EE 511 Machine Learning, Winter 2018: Homework 2

Due: Wednesday, January 31st, beginning of class

1 Fitting an SVM classifier by hand [50 Points]

(Source: Murphy text, Exercise 14.1) Consider a dataset with 2 points in 1d:

$$(x_1 = 0, y_1 = -1) \text{ and } (x_2 = \sqrt{2}, y_2 = 1).$$

Consider mapping each point to 3d using the feature vector $\phi(x) = [1, \sqrt{2}x, x^2]^T$ (This is equivalent to using a second order polynomial kernel). The max margin classifier has the form

$$\hat{w}, \hat{w_0} = \arg\min ||w||^2 \quad s.t. \tag{1}$$
$$y_1(w^T\phi(x_1) + w_0) \geq 1 \tag{2}$$
$$y_2(w^T\phi(x_2) + w_0) \geq 1 \tag{3}$$

1. (10 Points) Write down a vector that is parallel to the optimal vector $\hat{w}$. Hint: Recall that $\hat{w}$ is perpendicular to the decision boundary between the two points in the 3d feature space.

2. (10 Points) What is the value of the margin that is achieved by this $\hat{w}$? Hint: Recall that the margin is the distance from each support vector to the decision boundary. Hint 2: Think about the geometry of the points in feature space, and the vector between them.

3. (10 Points) Solve for $\hat{w}$, using the fact the margin is equal to $1/||\hat{w}||$.

4. (10 Points) Solve for $\hat{w_0}$ using your value for $\hat{w}$ and Equations 1 to 3. Hint: The points will be on the decision boundary, so the inequalities will be tight. A "tight inequality" is an inequality that is as strict as possible. For this problem, this means that plugging in these points will push the left-hand side of Equations 2 and 3 as close to 1 as possible.

5. (10 Points) Write down the form of the discriminant function $f(x) = \hat{w_0} + \hat{w}^T\phi(x)$ as an explicit function of x. Plot the 2 points in the dataset, along with $f(x)$ in a 2d plot. You may generate this plot by hand, or using a computational tool like Python.

2 Perceptrons [50 points]

Recall that a perceptron learns a linear classifier with weight vector w. It predicts

$$y^{(i)} = \text{sign}(w^T x^{(i)})$$

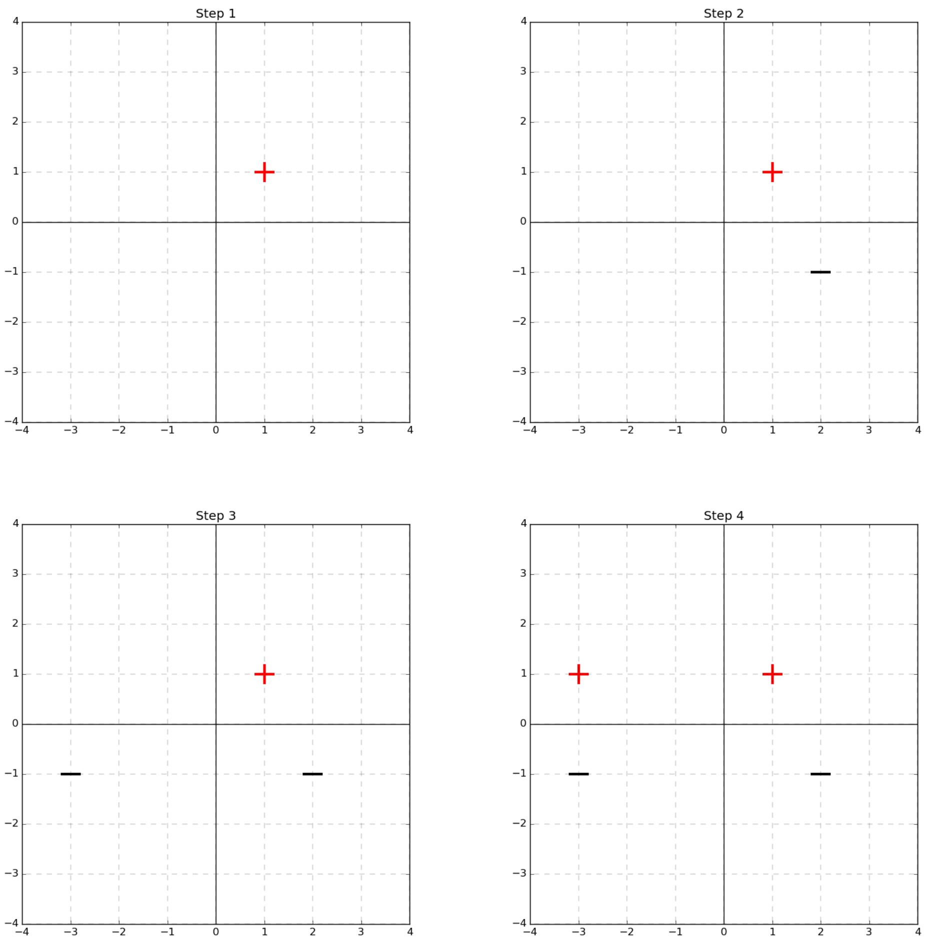
(assuming here that $y^{(i)} \in \{+1, -1\}$. Also, note that we are *not* using a bias weight w_0, for simplicity). When the perceptron makes a mistake, it updates the weights using the formula

$$w = w + y^{(i)}x^{(i)}$$

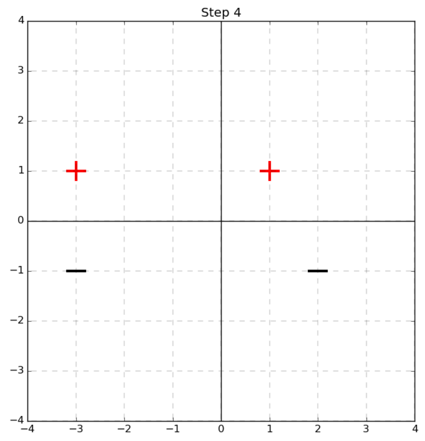
Imagine that we have $x^{(i)} \in \mathbb{R}^2$, and we encounter the following data points

x_1	x_2	y
1	1	1
2	-1	-1
-3	-1	-1
-3	1	1

1. (25 points) Starting with $w = [0 \quad 0]^T$, use the perceptron algorithm to learn on the data points in the order from top to bottom. Show the perceptron's linear decision boundary after observing each data point in the graphs below. Be sure to show which side is classifed as positive.



2. (10 points) Does our learned perceptron maximize the margin between the training data and the decision boundary? If not, draw the maximum-margin decision boundary on the graph below.



3. (15 points) Assume that we continue to collect data and train the perceptron. If all data we see (including the points we just trained on) are linearly separable separable with margin $\gamma = 0.5$ and have maximum norm $||x^{(i)}|| \leq 5$, what is the maximum number of mistakes we can make on future data?

3 Logistic Regression [100 points]

In this assignment, you will be predicting income from census data using logistic regression. Specifically, you will predict the probability that a person earns more than $50k per year. For more details about the dataset, please visit `http://www.cs.toronto.edu/~delve/data/adult/adultDetail.html`.

Please download the code.zip file. In the code fold, you will be modifying the file **sgd.py**. Tests for your program are in **test.py**. To test your implementation, run: `python test.py`.

- (10 points) **Logistic Function**: To perform logistic regression you have to be able to calculate the logistic function. Fill in the `logistic` function.
- (5 points) **Dot Prodct:** The model you are training is just a bunch of numerical weights. To run your model on a data points you will need to take the dot product of your weights and the features for that data point and run the result through your logistic function. First fill in the `dot` function to take the dot product of two vectors

- (5 points) **Prediction:** Now that you can calculate the dot product, predicting new data points should be easy! Fill in the `predict` function to run your model on a new data point. Take a look at `test.py` to see what the format for data points is.

  Prediction should be straightforward, to predict new points you simply multiply your model's weights by the corresponding features, sum up the result, and pass it through the logistic function. This should be easy with your dot product and logistic functions.

- (10 points) Once you start training your model you are going to want to know how well you are doing. Modify the `accuracy` function to calculate your accuracy on a dataset given a list of data points and the associated predictions.

- (20 points) **Train Your Model:** Fill in the `train` and `update` functions to train your model! You should use logistic regression with L2 regularization where `rate` is the learning rate and `lam` is the regularization parameter.

  The training should run for some number of epochs performing stochastic gradient descent.

  This means you will randomly select a point from the dataset and run the model on that data point. Then you will calculate the error for that point and adjust your model weights based on the gradient of that error. An epoch refers to a full pass over the dataset. In practice it is easier (and more statistically valid) to sample randomly with replacement. Thus an epoch just means examining N data points where N is the number of points in your training data.

  This is different than batch gradient descent where you look at all of the data points before updating. SGD converges faster but can also be less stable because you have a noisy estimate of the gradient instead of the true gradient. In practice it is often much better to use SGD than full batch gradient descent.

  When you see a new data point, you prediction will be:

  ```
  Prediction = P(income > 50k | W, x) = logistic(Wx)
  ```

  Your loss function will be:

  ```
  Loss = (Prediction - Truth)^2 + Lambda * || W ||
  ```

  By minimizing this loss, the model learns to make correct predictions and also use small weights to avoid overfitting.

  To adjust the model you have to calculate the gradient of the loss at a given point. The gradient will come from two sources, the error and the regularization.

  While this minimizing this loss looks different than maximizing the conditional log likelihood, the gradient is actually the same, just with opposite sign. Thus you can either descend the gradient of this loss function or ascend the gradient of the conditional log likelihood and you will be performing the same updates. If you want, you can derive why this is the case by taking the partial derivative of the above loss function with respect to a single weight in the model. Use the update rule from class to adjust the model, but remember, since we are doing SGD you only look at one point before updating the model:

  $$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left(-\lambda w_i^{(t)} + \sum_j x_i^j \left[y^j - \hat{P}(Y^j = 1 | \mathbf{x}^j, \mathbf{w}) \right] \right)$$

  When you run python `test.py` it will tell you your current accuracy on the training and validation set. By default these are the same dataset! To get a more accurate evaluation you can modify `data.py` to use different training and validation sets by splitting your data.

- (20 points) **Extract Better Features:** Take a look at the feature extracting code in `extract_features`, and at the raw data in adult.data. Right now your model is only considering age, education, and one possible marital status.

  Good feature extraction is often the key to making good machine learning models. Add more feature extraction rules to help improve your model's performance. This is very open ended, be creative and find features that work well with your model.

- (30 points) **Tune Your Submission:** Tune your `submission` function to train your final model. You should change your feature extraction and training code to produce the best model you can. Try different learning rates and regularization parameters, how do they compare? Often it is good to start with a high learning rate and decrease it over time, feel free to add this to your training code.

  Your final model will be trained on the full training data and run on test data that you don't have access to. Your grade for this section will be based on your performance relative to an untuned baseline and to the other other students in class. Good luck!

- Only submit your modified `sgd.py` file.