COL 774: Machine Learning. Assignment 1

Due Date: 11:50 pm, Friday Sep 1, 2023. Total Points: 80

Notes:

- You should submit all your code as well as any graphs that you might plot. Do not submit answers to theoretical questions.
- Do not submit the datasets.
- Include a **single write-up (pdf) file** which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- You should use Python for all your programming solutions.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, an addtional penalty of the negative of the total weightage of the assignment and possibly much stricter penalties (including a **fail grade** and/or referring to a **DisCo**).
- Many of the problems below have been adapted from the Machine Learning course offered by Andrew Ng at Stanford.
- You should normalize the data (x's) to have zero mean and unit variance in each dimension for Q1, Q3 and Q4, as described in class. Do Not perform any normalization for Q2.

1. **(20 points) Linear Regression**
   In this problem, we will implement least squares linear regression to predict density of wine based on its acidity. Recall that the error metric for least squares is given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right)^2$$

   where $h_\theta(x) = \theta^T x$ and all the symbols are as discussed in the class. The files linearX.csv and linearY.csv contain the acidity of the wine ($x^{(i)}$'s, $x^{(i)} \in \mathcal{R}$) and its density ($y^{(i)}$'s, $y^{(i)} \in \mathcal{R}$), respectively, with one training example per row. We will implement least squares linear regression to learn the relationship between $x^{(i)}$'s and $y^{(i)}$'s.

   (a) (**8 points**) Implement batch gradient descent method for optimizing $J(\theta)$. Choose an appropriate learning rate and the stopping criteria (as a function of the change in the value of $J(\theta)$). You can initialize the parameters as $\theta = \vec{0}$ (the vector of all zeros). Do not forget to include the intercept term. Report your learning rate, stopping criteria and the final set of parameters obtained by your algorithm.

(b) (**3 points**) Plot the data on a two-dimensional graph and plot the hypothesis function learned by your algorithm in the previous part.

(c) (**3 points**) Draw a 3-dimensional mesh showing the error function ($J(\theta)$) on z-axis and the parameters in the $x-y$ plane. Display the error value using the current set of parameters at each iteration of the gradient descent. Include a time gap of 0.2 seconds in your display for each iteration so that the change in the function value can be observed by the human eye.

(d) (**3 points**) Repeat the part above for drawing the contours of the error function at each iteration of the gradient descent. Once again, chose a time gap of 0.2 seconds so that the change be perceived by the human eye.(Note here plot will be 2-D)

(e) (**3 points**) Repeat the part above (i.e. draw the contours at each learning iteration) for the step size values of $\eta = \{0.001, 0.025, 0.1\}$. What do you observe? Comment.

2. **(20 points) Sampling and Stochastic Gradient Descent**

In this problem, we will introduce the idea of sampling by adding Gaussian noise to the prediction of a hypothesis and generate synthetic training data. Consider a given hypothesis h_θ (*i.e.* known $\theta_0, \theta_1, \theta_2$) for a data point $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$. Note that $x_0 = 1$ is the intercept term.

$$y = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Adding Gaussian noise, equation becomes

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$

To gain deeper understanding behind Stochastic Gradient Descent (SGD), we will use the SGD algorithm to learn the original hypothesis from the data generated using sampling, for varying batch sizes. We will implement the version where we make a complete pass through the data in a round robin fashion (after initially shuffling the examples). If there are r examples in each batch, then there is a total of $\frac{m}{r}$ batches assuming m training examples. For the batch number b ($1 \leq b \leq \frac{m}{r}$), the set of examples is given as: $\{x^{(i_1)}, x^{(i_2)}, \cdots, x^{(i_r)}\}$ where $i_k = (b-1)r + k$. The Loss function computed over these r examples is given as:

$$J_b(\theta) = \frac{1}{2k} \sum_{k=1}^{r} \left(y^{(i_k)} - h_\theta(x^{(i_k)})\right)^2$$

(a) **(4 points)** Sample 1 million data points taking values of $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}$, $x_1 \sim \mathcal{N}(3, 4)$ and $x_2 \sim \mathcal{N}(-1, 4)$ independently, and noise variance in y, $\sigma^2 = 2$.

(b) **(6 points)** Implement Stochastic gradient descent method for optimizing $J(\theta)$. Relearn $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$ using sampled data points of part a) keeping everything same except the batch size. Keep $\eta = 0.001$ and initialize $\forall j\ \theta_j = 0$. Report the θ learned each time separately for values of batch size $r = \{1, 100, 10000, 1000000\}$. Carefully decide your convergence criteria in each case. Make sure to watch the online video posted on the course website for deciding the convergence of SGD algorithm.

(c) **(6 points)** Do different algorithms in the part above (for varying values of r) converge to the same parameter values? How much different are these from the parameters of the original hypothesis from which the data was generated? Comment on the relative speed of convergence and also on number of iterations in each case. Next, for each of learned models above, report the error on a new test data of 10,000 samples provided in the file named q2test.csv. Note that this test set was generated using the same sampling procedure as described in part (a) above. Also, compute the test error with respect to the prediction of the original hypothesis, and compare with the error obtained using learned hypothesis in each case. Comment.

(d) **(4 points)** In the 3 dimensional parameter space(θ_j on each axis), plot the movement of θ as the parameters are updated (until convergence) for varying batch sizes. How does the (shape of) movement compare in each case? Does it make intuitive sense? Argue.

3. **(15 points) Logistic Regression**
Consider the log-likelihood function for logistic regression:

$$L(\theta) = \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

For the following, you will need to calculate the value of the Hessian H of the above function.

(a) **(10 points)** The files logisticX.csv and logisticY.csv contain the inputs ($x^{(i)} \in R^2$) and outputs ($y^{(i)} \in \{0, 1\}$) respectively for a binary classification problem, with one training example per row. Implement[1] Newton's method for optimizing $L(\theta)$, and apply it to fit a logistic regression model to the data. Initialize Newton's method with $\theta = \vec{0}$ (the vector of all zeros). What are the coefficients θ resulting from your fit? (Remember to include the intercept term.)

(b) **(5 points)** Plot the training data (your axes should be x_1 and x_2 , corresponding to the two coordinates of the inputs, and you should use a different symbol for each point plotted to indicate whether that example had label 1 or 0). Also plot on the same figure the decision boundary fit by logistic regression. (i.e., this should be a straight line showing the boundary separating the region where $h(x) > 0.5$ from where $h(x) \leq 0.5$.)

4. **(25 points) Gaussian Discrmimant Analysis**
In this problem, we will implement GDA for separating out salmons from Alaska and Canada. Each salmon is represented by two attributes x_1 and x_2 depicting growth ring diameters in 1) fresh water, 2) marine water, respectively. File q4x.dat stores the two attribute values with one entry on each row. File q4y.dat contains the target values ($y^{(i)}$'s $\in$ {Alaska, Canada}) on respective rows.

(a) **(6 points)** Implement Gaussian Discriminant Analysis using the closed form equations described in class. Assume that both the classes have the same co-variance matrix i.e. $\Sigma_0 = \Sigma_1 = \Sigma$. Report the values of the means, μ_0 and μ_1, and the co-variance matrix Σ.

(b) **(2 points)** Plot the training data corresponding to the two coordinates of the input features, and you should use a different symbol for each point plotted to indicate whether that example had label Canada or Alaska.

(c) **(3 points)** Describe the equation of the boundary separating the two regions in terms of the parameters μ_0, μ_1 and Σ. Recall that GDA results in a linear separator when the two classes have identical co-variance matrix. Along with the data points plotted in the part above, plot (on the same figure) decision boundary fit by GDA.

(d) **(6 points)** In general, GDA allows each of the target classes to have its own covariance matrix. This results (in general) results in a quadratic boundary separating the two class regions. In this case, the maximum-likelihood estimate of the co-variance matrix Σ_0 can be derived using the equation:

$$\Sigma_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}} \tag{1}$$

And similarly, for Σ_1. The expressions for the means remain the same as before. Implement GDA for the above problem in this more general setting. Report the values of the parameter estimates i.e. μ_0, μ_1, Σ_0, Σ_1.

(e) **(5 points)** Describe the equation for the quadratic boundary separating the two regions in terms of the parameters μ_0, μ_1 and Σ_0, Σ_1. On the graph plotted earlier displaying the data points and the linear separating boundary, also plot the quadratic boundary obtained in the previous step.

(f) **(3 points)** Carefully analyze the linear as well as the quadratic boundaries obtained. Comment on your observations.

[1] Write your own version, and do not call a built-in library function.

COL 774: Assignment 2 (Semester I, 2023-24)

Due Date: 11:50 pm, Wednesday Oct 4, 2023. Total Points: 64

Notes:

- This assignment has two main parts - Text Classification using Naïve Bayes (Part 1), and Image Classification using SVMs (Part II).
- You should submit all your code (including any pre-processing scripts written by you) and any graphs that you might plot.
- Do not submit the datasets. Do not submit any code that we will be providing to you for processing.
- Include a single write-up (pdf) file which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- You should use Python for all your programming solutions.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a fail grade and/or a DISCO).

1. **(30 points) Text Classification**
   In this problem, we will use the Naïve Bayes algorithm for text classification. The dataset for this problem is the Coronavirus tweets Dataset. Given a tweet related to coronavirus, the task is to predict the sentiment of the review. There are three possible sentiments (labels) - positive, neutral or negative. You have been provided with a subset of the Coronavirus tweets Dataset, with the training and validation splits containing 37,864 reviews (samples) and 32,93 reviews respectively. We will be testing your code on a held-out test split. Data is available at this link.

(a) (10 points) Implement the Naïve Bayes Multiclass classification algorithm to classify each sample into one of the given three categories. You should implement the model where for each word position in a document, we generate the word using a single (fixed across word positions) Multinoulli distribution.

i. Report the accuracy over the training as well as the validation set.

ii. Read about word cloud. Construct a word cloud representing the most frequent words for each class.

Notes:

- Make sure to use the Laplace smoothing for Naïve Bayes (as discussed in class) to avoid any zero probabilities.
- You should implement your algorithm using logarithms to avoid underflow issues.
- You should implement Naïve Bayes from the first principles and not use any existing Python modules.

(b) (2 points)

i. What is the validation set accuracy that you would obtain by randomly guessing one of the categories as the target class for each of the reviews (random prediction)?

ii. What accuracy would you obtain if you simply predicted each sample as positive?

iii. How much improvement does your algorithm give over the random/positive baseline?

(c) (2 points) Read about the confusion matrix.

i. Draw the confusion matrix for your results in parts (a) and (b) above (for both training and validation data).

ii. For each confusion matrix, which category has the highest value of the diagonal entry? What does that mean?

(d) (4 points) The dataset provided to you is in the raw format i.e., it has all the words appearing in the original set of articles. This includes words such as 'of', 'the', 'and' etc. (called stopwords). Presumably, these words may not be relevant for classification. In fact, their presence can sometimes hurt the performance of the classifier by introducing noise in the data. Similarly, the raw data treats different forms of the same word separately, e.g., 'eating' and 'eat' would be treated as separate words. Merging such variations into a single word is called stemming. Read about stopword removal and stemming (for text classification) online.

i. Perform stemming and remove the stop-words in the training as well as the validation data.

ii. Construct word clouds for both classes on the transformed data.

iii. Learn a new model on the transformed data. Report the validation set accuracy.

iv. How does your accuracy change over the validation set? Comment on your observations.

(e) (6 points) Feature engineering is an essential component of Machine Learning. It refers to the process of manipulating existing features/constructing new features in order to help improve the overall accuracy of the prediction task. For example, instead of using each word as a feature, you may treat bi-grams (two consecutive words) as a feature.

i. You can read here about Bi-grams. Use word-based bi-grams to construct new features. You should construct these features after doing the pre-processing described

in part d(i) above. Further, these should be added as additional features, on top of existing unigram (single word) based features. Learn your model again, and report validation set accuracy.

ii. Come up with at least one additional set of features to further enhance your model. Learn the model after doing pre-processing as described in part d(i) above, and report the validation set accuracy.

iii. For both your variations tried above, compare with the validation set accuracy that you obtained in parts (a) and parts (d). Do the additional set of features constructed in each case help you improve the overall accuracy? Comment on your observations.

(f) (6 points) In Domain Adaptation, given a model trained on data from the *source domain*, we apply it on data from the *target domain*. The idea is to exploit the similarity between source and target domains, and leverage the parameters learned on the target domain, making up for lack of sufficient training data in the target domain. In our setting, we will assume the source domain is coronavirus tweets, and the target domain is represented by another dataset of general purpose tweets. You are given subsets of different sizes of the training data (in the target domain), varying from $1\%, 2\%, 5\%, 10\%, 25\%, 50\%, 100\%$ of the total training set.

i. Learn a Naïve bayes model, on entire source training data, combined with the partial target training data for each of the splits above. Your vocabulary should be constructed on the combined dataset. You should use the model in part (d) above, i.e., after removing stopwords, and performing stemming but before doing any additional feature engineering. Compute (target) validation set accuracy for each split.

ii. Do the same thing as in the part above, but without adding any data from source domain. This represents learning from scratch on the target data. Compute the accuracy on target validation set for each of the splits.

iii. Plot on a single graph the validation set accuracy for the two algorithms described above, as we vary the size of target training data.

iv. What do you observe? Explain.

2. **Image Classification** In this problem, we will use Support Vector Machines (SVMs) to build a binary image classifier. We will be solving the SVM optimization problem using a general-purpose convex optimization package as well as using a scikit-learn library function based on a customized solver known as LIBSVM. This problem is based on Intel image classification dataset. The original dataset has 25000 images belonging to 6 classes. However, we will be using a subset of this, containing 2380 training samples and 200 validation samples each for 6 classes. We will be testing your code on held-out test set. Data is available at this link. There is one folder each for images in one of the 6 classes. Each sample is an RGB image of size $150 \times 150 \times 3$. Resize the images to size $16 \times 16 \times 3$ and normalize the data by dividing each pixel value by 255. For working with SVM's, you should flatten each image to create a vector of length 768.

(20 points) Binary Classification: Let d be the last digit of your entry number. Take the subset of images for the classes (d mod 6) and (($d + 1$) mod 6) from the train data provided to you and perform the following experiments in the context of binary classification.

(a) (8 points) Download and install the CVXOPT package. Express the SVM dual problem (with a linear kernel) in a form that the CVXOPT package can take. You will have to think about how to express the SVM dual objective in the form $\alpha^T P\alpha + q^T\alpha + c$ matrix

where P is an $m \times m$ matrix (m being the number of training examples), q is an m-sized column vector and c is a constant. For your optimization problem, remember to use the constraints on α_i 's in the dual. Use the SVM formulation which can handle noise and use $C = 1.0$ (i.e. C in the expression $\frac{1}{2}w^T w + C * \sum_i \xi_i$). You can refer this link to get a working overview of cvxopt module and it's formulation.

i. How many support vectors do you get in this case? What percentage of training samples constitute the support vectors?

ii. Calculate the weight vector w and the intercept term b and classify each of the examples in the validation file into one of the two labels. Report the validation set accuracy. You will need to carefully think about how to represent w and b in this case.

iii. Reshape the support vectors corresponding to the top- 6 coefficients to get images of $16 \times 16 \times 3$ and plot these. Similarly, reshape and plot the weight vector w.

(b) (6 points) Again use the CVXOPT package to solve the dual SVM problem using a Gaussian kernel. Think about how the P matrix will be represented. Use $C = 1.0$ and $\gamma = 0.001$ (i.e. γ in $K(x, z) = \exp^{-\gamma * \|x-z\|^2}$) for this part.

i. How many support vectors do you get in this case as compared to the linear case above? How many support vectors obtained here match with the linear case above?

ii. Note that you may not be able to explicitly store the weight vector (w) or the intercept term (b) in this case. Use your learned model to classify the validation examples and report the validation accuracy.

iii. Reshape the support vectors corresponding to the top- 6 coefficients to get images of $16 \times 16 \times 3$ and plot these.

iv. Compare the validation accuracy obtained here with part (a).

(c) (6 points) Repeat parts -(a) & (b) with the scikit-learn SVM function, which is based on the LIBSVM package.

i. Compare the nSV (Number of Support Vectors) obtained here with part (a) for linear kernel and part (b) for Gaussian kernel. How many of the support vectors obtained here match with the support vectors obtained in both these cases?

ii. Compare weight (w), bias (b) obtained here with part (a) for linear kernel.

iii. Report the validation accuracy for both linear and Gaussian kernel.

iv. Compare the computational cost (training time) of the CVXOPT with the sklearn implementation in both the linear and Gaussian case.

(d) **(Extra fun! No Credits)** Resize the original $150 \times 150 \times 3$ sized image to size $32 \times 32 \times 3$ and normalize the data by dividing each pixel value by 255. Then flatten each image to create a vector of length 3072. Repeat part (c) on this and compare the results with that of part (c). What do you observe?

(14 points) Multi-Class Image Classification: In this section, we will work with the entire subset of the data provided to you in this question focusing on a multi-class classification problem using SVMs. We will work with Gaussian kernel for this section.

(a) (4 points) In class, we described the SVM formulation for a binary classification problem. In order to extend this to the multi-class setting, we train a model on each pair of classes to get $\begin{pmatrix} k \\ 2 \end{pmatrix}$ classifiers, k being the number of classes (here, k = 5). During prediction

time, we output the class which has the maximum number of votes from all the $\binom{k}{2}$ classifiers. You can read more about one-vs-one classifier setting at the following link. Using your CVXOPT solver from the previous section, implement one-vs-one multi-class SVM. Use a Gaussian Kernel with $C = 1.0$ and $\gamma = 0.001$.

i. Classify the validation examples and report validation set accuracy. In the case of ties, choose the label with the highest score.

(b) (3 points) Now train a multi-class SVM on this dataset using the scikit-learn SVM function, which is based on the LIBSVM package. Repeat part (a) using a Gaussian kernel with $\gamma = 0.001$. Use $C = 1.0$ as earlier.

i. Classify the validation examples and report validation set accuracy.

ii. How do the validation set accuracy and the training time obtained here compare with part (a) above?

(c) (4 points) Draw the confusion matrix for both of the above parts (2(a) and 2(b)). What do you observe? Which classes are miss-classified into which ones most often? Visualize (and report) 12 examples of miss-classified objects. Do the results make sense? Comment.

(d) (3 points) Validation set is typically used to estimate the best value of the model hyper-parameters (e.g., C in our problem with the Gaussian kernel) by randomly selecting a small subset of the training data as the validation set and then training on the modified training data (original training data minus the validation set) and making predictions on the validation set. For a detailed introduction, you can refer to this video. You can check the correctness of your intuition by trying this test. K-fold cross-validation is another such technique in this regard that we use in practice. In this technique, we divide our training data into K-folds or parts and then treat each part as our validation set once and train on the remaining K-1 parts. You can read more about cross-validation here [1] (see Section 1) for more details. This process is repeated for a range of model hyper-parameter values and the parameters which give best K-fold cross-validation accuracy are reported as the best hyper-parameters. We will use scikit-learn SVM function for this part.

For this problem, we will do a 5-fold cross-validation to estimate the best value of the C parameter for the Gaussian kernel case. Validation data should not be touched.

i. Fix γ as 0.001 and vary the value of C in the set $\{10^{-5}, 10^{-3}, 1, 5, 10\}$ and compute the 5 -fold cross-validation accuracy and the validation accuracy for each value of C.

ii. Now, plot both the 5-fold cross-validation accuracy as well as the validation set accuracy on a graph as you vary the value of C on x-axis (you may use log scale on x-axis). What do you observe? Which value of C gives the best 5 -fold cross-validation accuracy? Does this value of the C also give the best validation set accuracy? Comment on your observations.

[1] These are from Andrew Ng notes posted on the course website, and the link is available only from the internal IIT Delhi network. Feel free to read additional material online about this topic.

COL 774: Assignment 3
Semester I, 2023-24

[Part A: Decision Trees]. Total Points: 44
[Part B: Neural Networks]. Total Points: 38
Due Date (for both parts): Tuesday Oct 31st, 11:50 pm.

Notes:

- This assignment has two implementation questions.
- You should submit all your code (including any pre-processing scripts written by you) and any graphs that you might plot.
- Do not submit the datasets. Do not submit any code that we have provided to you for processing.
- Include **a write-up (pdf) file**, one (consolidated) for each part, which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file (one for each of the parts A and B).
- You should use Python as your programming language. ~~For Decision tree question (only), we may allow using C/C++/Java - but you need to confirm with us first.~~
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment is supposed to be done individually. You should carry out all the implementation by yourself.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).

1. **(44 points) Decision Trees (and Random Forests):**
   Imagine you've been hired by the BCCI as a data analyst due to your machine-learning expertise. Your initial assignment is to construct a decision tree-based classifier(s) to forecast the result of a white ball cricket match using various attributes obtained after the first innings is over, e.g., score, as well as meta-information about the match, such us countries playing, who won the toss etc. Your work will revolve around data obtained from the StatsGuru repository which has been scraped for this purpose. You've been provided with predefined sets of test, train, and validation datasets available for download from the course website. To gain a comprehensive understanding of the dataset, be sure to read the detailed information in the accompanying readme file. Please refrain from making the data publicly accessible and solely utilize it for the purposes of this assignment.
   You've also been given a starter code that facilitates file reading and basic tree data structure definition. However, it's worth noting that you have the flexibility to create your code from scratch, as the use of the starter code is not obligatory. Your primary task is to implement the decision tree algorithm for predicting cricket match outcomes based on a variety of features. Additionally, you'll explore the application of Random Forests in the later stages of this problem.

(a) **(15 points) Decision Tree Construction** Construct a decision tree using the given data to predict whether the team will win or lose the match. You should use mutual information as the criteria for selecting the attribute to split on. At each node, you should select the attribute which results in the maximum decrease in the entropy of the class variable (i.e. has the highest mutual information with respect to the class variable). Remember some attributes are categorical and some are continuous. For handling continuous attributes, you should use the following procedure. At any given internal node of the tree, a numerical attribute is considered for a two-way split by calculating the median attribute value from the data instances coming to that node, and then computing the information gain if the data was split based on whether the numerical value of the attribute is greater than the median or not. For example, if you have 10 instances coming to a node, with values of an attribute being (0,0,0,1,1,2,2,2,3,4) in the 10 instances, then we will split on value 1 of the attribute (median). Note that in this setting, a numerical attribute can be considered for splitting multiple times. At any step, choose the attribute which results in the highest mutual information by splitting on its median value as described above.

For categorical attributes, you have to perform k way split where k is a number of unique attribute values. For example, if the following instance of training data arrives at a node ($India, India, England, Kenya, Australia, England$) then splitting on this attribute will result in 4 children nodes. While testing if an example coming at this node have a different attribute value (For eg: $Pakistan$) then consider the node as a leaf node and return the prediction.

Experiment with different maximum depths specifically for $\{5, 10, 15, 20, 25\}$ and report the accuracy on train and test datasets. Feel free to experiment with other depths for performance gain. Plot the train and test set accuracies against the maximum depth in the tree. On the X-axis you should plot the maximum depth in the tree and the Y-axis should represent the train and test accuracies. Comment on your observations. Also, report the test accuracies for only win prediction and only loss prediction. Comment on your observation.

(b) **(5 points) Decision Tree One Hot Encoding:** Categorical attributes having more than 2 categories can be encoded using one hot encoding. In one hot encoding, each category of attribute is represented by a new attribute which has $0, 1$ value depending on the attribute value. Replace the above categorical attributes having more than 2 categories with respective one hot encoding attributes. For example, the attribute $team$ has the following categories $\{India, Australia, England\}$ then after replacement the attribute $team$ will be replaced with these 3 attributes $team\_India, team\_Australia$, $team\_England$ each having 2 categories $0, 1$. Repeat part (a) for this transformed dataset with maximum depths for $\{15, 25, 35, 45\}$. Compare the results with part (a) and comment on your observations. Feel free to experiment with other depths for performance gain. We will extend this version of the Decision Tree for the part (c) below.

(c) **(8 points) Decision Tree Post Pruning** One of the ways to reduce overfitting in decision trees is to grow the tree fully and then use post-pruning based on a validation set. In post-pruning, we greedily prune the nodes of the tree (and the sub-tree below them) by iteratively picking a node to prune so that the resultant tree gives a maximum increase in accuracy on the validation set. In other words, among all the nodes in the tree, we prune the node such that pruning it(and the sub-tree below it) results in a maximum increase in accuracy over the validation set. This is repeated until any further pruning leads to a decrease in accuracy over the validation set. Read the following notes on pruning decision trees to avoid overfitting (also available from the course website). Post-prune the trees of different maximum depths obtained in part (b) above using the validation set. Again for each tree plot the training, validation and test set accuracies against the number of nodes in the tree as you successively prune the tree. Comment on your findings.

(d) **(8 points) Decision Tree sci-kit learn:** As there are number of libraries which provide Decision tree implementation, we would like to compare our own implementation against these. Use the scikit learn library to build the decision tree on the dataset provided to you. Change the value of the *criterion* parameter to *entropy*. Next, (i) Vary max_depth in range $\{15, 25, 35, 45\}$. Keep all other arguments default. Report the train and test accuracies and plot the accuracies against maximum depth. Use the validation set to determine the best value of the depth parameter for your final model. (ii) Next, choose the default value of the depth parameter (which grows the tree fully), and vary value of the pruning parameters ccp_alpha, in the range $\{0.001, 0.01, 0.1, 0.2\}$. Use default value of all other parameters. Again, plot train and test accuracies against the value of the pruning parameter. Choose the value of the ccp_alpha parameter [1] which gives the maximum accuracy on the validation set, and

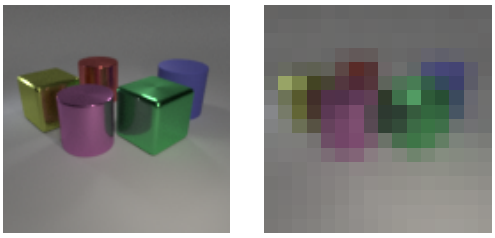
[1] You should read about the semantics of this parameter from Scikit's documentation.

use this as your final model. Now, compare the results obtained using the best model obtained in sub-parts (i) and (ii) above, i.e., with varying *max_depth*, and *ccp_alpha* parameters, with part (b) and (c) and comment on your observations. Note: Sci-kit's internal implementation uses a one-hot encoding to split multi-valued discrete attributes, so in some sense, results here are comparable to your implementations in parts (b) and (c), respectively.

(e) **(8 points) Random Forests:** Random Forests are extensions of decision trees, where we grow multiple decision trees in parallel on bootstrapped samples constructed from the original training data. A number of libraries are available for learning Random Forests over a given training data. In this particular question, you will use the sci-kit learn library of Python to grow a Random Forest. Click here to read the documentation and the details of various parameter options. Try growing different forests by playing around with various parameter values. Especially, you should experiment with the following parameter values (in the given range): (a) *n_estimators* (50 to 350 in range of 100). (b) *max_features* (0.1 to 1.0 in range of 0.2) (c) *min_samples_split* (2 to 10 in range of 2). You are free to try out non-default settings of other parameters too. Use the out-of-bag accuracy to tune to the optimal values for these parameters. You should perform a grid search over the space of parameters (read the description at the link provided for performing grid search). Report training, out-of-bag, validation and test set accuracies for the optimal set of parameters obtained. How do your numbers, i.e., train, validation and test set accuracies compare with those you obtained in part (c) and part (d) above?

(f) **Extra Fun: No Credits!**: Read about the XG-boost algorithm which is an extension of decision trees to Gradient Boosted Trees. You can read about gradient boosted trees here (link 1) and here (link 2). Try out using XG-boost on the above dataset. Try out different parameter settings, and find the one which does best on the validation set. Report the corresponding test accuracies. How do these compare with those reported for Random Forests?

2. **(38 points) Neural Networks:**



(a) Original Image (b) Re-scaled Image

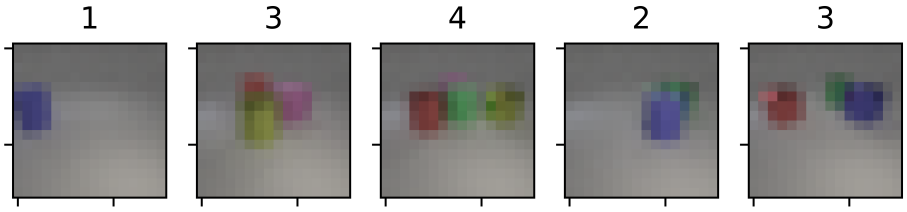


Figure 1: Sample data. The numeric label represents the true number of objects in an image.

In this problem, you will work with the modified version of CLEVR dataset. This is an image dataset consisting of RGBA pixel values of 16×16 size images of objects placed on grey surface. The task is to predict number of objects in an image. The number of objects in an image vary between 1 to 5. Therefore this is a multiclass dataset containing 5 classes. Fig 2 shows some sample examples. The Dataset contains a train and a test set consisting of 10000 and 1000 examples, respectively. Note that the dataset is balanced; i.e. the number of examples is approximately same for all classes in train and test set. Instead of using the standard squared error loss over the logistic function, we will instead use the cross-entropy loss defined over the softmax activation function. Softmax is a generalization of logistic function for the case of multi-valued variables. Given a set of linear inputs $\{net_k^{(L)}\}_{k=1}^r$, applying softmax over it, results in a probability vector of size r, given as $o_1, o_2 \cdots o_r$, such that $\sum_k o_k = 1$, and where $o_k = \frac{e^{net_k^{(L)}}}{\sum_{k'} e^{net_{k'}^{(L)}}}$ Given two distributions P and Q, defined over a random variable y, the cross-entropy between them is defined via the expression: $\sum_k P(y=k) log Q(y=k)$. Then, in our case, the loss $J(\theta)$, is defined as the cross entropy between the true label distribution given as $\tilde{p}(y=k|x) = \mathbb{1}\{k=\tilde{k}\}$, where $\tilde{k}$, is the true label, and the predicted distribution $\hat{p}(y=k|x) = o_k$ (recall that o_k is simply the probability of outputting label k by the network). Then, the cross entropy loss is defined as:

$$J(\theta) = \sum_{k=1}^{r} -\mathbb{1}\{k=\tilde{k}\} log(o_k) \tag{1}$$

The partial derivative of $J(\theta)$, as defined above is given as:

$$\frac{\partial J(\theta)}{\partial net_k^{(L)}} = \begin{cases} (o_k - 1), & \text{if } k = \tilde{k} \\ o_k & \text{otherwise} \end{cases} \tag{2}$$

(a) **(12 points)** Write a program to implement a generic neural network architecture to learn a model for multi-class classification which outputs probability distribution by using softmax. You will implement the backpropagation algorithm (from first principles) to train your network. You should use mini-batch Stochastic Gradient Descent (mini-batch SGD) algorithm to train your network. Use the Cross Entropy Loss over each mini-batch as your loss function (mentioned above in 1). You will use the sigmoid as activation function for the units in the hidden layer (we will experiment with other activation units in one of the parts below) and softmax at output layer to get final predicted probability distribution. Your implementation(including back-propagation) MUST be from first principles and not using any

pre-existing library in Python for the same. It should be generic enough to create an architecture based on the following input parameters:

- Mini-Batch Size (M)
- Number of features/attributes (n)
- Hidden layer architecture: List of numbers denoting the number of perceptrons in the corresponding hidden layer. Eg. a list [100 50] specifies two hidden layers; first one with 100 units and second one with 50 units.
- Number of target classes (r)

Assume a fully connected architecture i.e., each unit in a hidden layer is connected to every unit in the next layer.

(b) **(5 points)** Use the above implementation to experiment with a neural network having a **single** hidden layer. Vary the number of hidden layer units from the set {1, 5, 10, 50, 100}. Set the learning rate to 0.01. Use a mini-batch size of 32 examples. This will remain constant for the remaining experiments in the parts below. To be specific you will have following arguments in the generic neural network:

- $M = 32$
- $n = 1024 \;\; (16 \times 16 \times 4)$
- Hidden layer sizes to be choosen from options
- $r = 5$

Choose a suitable stopping criterion and report it. Read about precision, recall and F (also known as F1) score here. Report the precision, recall and F1 score for each class at different hidden layer units on test data and train data. You could compute these metrics using scikit-learn utility. Plot the average F1 score vs the number of hidden units. Comment your findings.

(c) **(5 points)** In this part we will experiment with the depth of neural network. Vary the hidden layer sizes as $\{\{512\}, \{512, 256\}, \{512, 256, 128\}, \{512, 256, 128, 64\}\}$. Keep learning rate and batch size same as part b. Use same stopping criteria as before and report the precision, recall and F1 score for all variations on test data and train data. Plot the average F1 score vs the network depth.

(d) **(5 points)** Use an adaptive learning rate inversely proportional to number of epochs i.e. $\eta_e = \frac{\eta_0}{\sqrt{e}}$ where $\eta_0 = 0.01$ is the seed value and e is the current epoch number[2] and repeat part c. See if you need to change your stopping criteria. Report your stopping criterion. As before, report the precision, recall and F1 score for each class at different hidden layer depth on test data and train data. Plot the average F1 score vs depth of hidden units. How do your results compare with those obtained in the part above? Does the adaptive learning rate make training any faster? Comment on your observations.

(e) **(6 points)** Several activation units other than sigmoid have been proposed in the literature such as tanh, and RelU to introduce non linearity into the network. ReLU is defined using the function: g(z) = max(0, z). In this part, we will replace the sigmoid activation units by the ReLU for all the units in the hidden layers of the network. You can read about relative advantage of using the ReLU over several other activation units on this blog.

Change your code to work with the ReLU activation unit in the hidden layers. The activations in the final layer still stay softmax. Note that there is a small issue with ReLU that it is non-differentiable at $z = 0$. This can be resolved by making the use of sub-gradients - intuitively, sub-gradient allows us to make use of any value between the left and right derivative at the point of non-differentiability to perform gradient descent see this (Wikipedia page for more details). Repeat the part d with ReLU activation. Report your training and test set precision, recall and F1 score. Plot the average F1 socre vs network depth. Also, make a relative comparison of test set accuracies obtained in part d. What do you observe? Which ones performs better?

(f) **(5 points)** Use MLPClassifier from scikit-learn library to implement a neural network with the same architectures as in part e above. Use Stochastic Gradient Descent as the solver. Note that MLPClassifier only allows for Cross Entropy Loss over the final network output. Set the following parameters:

- hidden_layer_sizes: to be vary according to part c.
- activation: relu
- solver: sgd

[2]One epoch corresponds to one complete pass through the data

- alpha: 0
- batch_size: 32
- learning_rate: invscaling

Keep all other parameter as default. You need to decide the stopping criteria accordingly. Now report the same metrics and plots as of part e. Compare the results with part e, and comment on your observations.