Mark Shinozaki
Homework #3

Q1. Answer the following with a yes or no along with proper justification.

a. Is the decision boundary of voted perceptron linear?
   - Yes, the decision boundary of voted perceptron is linear. Voted perceptron is an extension of the perceptron algorithm that uses multiple perceptron's and combines their decisions through a voting mechanism. Each perceptron in the ensemble still learns a linear decision boundary, and the final decision boundary is linear combination of individual decision boundaries.

b. Is the decision boundary of averaged perceptron linear?
   - Yes, the decision boundary of averaged perceptron is also linear. Averaged perceptron is another extension of the perceptron algorithm that updates the weight vector based on the average of all previous weight vectors. The final weight vector is linear combination of all the weight vectors, and thus the decision boundary is also linear.

Q2. (5 points) Consider the following setting. You are provided with n training examples: $(x_1, y_1, h_1), (x_2, y_2, h_2), \cdots, (x_n, y_n, h_n)$, where x_i is the input example, y_i is the class label (+1 or -1), and $h_i > 0$ is the importance weight of the example. The teacher gave you some additional information by specifying the importance of each training example. How will you modify the perceptron algorithm to be able to leverage this extra information? Please justify your answer.

- I believe the best thing to is have a perceptron algorithm that takes into account the importance of weights. Having a perceptron algorithm that takes into account the importance weights is useful because we could have an imbalance of classes and not useful examples in the training set. In this modified algorithm, each training example is given a weight based on its importance, and the updates to the weight vector during training are weighted by these importance weights. The overall idea is to give more importance to the examples that more informative, while still using all examples to update the weight vector.

- The input of the algorithm, X is the set of n training examples (xi, yi, hi) and T, will be the learned weight vectors.

- The first would be to set w to the learned weight vector

- For each iteration t from 1 to T

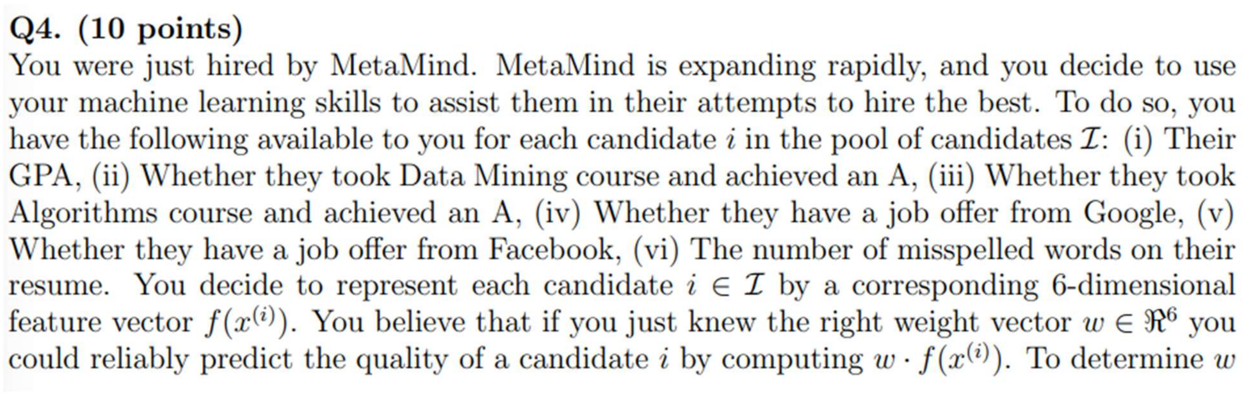
  - Shuffle the training examples in X
  - For each example $(x_i, y_i, h_i)\ in\ X$:
    - Compute the prediction for x_i: $y_{\_temp} = sign(w\ \times x_i)$
    - If $y_{\_temp}$ does not equal y_i then the weight vector is updated to:
      - w <- w + $h_i \times y_i \times x_i$
  - Then compute the training error by counting the number of misclassified examples
  - Then you want to print the training error rate for iteration t
  - And lastly, return the weight vector w
- The importance weight h_i is used to scale the update to weight vector during training. If an example happens to have higher importance weight, it will contribute more to the update of

the weight vector. This entails that the perceptron algorithm will focus more on the informative examples which improves its performance.

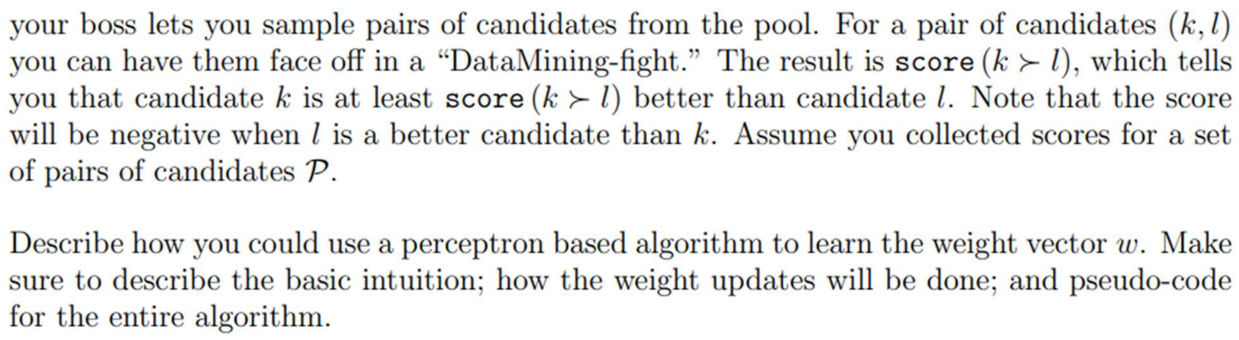
Q3. (5 points) Consider the following setting. You are provided with n training examples: $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, where x_i is the input example, and y_i is the class label (+1 or -1). However, the training data is highly imbalanced (say 90% of the examples are negative and 10% of the examples are positive) and we care more about the accuracy of positive examples. How will you modify the perceptron algorithm to solve this learning problem? Please justify your answer.

- Similar to the last question, one solution is to modifying the perceptron by weighing the particular class. In the case of class imbalance, the weight of the lower class is increased by some factor to make sure that one class is proportional to the other class.
    - We could modify the perceptron algorithm to incorporate class weighting:
        - We assign higher weight to positive examples: we can assign a higher weight to the positive examples during training. One common approach is to use the inverse frequency of each class as its weight. In this case, since 90% of the examples and a weight of .9 to negative examples.
        - Update the perceptron weights based on the weighted error: during the training, we need to update the perceptron weights based on the weighted error, which is the product of the weight of each example and its classification error.

Mark Shinozaki
Homework #3



Q4. (10 points)
You were just hired by MetaMind. MetaMind is expanding rapidly, and you decide to use your machine learning skills to assist them in their attempts to hire the best. To do so, you have the following available to you for each candidate i in the pool of candidates $\mathcal{I}$: (i) Their GPA, (ii) Whether they took Data Mining course and achieved an A, (iii) Whether they took Algorithms course and achieved an A, (iv) Whether they have a job offer from Google, (v) Whether they have a job offer from Facebook, (vi) The number of misspelled words on their resume. You decide to represent each candidate $i \in \mathcal{I}$ by a corresponding 6-dimensional feature vector $f(x^{(i)})$. You believe that if you just knew the right weight vector $w \in \Re^6$ you could reliably predict the quality of a candidate i by computing $w \cdot f(x^{(i)})$. To determine w



your boss lets you sample pairs of candidates from the pool. For a pair of candidates (k, l) you can have them face off in a "DataMining-fight." The result is $\texttt{score}\,(k \succ l)$, which tells you that candidate k is at least $\texttt{score}\,(k \succ l)$ better than candidate l. Note that the score will be negative when l is a better candidate than k. Assume you collected scores for a set of pairs of candidates $\mathcal{P}$.

Describe how you could use a perceptron based algorithm to learn the weight vector w. Make sure to describe the basic intuition; how the weight updates will be done; and pseudo-code for the entire algorithm.

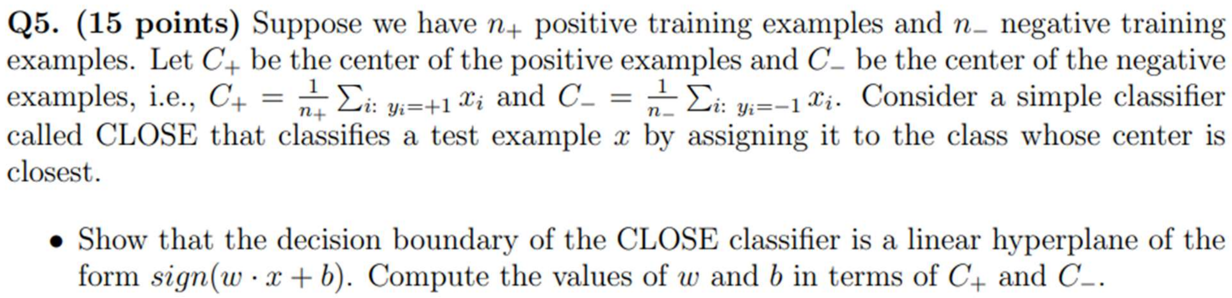
- In this question, to learn the weight vector w for the candidate evaluation task, we can use a variant of the perceptron algorithm called the pairwise perceptron algorithm. This particular algorithm is used to learn a linear ranking function that can compare pairs of candidates and output a score that indicates which candidate is more suitable.
- The basic intuition behind the pairwise perceptron algorithm is to use pairs of examples to learn a ranking function that assigns a higher score to better candidates. The algorithm updates the weight vector based on the difference in scores between pairs of examples, with the goal of maximizing the margin between the scores of the winning and losing candidates in each pair.
- For this specific example, to apply the pairwise perceptron algorithm we have to represent each candidate I as a 6-dimensional feature vector f(x^(i)), where the first dimension is GPA and the second and third are the binary indicators for whether the candidate achieved an A in the class and the algorithm courses, the fourth and fifth would be binary indictors for whether the candidate has a job offer from google or facebook and the sixth dimension is the number of misspelled words in the candidate's resume.
  - We need to initialize the weight vector w to zero, then for each iteration t, we shuffle the pairs of candidates in P, for each pair of candidates (k,l) in P:,

```
1. Initialize the weight vector w to zero.
2. For each iteration t from 1 to T:
   a. Shuffle the pairs of candidates in P.
   b. For each pair of candidates (k,l) in P:
      i. Compute the score difference between the candidates
   c. Compute the training error rate by counting the number of pairs that are misclassified by the
current weight vector
```

d. Compute the validation error rate by counting the number of validation examples that are misclassified by the current weight vectors

e. Print the training and validation error rates for iteration t.

Then lastly, return the learned weight vector w.



Q5. (15 points) Suppose we have n_+ positive training examples and n_- negative training examples. Let C_+ be the center of the positive examples and C_- be the center of the negative examples, i.e., $C_+ = \frac{1}{n_+}\sum_{i:\ y_i=+1} x_i$ and $C_- = \frac{1}{n_-}\sum_{i:\ y_i=-1} x_i$. Consider a simple classifier called CLOSE that classifies a test example x by assigning it to the class whose center is closest.

- Show that the decision boundary of the CLOSE classifier is a linear hyperplane of the form $sign(w \cdot x + b)$. Compute the values of w and b in terms of C_+ and C_-.

  - The CLOSE boundary is a hyperplane that is linearly separable. A distinction is made with positive and negative training examples. When the positive is on one side of a linear line and the negative is on the other. The Training example is considered as support vectors are the ones that are closest to that linear separator.

Mark Shinozaki
Homework #3

Q6. (10 points) Please read the following paper and write a brief summary of the main points in at most TWO pages.

Pedro M. Domingos: A few useful things to know about machine learning. Communications of ACM 55(10): 78-87 (2012)
`https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf`

Summary of the article

In the first part of the article, the authors go over a summary of important topics and a brief overview of the kind of concepts that machine learning covers. This article discusses machine learning systems, which are programs that automatically learn from data, and their use in various fields such as web search, spam filters, and fraud detection. It explains the three components of machine learning: representation, evaluation, and optimization, and how they apply to classification, which is the most commonly used type of machine learning. The article also provides examples of machine learning algorithms for each component discusses some the key issues in choosing them. The next part of the article also discusses fundamental parts of machine learning, it discusses that a goal of machine learning is to have generalization beyond the training set. Essentially, its easy to do well on the training set by just memorizing examples but warns that its not indictive of success on new data. The article also discusses the importance of separating the training and the test data sets to reduce contamination of the classifier by test data, and the need for cross-validation. The article emphasizes that data alone is not enough, and some knowledge or assumptions beyond that data must be embodied in the learner to generalize beyond it. Finally, it explains that the choice of representation is crucial as it determines the kinds of knowledge easily expressed in it, and that the most useful learners are those that embody knowledge relevant to the domain. The next part of the article discusses two major problems in machine learning, overfitting and curse of dimensionality. Overfitting is when the model learns to capture the noise in the training data, causing it to perform poorly on new data. The text provides several techniques to combat overfitting, including regularization and statistical significance tests. The curse of dimensionality refers to the difficulty in generalizing correctly when the number of features in the input space increases. As the number of features increases, a fixed-sized training set covers a dwindling fraction of the input space, making generalization exponentially harder. Additionally, the similarity-based reasoning that machine learning algorithms depend on break downs in high dimensions. In high dimensions, examples become increasingly alike, and the choice of the nearest neighbor (and therefore of the class) becomes effectively random. Next the article discusses the most common type of guarantee is a bound on the number of examples needed for good generalization. The article explains the simple argument behind these bounds and their pessimistic nature due to the hypothesis space. Theoretical guarantees are useful for algorithm design and understanding but not as a criterion for practical decisions, The most important factor for success in machine learning projects is feature engineering. Learning is easier when independent features correlate well with the class. On the other hand, if the features are correlated or noisy, learning becomes more difficult, and overfitting may occur. The article concludes by emphasizing the importance of human expertise in feature engineering and iterative nature of machine learning projects. In the final section of the article the text discusses some of the more challenges that exist in machine learning, one such issue refers to the limitations of Occam's razor. It argues that there is no necessary connection between the number of parameters of a model and tis tendency to overfit, and that smaller hypothesis spaces may not necessarily result in better generalization performance. Furthermore, the fact that a function can be represented does not mean it can be learned, and the existence of many local optima can make it difficult for learner to find the true function. The text also emphasizes the importance of trying different learners and representations and suggests that finding methods to learn deeper representations is one of the major research frontiers in machine learning. Finally in conclusion, the wrapping up part of this article, It argues that there is no necessary connection between the number of parameters of a model and its tendency to overfit, and that smaller hypothesis spaces may not necessarily result in better generalization performance. Furthermore, the fact that a function can be represented does not mean it can be learned, and the existence of many local optima can make it difficult for a learner to find the true function. The text also emphasizes the importance of trying different learners and representations and suggests that finding methods to learn deeper representations is one of the major research frontiers in machine learning. Finally, it cautions against the common mistake of assuming that correlation implies causation and suggests that machine learning models should be used as guides to action rather than definitive proof of causation.