COMS 4771 HW2 (Spring 2022)

Due: Mon March 07, 2022 at 11:59pm

This homework is to be done **alone**. No late homeworks are allowed. To receive credit, a typesetted copy of the homework pdf must be uploaded to Gradescope by the due date. You must show your work to receive full credit. Discussing possible approaches for solutions for homework questions is encouraged on the course discussion board and with your peers, but you must write their own individual solutions and **not** share your written work/code. You must cite all resources (including online material, books, articles, help taken from/given to specific individuals, etc.) you used to complete your work.

1 A variant of Perceptron Algorithm

One is often interested in learning a *disjunction model*. That is, given binary observations from a d feature space (i.e. each datapoint $x \in \{0,1\}^d$), the output is an 'OR' of few of these features (e.g. $y = x_1 \vee x_{20} \vee x_{33}$). It turns out that the following variant of the perceptron algorithm can learn such disjunctions well.

Perceptron OR Variant
learning:
- Initialize $w := 1$
- for each datapoint (x, y) in the training dataset
- $\hat{y} := \mathbf{1}[w \cdot x > d]$
- if $y \neq \hat{y}$ and $y = 1$
- $w_i \leftarrow 2w_i$ (for $\forall i : x_i = 1$)
- if $y \neq \hat{y}$ and $y = 0$
- $w_i \leftarrow w_i/2$ (for $\forall i : x_i = 1$)

classification:
$f(x) := \mathbf{1}[w \cdot x > d]$

We will prove the following interesting result: The **Perceptron OR Variant** algorithm makes at most $2 + 3r(1 + \log d)$ mistakes when the target concept is an OR of r variables.

(i) Show that for any positive example, **Perceptron OR Variant** makes $M_+ \leq r(1 + \log d)$ mistakes.

(ii) Show that each mistake made on a negative example decreases the total weight $\sum_i w_i$ by at least $d/2$.

(iii) Let M_- denote the total number of mistakes on negative examples, and $TW(t)$ denote the total weight of w at iteration t. Observing that

$$0 < TW(t) \leq TW(0) + dM_+ - (d/2)M_-,$$

conclude that the total number of mistakes (i.e., $M_+ + M_-$) is at most $2 + 3r(1 + \log d)$.

2 Understanding the RBF Kernel

Recall the RBF kernel with parameter $\gamma > 0$ which, given $x, x' \in \mathbb{R}^d$, is defined as $K(x, x') = \exp\left(-\gamma\|x - x'\|^2\right)$. In this question we will try to understand what feature transformation this kernel function corresponds to and what makes this kernel function so powerful and useful in practice.

For the sake of simplicity[1], assume that $d = 1$. Let ϕ be the feature transform which maps an input $x \in \mathbb{R}$ to an infinite dimensional vector whose ith (i takes on values $0, 1, 2, \dots$) coordinate is $(\phi(x))_i = \sqrt{\frac{2^i\gamma^i}{i!}}\exp(-\gamma x^2)x^i$. For example $\phi(2)$ is an infinite dimensional vector whose first few entries are as follows:

$$\phi(2) = \begin{pmatrix} e^{-4\gamma} \\ 2\sqrt{2\gamma}e^{-4\gamma} \\ 4\gamma e^{-4\gamma} \\ \frac{8\gamma\sqrt{2\gamma}}{3}e^{-4\gamma} \\ \vdots \end{pmatrix}.$$

We can then define $\phi(x) \cdot \phi(x') = \sum_{i=0}^{\infty} \phi(x)_i\phi(x')_i$.

(a) Show that $K(x, x') = \phi(x) \cdot \phi(x')$ for the specific K and ϕ defined above.

Hint: You can use the fact that for any $x \in \mathbb{R}$: $\exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ (this is just the Taylor series of $\exp(x)$).

Consider any polynomial function $g : \mathbb{R}^d \to \mathbb{R}$ (where again we assume $d = 1$ for simplicity). Without loss of generality (which is often abbreviated WLOG and just means that this assumption is always true and so isn't really an assumption) we can write $g(x) = \sum_{i=1}^{m} a_i x^{p_i}$ for some $m > 0$, coefficients $a_1, a_2 \dots, a_m \in \mathbb{R}$, and some distinct non-negative integers $p_1, p_2, \dots, p_m$.

(b) Show that there is an infinite length vector w such that $g(x) = \exp\left(\gamma x^2\right) w \cdot \phi(x)$. Conclude that any polynomial boundary $g(x) = 0$ becomes linear when using the RBF kernel. Specifically, show that $g(x) = 0$ iff $w \cdot \phi(x) = 0$ for the w you found.

(c) Look at how the parameter γ affects the coefficients of your vector w from part (b). Use this observation to explain the relationship between γ and over/under-fitting. If you have lots of data and know that the "best" boundary is a high degree polynomial, should γ be big or small? What if you have very little data and are afraid of fitting to noise, should γ be large or small?

[1]There is a d dimensional equivalent to everything you will show below, and the conclusions are the same. The only difference is that the algebra gets uglier. If that doesn't scare you, feel free to do this question without the $d = 1$ assumption (you'll need to come up with the correct ϕ for d-dimensions and change how we write g).

3 Analyzing Games via Optimization

Recall the game of rock-paper-scissors. In this game there are two players, p_1 and p_2, that each choose an element of $\{r, p, s\}$ as their only move for the game. One can study such games more systematically. For example, this game can be analyzed by examining the possible moves each player can make and representing the outcomes as a **payoff matrix** P. In particular, given that p_1 plays move $i \in \{r, p, s\}$ and p_2 plays move $j \in \{r, p, s\}$, the resulting game *payoff* (from the perspective of p_1) is the amount P_{ij}.

(i) What is the payoff matrix for p_1 in the game of rock paper scissors, where winning results in a payoff of 1, losing a payoff of -1, and a tie a payoff of 0?

(ii) If the matrix Q is the payoff from the perspective of player p_2, then what is the relationship between P and Q?

Now consider a game with payoff matrix $A \in \{-1, 1\}^{n \times n}$ (notice, $+1$ means win, -1 means lose, and there are no ties in this game). There are two players, p_1 and p_2, that must choose moves i and $j \in \{1, \ldots, n\}$, respectively. Similar to rock-paper-scissors, neither p_1 nor p_2 knows the other's choice before making their own choice, but they do have access to the payoff A. Following their choices of i and j, p_1 receives reward $A_{i,j}$. Of course, p_1's objective is to receive maximum possible payoff, and since p_2 and p_1 are in competition, p_2's objective is for p_1 to receive minimum possible payoff.

(iii) Describe the set of matrices $S \subset \{-1, 1\}^{n \times n}$ s.t. $\forall A \in S$, there exists some choice of $i \in \{1, \ldots, n\}$ that guarantees a win for p_1 (no more than two sentences are needed).

If neither p_1 nor p_2 has a move that can result in a guaranteed win for them, they might have some tough choices to make about how to select their move. Suppose they can each formulate a probabilistic strategy – instead of committing to a particular move as their strategy, they can each define a probability distribution over the moves $\{1, \ldots, n\}$ for p_1 and p_2, as their strategy (meaning, at play time, a move is randomly selected according to the described probability distribution). We can define the strategy of player i as $s^i \in \mathbb{R}^n$, such that s^i_k is the probability that player i chooses to play move k. Naturally, we have that $\quad \forall i, k \quad s^i_k \geq 0$ and $\sum_j s^i_k = 1$.

(iv) We will now analyze the situation for when $n = 2$. (Note that this analysis can be extended naturally to cases $n > 2$.) Assuming that p_1 chooses strategy $s^1 = \begin{bmatrix} s^1_1 \\ s^1_2 \end{bmatrix}$, what is the best possible strategy s^2 for p_2?
(notice that a best s^2 is necessarily deterministic)

(v) Now formulate an optimization problem over s^1, which yields p_1's best strategy in defense of p_2 (this should follow directly from part (iv))?

(vi) Show that the optimization in (v) can be reformulated as:

$$\begin{aligned} \min_{s^1} \; & -z \\ \text{s.t. } & z \le A_{11}s_1^1 + A_{21}s_2^1 \\ & z \le A_{12}s_1^1 + A_{22}s_2^1 \\ & s_1^1 + s_2^1 = 1 \\ & s_1^1, s_2^1 \ge 0 \end{aligned}$$

and show that this is a convex optimization problem (in s^1).

(vii) What is the Lagrange dual of the problem in (vi)?

Observe that, interestingly, if we repeat parts (iv)-(vi) from the perspective of p_2, we arrive at the same optimization as the dual of the analysis of p_1 perspective. In game theory, this equivalence emerges as a result of duality is known as the minimax theorem, and is the reason why it does not matter if any of the players announce their optimal strategies *a priori*.

4 Predicting Restaurant-reviews via Perceptrons

Download the review dataset `restaurant_reviews_data.zip`. This data set is comprised of reviews of restaurants in Pittsburgh; the label indicates whether or not the reviewer-assigned rating is at least four (on a five-point scale). The data are in CSV format (where the first line is the header); the first column is the label (`label`; 0 or 1), and the second column is the review text (`text`). The text has been processed to remove non-alphanumeric symbols and capitalization. The data has already been separated into training data `reviews_tr.csv` and test data and `reviews_te.csv`.

The first challenge in dealing with text data is to come up with a reasonable "vectorial" representation of the data so that one can use standard machine learning classifiers with it.

Data-representations

In this problem, you will experiment with the following different data representations.

1. Unigram representation.

   In this representation, there is a feature for every word t, and the feature value associated with a word t in a document d is

   $$\mathrm{tf}(t;d) := \text{number of times word } t \text{ appears in document } d.$$

   (tf is short for *term frequency*.)

2. *Term frequency-inverse document frequency (tf-idf)* weighting.

   This is like the unigram representation, except the feature associated with a word t in a document d from a collection of documents D (e.g., training data) is

   $$\mathrm{tf}(t;d) \times \log_{10}(\mathrm{idf}(t;D)),$$

   where $\mathrm{tf}(t;d)$ is as defined above, and

   $$\mathrm{idf}(t;D) := \frac{|D|}{\text{number of documents in } D \text{ that contain word } t}.$$

This representation puts more emphasis on rare words and less emphasis on common words. (There are many variants of tf-idf that are unfortunately all referred to by the same name.)

Note: When you apply this representation to a new document (e.g., a document in the test set), you should still use the idf defined with respect to D. This, however, becomes problematic if a word t appears in a new document but did not appear in any document in D: in this case, $\mathrm{idf}(t; D) = |D|/0 = \infty$. It is not obvious what should be done in these cases. For this homework assignment, simply ignore words t that do not appear in any document in D.

3. Bigram representation.

   In addition to the unigram features, there is a feature for every *pair* of words (t_1, t_2) (called a *bigram*), and the feature value associated with a bigram (t_1, t_2) in a given document d is

   $$\mathrm{tf}((t_1, t_2); d) := \text{number of times bigram } (t_1, t_2) \text{ appears consecutively in document } d.$$

   In the sequence of words "a rose is a rose", the bigrams that appear are: $(\mathrm{a}, \mathrm{rose})$, which appears twice; $(\mathrm{rose}, \mathrm{is})$; and $(\mathrm{is}, \mathrm{a})$.

For all the of these representations, ensure to do data "lifting".

Online Perceptron with online-to-batch conversion

Once can implement the Online Perceptron algorithm with the following online-to-batch conversion process.

- Run Online Perceptron to make *two* passes through the training data. Before each pass, randomly shuffle the order of the training examples. Note that a total of $2n + 1$ linear classifiers $\hat{w}_1, \ldots, \hat{w}_{2n+1}$ are created during the run of the algorithm, where n is the number of training examples.
- Return the linear classifier $\hat{w}_{\mathrm{final}}$ given by the simple average of the final $n + 1$ linear classifiers:

$$\hat{w}_{\mathrm{final}} := \frac{1}{n+1} \sum_{i=n+1}^{2n+1} \hat{w}_i.$$

Recall that as Online Perceptron is making its pass through the training examples, it only updates its weight vector in rounds in which it makes a prediction error. So, for example, if $\hat{w}_1$ does not make a mistake in classifying the first training example, then $\hat{w}_2 = \hat{w}_1$. You can use this fact to keep the memory usage of your code relatively modest.

(i) What are some potential issued with Unigram representation of textual data?

(ii) Implement the online Perceptron algorithm with online-to-batch conversion. You must submit your code to receive full credit.

(iii) Which of the three representations give better predictive performance? As always, you should justify your answer with appropriate performance graphs demonstrating the superiority of one representation over the other. Example things to consider: you should evaluate how the classifier behaves on a holdout 'test' sample for various splits of the data; how does the training sample size affects the classification performance; etc.

(iv) For the classifier based on the unigram representation, determine the 10 words that have the highest (i.e., most positive) weights; also determine the 10 words that have the lowest (i.e., most negative) weights.