rest of the experiments; we also ask that you give us your best parameters in the report.

2.2 Evaluation

You are given three sets of [training, development and test] datasets. The datasets are explained below and here we just list them:

1. Synthetic Sparse data: training, development, test.
2. Synthetic Dense data: training, development, test.
3. Real data set: Training = CoNLL; Dvelopment = CoNLL and Enron; Test: CoNLL and Enron.

Once you tune parameters as mentioned above, using your best parameters, you should train on the training set, evaluate on development set and predict on the test data.

2.2.1 Experiments with Synthetic data

The key experiment you will run on synthetic data is a learning curve experiment. You will run it twice, for the sparse data and the dense data.

- **Learning curves of algorithms**
  Each of the training sets has $50,000$ examples.You will train 11 models, on $500; 1000; 1,500; \ldots; 5,000; 50,000$ examples, respectively.

  In each case, evaluate the results on the given Dev set. Report the accuracy of each of these 11 hypothesis and plot a curve.

  In your report, comment on the lessons you can draw from the various learning curves.

  Note: For all the six on-line algorithms "training" means running 20[2] iterations on the data (for each of the 11 check points above). SVM is a batch learning algorithm and you only need to run it once for each check point.

- **Performance of algorithms**
  Report the accuracy of each of the 7 algorithms, when trained on all the $50,000$ examples, on the Dev set. (This is the last point in your learning curve).

- **Predicted Label**
  For Averaged basic Perceptron, and SVM, submit the predictions of your final model on the Test set. These will be used by the auto grader.

2.2.2 Experiments with Real data

In this case you will only use two algorithms, Averaged basic Perceptron and SVM, which you will train on the CoNLL training data given to you.

Note that in this case no tuning is needed.

- **Performance of algorithms**
  Report the accuracy of each of the two algorithms, when trained on all the CoNLL training data on the two Dev sets – the CoNLL and the Enron.

- **Predicted Label**
  Submit the predictions of the two models on the two test sets – the CoNLL and the Enron. These will be used by the auto grader.

[2]You can change the iteration number of the Perceptron as you want in the whole homework, even though we suggested 20 here and in other places.

3 Datasets Description

3.1 Synthetic Data

We provide synthetic data for you to evaluate your algorithms. First, we generate examples that are labeled according to a simple l-of-m-of-n boolean function. That is, the function is defined on the n-dimensional Boolean cube $\{0,1\}^n$, and there is a set of m attributes such that an example is positive iff at least l of these m are active in the example. l, m and n define the hidden concept that your algorithms will attempt to learn. The instance space is $\{0,1\}^n$ (that is, there are n boolean features in the domain). We provided two versions of generated synthetic data, sparse and dense. We set $l = 10$ and $m = 20$ for both situations. We set $n = 200$ for sparse data and $n = 40$ for dense data, so you can see the performance of the algorithms both in the sparse and the dense case.

In addition, we added small amount of noise to the training data. The label y is flipped with probability 0.05 and each attribute is flipped with probability 0.001. Consequently, the data not linearly separable.

Each of the algorithms (SVM and the six variations of Perceptron), will be evaluated on both the sparse and the dense data, as described in the evaluation section.

There are 50000 examples in training set, 10000 examples in development set and 10000 examples in test set for both sparse and dense situations.

- **Extra Credits (10%)**, we also provide the data generation code (gen.py and add_noise.py). You can generate datasets with or without the noise option and then compare the results to understand the influence of the noise. Report your results to qualify for the extra credit. You can also use this code to generate easier or more difficult datasets.

3.2 Real-world Problem

The data given is from the named entity recognition (NER) task. The goal is to identify whether strings in text represent names of People, Organizations, Locations, etc. The following text snippets illustrates the annotation for the general NER problem:

For example:

(1) [PER Wolff] , currently a journalist in [LOC Argentina] , played with [PER Del Bosque] in the final years of the seventies in [ORG Real Madrid] .

In this problem set we simplify the task and treat named entity recognition as a binary classification task. The goal is to only identify whether a word is in a named entity or not. We use **I** to denote that the word is in a named entity and **O** to mean that the word is not in any named entity.

Given a sentence $S = w_1, ..., w_n$, you need to predict the {I, O} tags for each word in the sentence. That is, you will produces the sequence $Y = y_1, ..., y_k$, where $y_i \in \{I, O\}$. For instance, the above example is tagged as follows:

(2) [Wolff I] [, O] [currently O] [a O] [journalist O] [in O] [Argentina I] [, O] [played O] [with O] [Del I] [Bosque I] [in O] [the O] [final O] [years O] [of O] [the O] [seventies O] [in O] [Real I] [Madrid I] [. I]

3.2.1 Feature Representation

As for feature representation, we suggest to use **sklearn.feature_extraction.DictVectorizer**. This feature representation can be used to generate sparse example representations for all algorithms on synthetic and real world data. You can also choose other representation methods.

3.2.2 Feature Extraction

The data is provided in column format (that is, each row in the file corresponds to a word in the text, along with its tag. The details can be found in the readme file in the data directory.) The sentences are provided as raw text, so you will need to extract features for the classifier and generate training and test examples.

In this homework, we only consider Boolean contextual features. As you sit on a word w and generate the (training or test) example for this word, the features types should be:

$$w_{-2} = w, w_{-1} = w, w_{+1} = w, w_{+2} = w, w_{-2}\&w_{-1} = w_1w_2, w_{+1}\&w_{+2} = w_1w_2, w_{-1}\&w_{+1} = w_1w_2$$

(7 types), where w_{-1} means the word just before the target word and w_{+1} means the word just after the target word, and w, w_1, w_2 range over all the vocabulary in the training set. That is, $[w_{-1} = w]$ is a Boolean condition that asks whether the word before the target word is 'w'. $[w_{-1}\&w_{+1} = w_1w_2]$ is a Boolean condition asking whether the word before the target is w_1 AND the word after it is w_{-2}. Consequently, each of the 7 types would generate a large number of Boolean features. However, in a given example that corresponds to a word, only up to 7 features will be *active* (value = 1).

Consider, for example, the sentence "Obama traveled to France". Consider the target word "Obama" (that is, we now generate features for the training example corresponding to the word "Obama"). The *active* features will be (w_{+1}= "traveled", w_{+2}= "to", $w_{+1}\&w_{+2}$= "traveled to"); if we consider the target word "traveled", the features will be (w_{-1} = "Obama", w_{+1}="to", w_{+2}="France", $w_{+1}\&w_{+2}$= "to France", $w_{-1}\&w_{+1}$= "Obama to").

In order to deal with the first two words and the last two words in a sentence, we will add special symbol "SSS" and "EEE" to the vocabulary to represent the words before the first word and the words after the last word.

In your implementation you will generate 7 different dictionaries, one for each feature type, and you will instantiate them using the training data. Notice that in the test data you may encounter a word that was not observed in training, and therefore is not in your dictionary. In this case, you cannot generate a feature for it, resulting in *less* than 7 active features in some of the test examples.

3.2.3 News Dataset (CoNLL)

This dataset contains a collection of news articles. We choose to use the CoNLL dataset [3]. We provided a reader to this data as well as features extractors for two of the types. You will have to implement the other feature types.

Since you are running only the Averaged basic Perceptron and the SVM on this data sets, no tuning is necessary. You will only train your models in the news dataset and evaluate it on the development set provided. Finally, you will also submit the prediction on your trained models on the CoNLL test data.

Note that the training, development and test data contain, respectively, 14987 sentences (204567 words), 336 sentences (3779 words), 303 sentences (3880 words). The number of entities in the training and development datasets is 34043 and 772, respectively.

3.2.4 Email Dataset (Enron)

We also provided a different dataset, only for testing. This is an email dataset – the Enron dataset [4]). You do not need to train on this data set; only evaluate your trained models on the development set, and compare with the results you got on the CoNLL development set. This should give you an idea of the difficulty of transfer learning; make sure to present your findings in the report.

Note that the development and test data contain, respectively, 310 sentences (8541 words) and 368 sentences (11852 words). The number of entities in the development set is 562.

[3] https://www.clips.uantwerpen.be/conll2003/ner/

[4] https://www.cs.cmu.edu/∼enron/

What we provide

- The code for the basic version of Perceptron.
- The python readers for the synthetic and the real-world data.
- The feature extractors of two feature types, w_{-1} and w_1 for real-world data.

Although the code is provided, you can use your own method.

What to submit

- A detailed report of your experiments.
  - You will have 14 learning curve plots in total from the 7 algorithms evaluated on the synthetic datasets (sparse and dense).
  - Then you will provide the accuracy of two algorithms: Averaged basic Perceptron and SVM. You will report 8 accuracies, for 2 algorithms on 4 datasets, two in the synthetic data and two in real-world data.
  - Include your observation on the results. Discuss differences you see in the performance of the algorithms across target functions and try to explain it. Make sure you discuss each of the plots, as well as the final results.
- Your source code. This should include the algorithm implementation and the code that runs the experiments. You **must** include a README (no extension), documenting how someone should run your code.
- Provide your prediction on the test data sets provided. You will only evaluate the Averaged basic Perceptron and the SVM on the four different test datasets. You should include $2 * 4 = 8$ predicted label files in total. The test labels files should be named as follows:

  svm-conll.txt, svm-enron.txt, svm-dense.txt, svm-sparse.txt, p-conll.txt, p-enron.txt, p-dense.txt, p-sparse.txt.

  The labels should be in the provided format, -1,1 (for the synthetic data) and O,I (for the real data). The label files should have one label per line with no empty lines. The order of test labels should respect the order of test data.

  The real data test labels should look like:
  I
  O
  I
  ...
  And the synthetic data test labels should look like:
  -1
  1

-1
...

- To submit homework 2, transfer all of your files into your SEAS account on eniac. Then, from a command prompt on eniac, use the turnin command to submit all files:

  ```
  turnin -c cis519 -p hw2 hw2.pdf README hw2.py test-labels.zip
  ```

  Make certain that your submitted files are named correctly, and do not submit additional files, or place the files in a folder. You can check that your submission was received by running:

  ```
  turnin -c cis519 -p hw2 -v
  ```

  to list the files submitted. You should see one listing for each file submitted. The turnin command will also send you an email with a subject "cis519: submission received" to verify receipt. Subsequent submissions will overwrite earlier submissions; if you re-submit files after the due date, your submission will be counted as late regardless of whether or not the file contents changed. The official timestamp of your submission will be the most recent (i.e., latest) timestamp of ANY file you submit. Therefore, if you submit even one file late, your entire submission will be considered late.

- In addition, you should zip the files and submit the zip folder on Canvas.