Machine Learning M146 UCLA Spring quarter 2017-2018
Prof. Suhas Diggavi Handout # 22, Friday, 1st June 2018

Homework Set #5 : GMM, HMM, Boosting, Neural Networks

Due 10th June 2018, Sunday, before 11:59 pm

Submission

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

Parts of this assignment are adapted from course material by Andrew Ng (Stanford), Jenna Wiens (UMich) and Jessica Wu (Harvey Mudd).

Problem 1 (Supervised and Unsupervised Gaussian Mixture Models [18 pts])

We now consider clustering 1D data using a **Gaussian Mixture Model**. We assume the number of components of the mixture (equivalently the number of clusters) $K = 2$. You are given three instances: $(x_1, x_2, x_3) = (1, 10, 20)$ where each $x_n \in \mathbb{R}, n \in \{1, 2, 3\}$.

(a) **Supervised Case**: Suppose we have complete data, i.e. $(z_1, z_2, z_3) = (1, 2, 2)$, then we can estimate the parameters of GMM using maximum likelihood estimation.

1. **(4 pts)** Write the log likelihood function $\log p(x, z|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{C})$ in terms of parameters $\boldsymbol{\pi} = [\pi_1, \pi_2]$, $\boldsymbol{\mu} = [\mu_1, \mu_2]$ and variance $\boldsymbol{C} = [C_1, C_2]$. You can leave the function in terms of $\mathcal{N}(\cdot)$.
2. **(4 pts)** Show the maximum likelihood estimator of $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{C}$.

(b) **Unsupervised case**: Now consider we don't have complete data, i.e. the label z are unknown. We will use the EM algorithm to maximize the likelihood. Suppose the output of the E-step is the following matrix:

$$\Gamma = \begin{pmatrix} 1 & 0 \\ 0.4 & 0.6 \\ 0 & 1 \end{pmatrix}$$

Here entry (n, k) of this matrix $\gamma_{n,k}$ is the posterior probability that instance n belongs to mixture component k. For this question, you can leave your final answer in the form $\frac{a}{b}$.

1. **(5 pts)** Show the M-step update for the mixing weights π_1, π_2.
2. **(5 pts)** Show the M-step update for the means μ_1, μ_2.

Problem 2 (Hidden Markov Model [16 pts])

Consider a Hidden Markov Model with two hidden states, $\{1, 2\}$, and two possible output symbols, $\{A, B\}$. The initial state probabilities are

$$\pi_1 = P(q_1 = 1) = 0.49 \quad \text{and} \quad \pi_2 = P(q_1 = 2) = 0.51,$$

the state transition probabilities are

$$q_{11} = P(q_{t+1} = 1|q_t = 1) = 1 \quad \text{and} \quad q_{12} = P(q_{t+1} = 1|q_t = 2) = 1,$$

and the output probabilities are

$$e_1(A) = P(O_t = A|q_t = 1) = 0.99 \quad \text{and} \quad e_2(B) = P(O_t = B|q_t = 2) = 0.51.$$

Throughout this problem, make sure to show your work to receive full credit.

(a) **(4 pts)** There are two unspecified transition probabilities and two unspecified output probabilities. What are the missing probabilities, and what are their values?

(b) **(6 pts)** What is the most frequent output symbol (A or B) to appear in the first position of sequences generated from this HMM?

(c) **(6 pts)** What is the sequence of three output symbols that has the highest probability of being generated from this HMM model?

Problem 3 (Ensemble Methods [32 pts])

The code provided to you earlier fits an AdaBoosted decision stump on a non-linearly separable classification dataset composed of two Gaussian quantiles clusters. In this assignment, you will work with the handwritten digits dataset which has 1797 samples total, each having 64 features (8×8 grayscale images) in 0-16, and labelled under 10 classes which are integers between 0-9.

Download `adaboost.py`. We will walk through training multiclass classifiers first using the libraries for Bootstrap Aggregation, followed by using the AdaBoost algorithm provided to you, with decision trees of three levels as the weak learner. We will make use of the SAMME algorithm[1] which is a generalization of adaptive boosting you have seen in lectures to multi-class classification.

(a) **(0 pts)** Import the digits dataset from sklearn and split the data into train (90%) and test set (10%).

(b) **(4 pts) Baseline:** On the entire dataset of 1797 samples train a baseline decision tree classifier model using entropy criterion and report the cross-validation error.

(c) **(4 pts) Bagging:** In this exercise we will train a bagging classifier using decision trees with entropy criterion.

1. Refer code to observe how a model is instantiated. Note that `BaggingClassifier` uses 10 estimators by default. We will use 10 estimators for this homework, but to modify the number, refer documentation.[2] Also, max_features = None implies all features will be considered for split at every node.
2. Use cross_val_score to evaluate the model for cv=10. Use X_train, y_train for this purpose and report the cross-validation error.

(d) **(8 pts) Bagging (Random forest):** In this exercise we will train a random forest classifier which is a special bagging classifier with a subset of features selected at random during training at every node of a decision tree. The number of features to randomly sample from at every node is a hyperparameter and must be tuned.

1. Refer code to observe how a random forest classifier is instantiated by setting max_features of the decision tree to a positive integer.
2. Perform 10 fold cross validation to tune the hyperparameter i.e. max_features which can range from 1 to 64. Use training set for doing so and plot a graph of cross-validation error vs max_features and pick an appropriate number. Report the optimal hyperparameter, and the corresponding cross validation error.

[1] SAMME reference
[2] BaggingClassifier

(e) **(8 pts)Boosting:** In this exercise we will train a boosted classifier using decision trees with three levels as weak learner.

1. Split the X_train into training set (90%) and validation set (10%).
2. Plot the training error and the validation error or misclassification rate of the boosted model against the number of iterations of the algorithm. (# iter= [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 1536, 2048]). Report the number of iterations with lowest validation error along with the error.

(f) **(8 pts)** Evaluate the performance of the the bagging classifier, the random forest with the appropriate hyperparameter and the boosted decision tree with appropriate number of iterations that you have trained by retraining all the models on the entire training set i.e. (X_train, y_train) and evaluate the performance using (X_test, y_test). Comment on their relative performances in this scenario.

Problem 4 (Multi-Class Classification Using Neural Network [34 pts])

In this section we'll walk through a complete Numpy implementation of a toy Neural Network in 2 dimensions. We'll first implement a simple linear classifier and then extend the code to a 2-layer Neural Network. As we'll see, this extension is surprisingly simple and very few changes are necessary.

A generic neural network is literally a combination of multiple layers with trainable parameters, serving different functionality. In this project you will see the explicit forward and backward propagation implementation of layers, and you will use the code provided to train the network.

Download the code and data sets from the course website. It contains a python file named `neuralnet.py`. The file contains complete implementation for the classifiers you need in the project. In the next section we will skim through the code together to understand its structure.

Visualize the Dataset

(a) **(4 pts)** Run the first part of the code (`neuralnet.py`) to plot the spiral dataset generated to train the classifier. Briefly describe the properties of the dataset, including dimensionalities, number of classes, and linearly separability.

Note: In neural network, normally we would want to pre-process the dataset so that each feature has zero mean and unit standard deviation, but in this case the features are already in a nice range from -1 to 1, so we skip this step.

Training a Multi-Class Softmax Linear Classifier

One of the advantages of neural network is that it can easily extend to multi-class classification. In multi-class classification problem, the dimension of the output matrix $\boldsymbol{Y}$ will have shape (N, K), where N is the number of samples, and K the number of classes. More specifically, k-th column of the output matrix $\boldsymbol{Y}$ corresponds to the scores (or probabilities) of class k. And we will denote vector $\boldsymbol{y}$ as one row of matrix $\boldsymbol{Y}$, which is the output scores of a example.

Lets first train a Softmax Linear Classifier on this classification dataset. The implementation of the class are under `Class LinearClassifier`.

(b) **(6 pts)** In this project we will use the *softmax* function, or *normalized exponential function.* Softmax function is a generalization of the logistic function that "squashes" a K-dimensional vector $\boldsymbol{s}$ of arbitrary real values to a K-dimensional vector $\sigma(\boldsymbol{s})$ [3] of real values, where each entry is in the range $(0, 1)$, and all the entries adds up to 1 (so we also regard it as a discretized probability for classes) . The function is given by

$$y_k \doteq \sigma(\boldsymbol{s})_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}} \text{ for } k = 1, \cdots, K.$$

where $\sigma(\boldsymbol{s})_k$ is the k-th entry of the vector $\sigma(\boldsymbol{s})$. Show that when $K = 2$, the softmax function is Sigmoid function.
Hint: Recall that Sigmoid function is $\sigma(s) = \frac{1}{1+e^{-s}}$ and you can set $s = s_0 - s_1$.

(c) **(6 pts)** The softmax function introduced above will provide probabilities corresponding to classes that can be used for prediction. In neural network, one of the key ingredients we need is a *loss function*, which is a differentiable objective that quantifies our "unhappiness" with the computed class scores. Intuitively, we want the correct class to have a higher score than the other classes. When this is the case, the loss should be low and otherwise the loss should be high. In this example we will use the *cross-entropy loss* that is associated with the Softmax classifier. Again, suppose $\boldsymbol{s}$ as the class scores for a single example, then the softmax cross-entropy loss for that example is

$$L_{CE} = -\sum_{k=1}^{K} t_k \log(y_k)$$

where $\boldsymbol{t}$ is a indicator vector with K dimensions. $t_k = 1$ when the correct label is k, 0 otherwise. Show that

$$\frac{\partial L_{CE}}{\partial s_k} = y_k - t_k$$

Hint: You may find the following useful and you don't need to prove it before using it. The partial derivative of softmax function with respect to one of its input scores is

$$\frac{\partial y_j}{\partial s_k} = \begin{cases} y_j(1 - y_j) & j = k \\ -y_j y_k & j \neq k \end{cases}$$

[3]Here we denote $\boldsymbol{s}$ as the input notation of softmax function. In practice, you can think of $\boldsymbol{s}$ as the class scores for a single example (e.g. $\boldsymbol{s} = \boldsymbol{w}^T\boldsymbol{x} + b$ as a array of 3 numbers here.)

(d) **(6 pts)** The full cross-entropy loss is then defined as the average cross-entropy loss over the training examples and the regularization:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_{CE,i} + \frac{1}{2}\lambda \sum_{l=1}^{L} ||W_l||_2^2$$

where l is the index of layers. Run the code to train the softmax linear classifier using function `fit`. Plot the learning curve (i.e. the training loss v.s. iterations), report the training accuracy, and use the function `plot_classifier` to plot the decision boundary of the classifier.

Training a Neural Network

Clearly, a linear classifier is inadequate for this dataset and we would like to use a Neural Network. Neural network with one additional hidden layer will suffice for this toy data. We will now need two sets of weights and biases (for the first and second layers). The classifier is implemented under `Class NeuralNet`.

(e) **(6 pts)** Run the code to train the neural network using function `fit`. Plot the learning curve, report the training accuracy, and use the function `plot_classifier` to plot the decision boundary of the classifier.

(f) **(6 pts)** Compare the training error with linear classifier implemented in the previous session. Briefly Explain.