Homework 3

Boosting, Decision Trees, Neural Networks, Model Selection

CMU 10-701: Introduction to Machine Learning (Spring 2018)

OUT: February 26, 2018
DUE: **March 9, 2018, 10:30 AM**

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 3.4"). Second, write your solution independently: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.
- **Submitting your work:** Assignments should be submitted as PDFs using Gradescope unless explicitly stated otherwise. Each derivation/proof should be completed on a separate page. Submissions can be handwritten, but should be labeled and clearly legible. Else, submissions can be written in LaTeX. Upon submission, label each question using the template provided by Gradescope. Please refer to Piazza for detailed instruction for joining Gradescope and submitting your homework.
- **Programming**: All programming portions of the assignments should be submitted to Gradescope as well. We will not be using this for autograding, meaning you may use any language which you like to submit.

1 Boosting and Decision Trees (20 pts) [Dimitris and Lam]

1.1 Build your own Decision Tree [12 pts]

In the previous homework assignment, you had a lot of fun implementing a Naive Bayes classifier that predicts whether a person makes over $50K a year based on various attributes about this person. This time, you decide to predict a person's mobile phone usage instead.

You obtained the following data from interviewing 15 people on the street. Based on a person's relationship status, age, education level and income, you can now build a decision tree to predict a person's phone usage. In this section, you can assume that the decision tree is built using the **ID3 algorithm**, where each attribute is used only as an internal node.

Relationship Status	Age	Education	Income	Usage
Single	>25	University	>50K	Low
In a relationship	≤25	College	≤50K	Medium
Single	>25	University	>50K	Low
Married	≤25	University	≤50K	High
Single	>25	University	>50K	Low
Married	≤25	College	≤50K	Medium
In a relationship	≤25	College	>50K	Medium
In a relationship	>25	High School	≤50K	Low
Married	>25	University	≤50K	High
Single	>25	High School	>50K	Low
In a relationship	≤25	College	>50K	Medium
In a relationship	>25	High School	≤50K	Low
Married	>25	University	≤50K	High
Single	>25	High School	>50K	Low
In a relationship	≤25	College	>50K	Medium

1. **[2 pts]** What is the initial entropy of Usage?

2. **[5 pts]** Which attribute should be chosen at the root of the tree? Show your calculation for the information gains (IG) and explain your choice in a sentence.

3. **[5 pts]** Draw the full decision tree for the above data.

1.2 Boosting [8 pts]

As we saw in the class, Adaptive Boosting (Adaboost) method is an "ensemble method" that learns a set of classifiers, as well as weights for each of them, and makes a final decision by a weighted voting of the classifiers. Every iteration, Adaboost updates the weights on training samples so that classifier can focus on learning samples that are difficult for classification. It reduces both bias and variance, thus is effective in reducing the prediction error.

Algorithm 1: Adaboost algorithm

Data: A set $\mathcal{S}$ of m labeled training samples: $\mathcal{S} = \{(x_i, y_i), i = 1, \cdots, m\}$, where $y_i \in \{-1, 1\}$ are labels, number of iterations T, a set of hypothesis $\mathcal{H} = \{h_j\}$ from a learning algorithm

Result: A set of weights $\mathcal{A} = \{\alpha_t, t = 1, \cdots, T\}$ and classifiers selected at each iteration $\{h_t, t = 1, \cdots, T\}$

Initialize the weights $D_1(i) = \frac{1}{m}$;

for $t = 1, \cdots, T$ **do**

- Select a weak classifier with smallest weighted error $h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j$, where $\epsilon_j = \sum_{i=1}^{m} D_t(i) \mathbb{1}_{\{y_i \neq h_j(x_i)\}}$, and let $\epsilon_t = \min_{h_j \in \mathcal{H}} \epsilon_j$;
- Compute the weight for the classifier $\alpha_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$;
- Update the weight $D_{t+1}(i) = D_t(i) \exp(-\alpha_t y_i h_t(x_i))$;
- Normalize the weight $D_{t+1}(i) = \frac{D_{t+1}(i)}{\sum_i D_{t+1}(i)}$;

end

Return weights α_t and classifiers h_t $t = 1, \cdots, T$.

In this problem, you will prove the error bound of the Adaboost algorithm. H denotes the strong classifier. The decision of the strong classifier on new sample x^* is given as $H(x^*) = sign(\sum_t \alpha_t h_t(x))$. "Weak" classifiers are defined as classifiers that do slightly better job than random guessing, $\epsilon_t = \frac{1}{2} - \gamma_t$, where $\gamma_t > 0$ is a small value.

The upper bound of the training error of the strong classifier H is given as

$$err(H) \leq \prod_t^T 2\sqrt{\epsilon_t(1-\epsilon)} \tag{1}$$

$$= \prod_t^T \sqrt{1 - 4\gamma_t^2} \tag{2}$$

$$\leq \exp\left(-2\sum_t^T \gamma_t^2\right) \tag{3}$$

Now you will prove this bound step-by-step through (1)-(4). For notational simplicity, let $f(x) = \sum_t \alpha_t h_t(x)$, so that $H(x) = sign(f(x))$. Let Z_t denote a normalizing constant for weak classifier t: $Z_t = \sum_i D_{t+1}(i)$

1. **[2 pts]** Show that $D_{T+1}(i) = \frac{1}{m} \frac{1}{\prod_t^T Z_t} \exp(-y_i f(x_i))$.

2. **[2 pts]** Show that error of the strong classifier H is upper bounded by the product of Z_t: $err(H) \leq \prod_t^T Z_t$.

   Hint 1: the error of the strong classifier H is given as: $err(H) = \frac{1}{m} \sum_i \mathbb{1}_{\{y_i \neq H(x_i)\}}$
   Hint 2: use the fact that 0-1 loss is upper bounded by exponential loss - recall that we are using 0-1 loss here by defining error as $\mathbb{1}_{\{y_i \neq H(x_i)\}}$.

3. **[2 pts]** Now show that $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$.

   Hint 1: start from the $Z_t = \sum_i D_{t+1}(i) = \sum_i D_t(i)\exp(-\alpha_t y_i h_t(x_i))$.
   Hint 2: separate the Z_t *expression for correctly classified cases and incorrectly classified cases.*
   Hint 3: express in terms of ϵ_t.
   Hint 4: plug in α_t

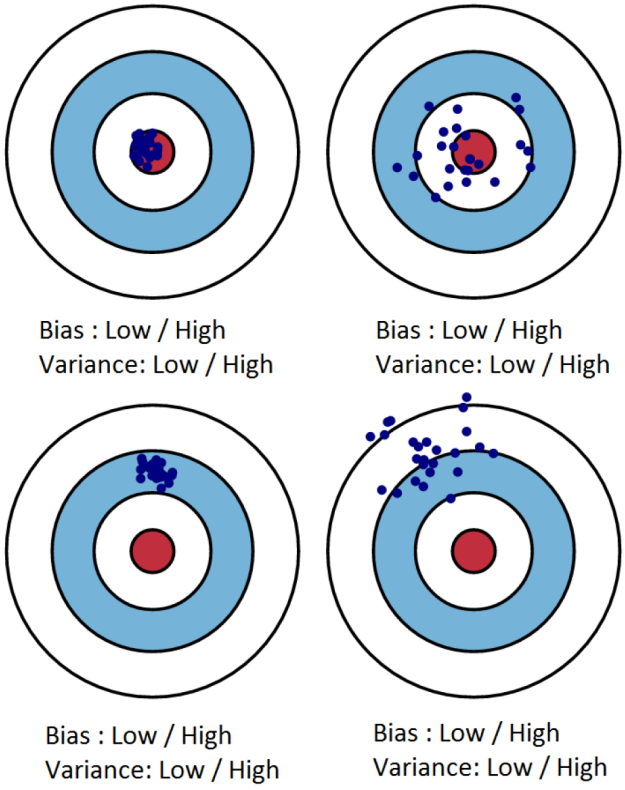
4. **[2 pts]** Combining (b) and (c), now we have upper bound on the error as $err(H) \leq \prod_t^T 2\sqrt{\epsilon_t(1-\epsilon_t)}$. Now show the following: $err(H) \leq \exp\left(-2\sum_t^T \gamma_t^2\right)$.

   Hint 1: use the definition of weak classifier$\epsilon_t = \frac{1}{2} - \gamma_t$, *and plug in* ϵ_t .
   Hint 2: use the fact that $1 - x \leq \exp^{-x}$

2 Model selection(20 pts) [Satya]

2.1 Bias-Variance Decomposition [15 Points]

1. To understand bias and variance, we will create a graphical visualization using a bulls-eye. Imagine that the center of the target is our true model (a model that perfectly predicts the correct values). As we move away from the bulls-eye, our predictions get worse and worse. Suppose we have a tuning parameter that controls the complexity of the model; and as part of model selection, we have to select a value for this tuning parameter. Given a fixed value for the tuning parameter, suppose we draw n training samples, and train a model with these n samples. Each such model corresponds to a hit on the target. Suppose we repeat this process multiple times, fixing the value of the tuning parameter. Sometimes these hits will all be close to each other, sometimes they will be farther from each other, and they might also differ in their distance from the bulls-eye (the true model). Consider these four different scatters of hits on the target for four distinct values of the tuning parameter. Characterize the bias and variance of the estimates of the following models on the data with respect to the true model as low or high by circling the appropriate entries below each diagram.



2. Explain what effect will the following operations have on the bias and variance of your model. Fill in one of 'increases', 'decreases' or 'no change' in each of the cells:

	Bias	Variance
Regularizing the weights in a linear/ logistic regression model		
Increasing k in k-nearest neighbor models		
Pruning a decision tree (to a certain depth for example)		
Increasing the number of hidden units in an artificial neural network		

2.2 Bayesian Model Selection [5 Points]

In this problem, we illustrate Bayesian model selection on probabilistic suffix trees (PSTs). A probabilistic suffix tree (PST) is a rooted tree, with edges corresponding to a specific value for a specific feature, similar to decision trees (we will consider only categorical features, which take a finite set of values). In contrast to decision trees however, here each node in the tree is associated with a predictive distribution $P(y|x_1, x_2, ..., x_i)$, where $x_1, x_2, ..., x_i$ are the feature values on the unique path from the root of the tree to that node; and to classify a datapoint represented by a sequence of features, $\mathbf{x} = (x^1, x^2, x^3, ...)$, we start from the root and follow the longest path of matching consecutive features, and assign to $\mathbf{x}$ a class according to the distribution of the last node reached. Note that we always stop at a node since if there is no match, then we simply stop at the root.

One of the problems is to estimate a PST for a binary classification task ($y \in \{0, 1\}$) where each feature can take values in {0, 1, 2} from a set of training points $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)$. This is a model selection problem because aside from estimating the probability distributions at each node, we also need to specify the structure of the tree (include the edge feature values). Consider a PST with K nodes, let $p_1, p_2, ..., p_K$ be the conditional probability of $y = 1$ associated with each node. The structure of the tree, including the edge feature values, represent the model, while the probabilities $\theta = (p_1, p_2, ..., p_K) \in [0, 1]^K$ constitute its parameters. One way to compare models is via their Bayesian scores (or marginal likelihood). We place a prior distribution over the model parameters; in this case we impose the "uninformative" prior $P(\theta) = 1 \ \forall \theta \in [0, 1]^K$, and integrate out the parameters:

$$P(y_1, ...y_n|\mathbf{x}_1, ...\mathbf{x}_n, \text{PST}) = \int_{[0,1]^K} \left[\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \theta)\right] P(\theta)d\theta$$

It is not always possible to evaluate the Bayesian score analytically but in our case it is. We need a bit of notation to get started:

1. $n_1, n_2, ..., n_K$ are the number of data points from the training set classified using each of the K nodes;
2. n_i^+ is the number of positive (y = 1) training samples ending up at node i;
3. n_i^- is the number of negative samples at node i $(y = 0)$.

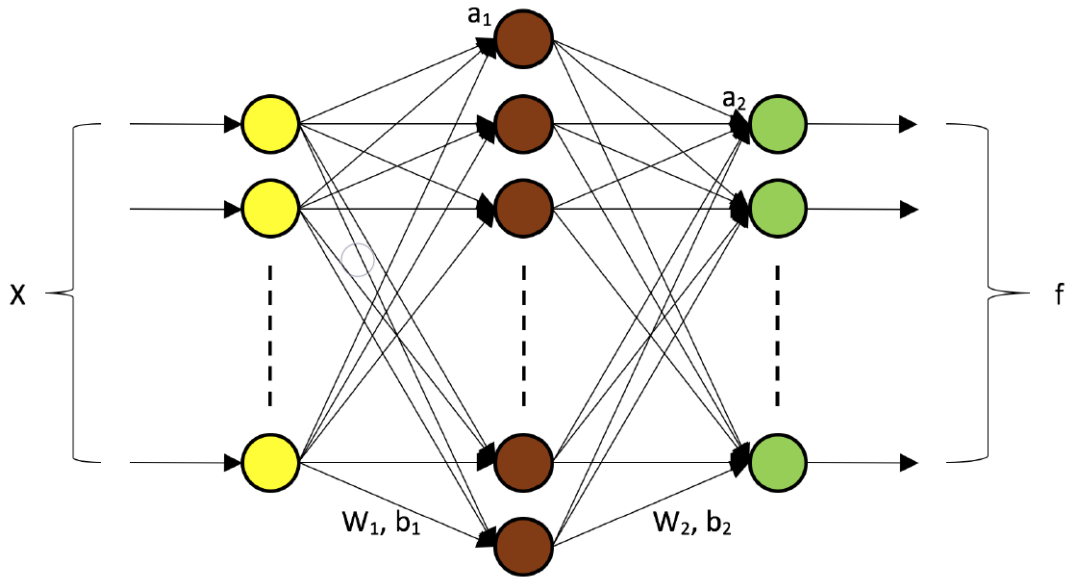
Note that $n = \sum_{i=1}^{K} n_i$ and $n_i = n_i^+ + n_i^-$. With this notation the likelihood of the training data can be written as:

$$\prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \theta) = \prod_{i=1}^{K} p_i^{n_i^+} (1 - p_i)^{n_i^-}.$$

Compute the Bayesian model selection score in terms of n_i, n_i^+ and n_i^-. Can you think of a drawback of computing this score? If yes, suggest a change?
Hint: $\int_{[0,1]} \theta^\alpha (1 - \theta)^\beta d\theta = B(\alpha, \beta)$, where B is the beta function.

3 Neural Networks (20 pts) [George and Sreena]

Consider the following two layer neural network architecture in the figure below. The specifications of the



network are as follows:

- X is the input vector of N examples each of k features.
- f is the output vector of shape $N \times h_2$.
- a_1, a_2 are the activations (outputs) of the first and second layer.
- W_1, W_2 are weight matrices of shape $k \times h_1$ and $h_1 \times h_2$ respectively.
- b_1, b_2 are bias vectors of shape $1 \times h_1$ and $1 \times h_2$ respectively.
- h_1 is the number of hidden units in layer 1 and h_2 is the number of hidden units in layer 2.

In this problem, we will consider neural networks constructed using the following two types of non-linear activation functions:

- **Sigmoid** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **tanh** $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$

Furthermore, we will use the Cross Entropy Loss given by:

$$Error(network) = -\sum_{i=1}^{h_2} y_i \log y_i',$$

where $y_i \in \{0, 1\}$ is the target label, and y_i' is the i^{th} predicted output of the network.

3.1 Reveal the Black box [15 points]

Derive the backpropagation algorithm with respect to the adjustable parameters of this network, using (a) **Sigmoid** $\sigma(x)$ as the activation function and (b) **tanh** as the activation function.
Note : consider softmax activation for layer 2 in both cases (a) and (b)

3.2 Neural Networks Meet Logistic Regression [5 points]

Recall that Logistic Regression models the conditional probability of a label $Y \in \{0, 1\}$ given p-dimensional input $X \in \mathbb{R}^p$ as:

$$P(Y = 0|X = x) = \frac{1}{1 + \exp(w^T x)}$$

and

$$\begin{aligned} P(Y = 1|X = x) &= 1 - P(Y = 0|X = x) \\ &= \frac{\exp\big(w^T x\big)}{1 + \exp(w^T x)}, \end{aligned}$$

where $w \in \mathbb{R}^p$ denotes a weight vector.

Using only sigmoid and linear activation functions, create a Neural Network that for a three-dimensional input $X \in \mathbb{R}^3$ behaves like an ensemble of two Logistic Regression classifiers. Note that a Linear Activation function has the form $L(x) = C\,(w^T\,x)$, where x is an input vector, w is a weight vector, and C is a constant; and by ensemble of classifiers here we simply mean a weighted linear combination of classifiers.

4 Multiple Choice Questions (10 pts) [Shaojie]

There might be more than one right answer. Please explain your choice in one or two sentences.

1. **[3 pts]** Which of the following statements is/are correct?
   (A) Logistic regression is equivalent to a neural network with one layer and sigmoid activation.
   (B) Compared to BIC, AIC is more likely to penalize overly complex models.
   (C) Say $f = \text{Adaboost}(\mathcal{H})$, where $\mathcal{H}$ is a family of weak classifiers (e.g. decision trees). Then the output predictor f has a decision boundary that is also in the class $\mathcal{H}$.
   (D) Using cross-validation to select the # of iterations can reduce the likelihood of overfitting in boosting.
   (E) SVM can only learn linear boundary in the input space, and it's hard to improve on this fact.
2. **[2 pts]** Which of the following statements is/are wrong?
   (A) All neural network architectures are discriminative models i.e. model the distribution of output given input.
   (B) For all L_p expected loss, we can decompose it into a bias term and a variance term.
   (C) In each iteration of Adaboost, the weights are increased only for data points that are misclassified.
   (D) A one-layer neural network cannot represent an XOR function.
3. **[3 pts]** Which of the following about deep neural networks (DNNs) is/are correct?
   (A) Recurrent neural networks (RNNs) are usually used for time-series (i.e., sequential) data.
   (B) Suppose A is a 2-layer neural network with sigmoid activation; i.e. $y = \mathbf{W}_1(\sigma(\mathbf{W}_0\mathbf{x} + \mathbf{b}_0)) + \mathbf{b}_1$. Then one can always obtain another equivalent network B (also 2 layers, with same # of parameters) that uses tanh activation instead of σ, but computes the same function as the original network A on all inputs $\mathbf{x}$.
   (C) Gradient descent (GD) is usually preferred over stochastic gradient descent (SGD) in training DNNs because theoretically, SGD has a slower convergence rate than GD.
   (D) Larger networks are more likely to overfit to data than smaller networks, if no regularization is applied to either.
4. **[2 pts]** Which of the following about entropy H, mutual information I, and decision trees is/are correct?
   (A) The depth of a decision tree learned from n training samples may be larger than n, but is always smaller than $2n$.
   (B) One can regularize a decision tree by pruning it.
   (C) For any random variables X, Y, we always have $H(X, Y) < H(X) + H(Y)$.
   (D) $I(X, Y) = 0$ if and only if X, Y are independent.

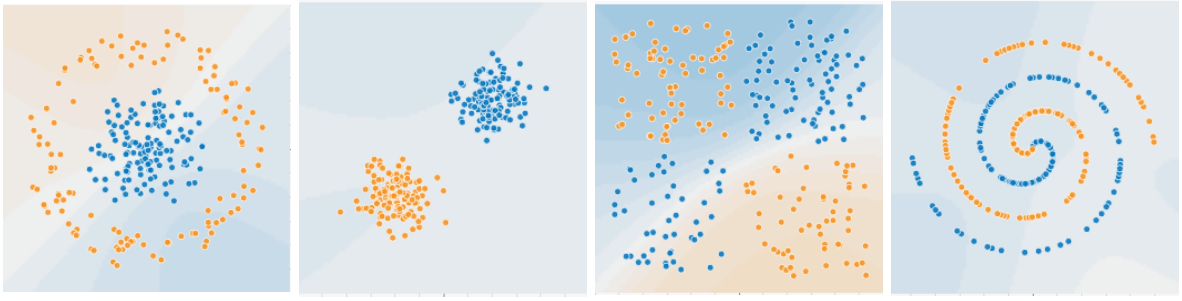
5 Programming Exercise (20 pts) [Adarsh and Wenhao]

Note: Your code for all of the programming exercises should also be submitted to Gradescope. In particular, there is a separate 'programming assignment' in Gradescope to which you should upload your code, while visualizations and written answers should still be submitted within the primary Gradescope assignment. In your code, **please use comments to point out primary functions that compute the answers to each question.**

Feel free to use any programming language, as long as your TAs can read your code. Turn in your code in a single zipped folder that might contain multiple source code files.

5.1 Tensor Flow Playground for Neural Networks [5 pts]

In this problem we will use the TensorFlow playground http://playground.tensorflow.org, which is a nice visual tool for training simple Multi Layer Perceptrons (MLPs). Your goal in this problem is to carefully select input features to design the "smallest" MLP classifiers that can achieve low test loss for each of the 4 datasets in TensorFlow playground (Figure 1 shows the 4 datasets available on TensorFlow playground). Note that you have to design a separate classifier for each individual dataset. Here smallest is defined as having least number of neurons in the network. By low test loss we mean a test loss < 0.1 for the swiss roll dataset and a test loss of ≈ 0 for the rest of the datasets. Submit screenshots after your networks achieve the required test loss for each of the datasets.



(a) Circles (b) Clusters (c) Squares (d) Swiss Roll

Figure 1: Datasets on TensorFlow playground.

5.2 Binary Classification [15pts]

In this problem you will be using a binary (two class) version of mnist dataset. The data and code template can be downloaded from:
`http://www.cs.cmu.edu/~adarshp/HWData/HW3_Data.zip`.
The data folder has `mnist2.mat` file which contains the train, test and validation datasets. The python folder has `python` code template, and matlab folder has the `matlab` code template, which you can use for your implementation. Note that you can use any language of your choice, but we've just provided templates for MATLAB and python.

We use the following formulation in this problem

$$\min_{w \in \mathbb{R}^d} \frac{\lambda}{2}\|w\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max(1 - y_i \langle w, X_i \rangle, 0).$$

This is only done to simplify calculations. You will optimize this objective using Stochastic Sub-Gradient

Descent (SSGD) (see [1]). This approach is very simple and scales well to large datasets[1]. In SSGD we randomly sample a training data point in each iteration and update the weight vector by taking a small step along the direction of negative "sub-gradient" of the loss[2]. The SSGD algorithm is given by

- Initialize the weight vector $w = 0$.
- For t = 1 ... T
  - * Choose $i_t \in \{1, \dots n\}$ uniformly at random
  - * Set $\eta_t = \frac{1}{\lambda t}$.
  - * If $y_{i_t} \langle w, X_{i_t} \rangle < 1$ then:
    - – Set $w \leftarrow (1 - \lambda \eta_t) w + \eta_t y_{i_t} X_{i_t}$
  - * Else:
    - – Set $w \leftarrow (1 - \lambda \eta_t) w$
- Return w

Note that we don't consider the bias/intercept term in this problem.

- Complete the `train(w0, Xtrain, ytrain, T, lambda)` function in the `svm.py` file (`matlab` users complete the `train.m` file).
  - – The function `train(w0, Xtrain, ytrain, T, lambda)` runs the SSGD algorithm, taking in an initial weight vector `w0`, matrix of covariates `Xtrain`, a vector of labels `ytrain`. `T` is the number of iterations of SSGD and `lambda` is the hyper-parameter in the objective. It outputs the learned weight vector `w`.
- Run `svm_run.py` to perform training and see the performance on training and test sets.

Evaluation

- Use `validation` dataset for picking a good `lambda(`λ`)` from the set {`1e3, 1e2, 1e1, 1, 0.1`}.
- Report the accuracy numbers on train and test datasets obtained using the best `lambda`, after running SSGD for 200 epochs (i.e., $T = 200 * n$). Generate the training accuracy vs. training time and test accuracy vs training time plots.

References

[1] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. Mathematical programming, 127(1):3–30, 2011.

[1] To estimate optimal w, one can also optimize the dual formulation of this problem. Some of the popular SVM solvers such as LIBSVM solve the dual problem. Other fast approaches for solving dual formulation on large datasets use dual coordinate descent.

[2] Sub-gradient generalizes the notion of gradient to non-differentiable functions