CS 5350/6350, DS 4350: Machine Learning Fall 2024

Homework 6

Handed out: 9 April, 2024
Due date: 23 April, 2024

General Instructions

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 10 pages**. Every extra page will cost a point.
- Handwritten solutions or photos of handwritten solutions will not be accepted.
- The homework is due by midnight of the due date. Please submit the homework on Canvas. You should upload two files: a report with answers to the questions below, and a compressed file (`.zip` or `.tar.gz`) containing your code.

Important Do not just put down an answer. We want explanations of your answers. No points will be given for just the final answer without an explanation.

1 Logistic Regression

We saw Maximum A Posteriori (MAP) learning of the logisitic regression classifier in class. In particular, we showed that learning the classifier is equivalent to the following optimization problem:

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)) + \frac{1}{\sigma^2} \mathbf{w}^T \mathbf{w}$$

In this question, you will derive the stochastic gradient descent algorithm for the logistic regression classifier.

1. [5 points] What is the derivative of the function $g(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$ with respect to the weight vector? Your answer should be a vector whose dimensionality is the same as $\mathbf{w}$. The gradient of the logistic loss function with respect to the weight

vector $\mathbf{w}$ for a single data point (x_i, y_i) is given by:
Ans)

$$\nabla_{\mathbf{w}} g(\mathbf{w}) = -\frac{y_i \mathbf{x}_i \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

2. [5 points] The inner most step in the SGD algorihtm is the gradient update where we use a single randomly chosen example instead of the entire dataset to compute a stochastic estimate of the gradient. Write down the objective where the entire dataset is composed of a single example, say $(\mathbf{x}_i, y_i)$.
Ans)

$$J = \min_{\mathbf{w}} \left\{ \log(1 + \exp(-y_i \mathbf{w}^T x_i)) + \frac{1}{\sigma^2} \mathbf{w}^T \mathbf{w} \right\}$$

3. [5 points] Derive the gradient of the SGD objective for a single example (from the previous question) with respect to the weight vector.
Ans)

$$\nabla_{\mathbf{w}} L(\mathbf{w}; x_i, y_i) = -\frac{\exp(-y \mathbf{w}^T \mathbf{x})}{1 + \exp(-y \mathbf{w}^T \mathbf{x})} y \mathbf{x} + \frac{2\mathbf{w}}{\sigma^2}$$

4. [15 points] Write down the pseudo code for the stochastic gradient algorithm using the gradient from previous part.

   Hint: The answer to this question will be an algorithm that is similar to the SGD based learner we developed in the class for SVMs.
   Ans)

Algorithm 1 Stochastic Gradient Descent Algorithm for Logistic Regression

1: Initialize $\mathbf{w}^0 = \mathbf{0} \in \mathbb{R}^n$;
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Pick a random example (x_i, y_i) from the training set S
4: Treat random example (x_i, y_i) as the full dataset and get
5: $\nabla_{\mathbf{w}} J^{(t-1)} = -\left(\frac{\exp(-y_i \mathbf{w}^{(t-1)T} x_i)}{1+\exp(-y_i \mathbf{w}^{(t-1)T} x_i)} \right) y_i x_i + \frac{2\mathbf{w}^{(t-1)}}{\sigma^2}$
6: Apply learning rate policy $\gamma(t) = f(\gamma^{(t-1)})$
7: Update $\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \gamma(t) \nabla_{\mathbf{w}} J^{(t-1)}$
8: **end for**

2 Experiments

For this question, you will have to implement and compare different learning strategies: SVM, logistic regression (from your answer to the previous question), and an ensemble that combines SVMs and decision trees.

2.1 The task and data

The data for this homework is adapted from the UCI credit card dataset. The goal is to predict whether a bank customer will default on their credit card payment. For more details about the data, see
Yeh, I. C., & Lien, C. H. (2009). *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients.* Expert Systems with Applications, 36(2), 2473-2480.

We have transformed the original features into a collection of binary features and have split the data into the usual training, testing and cross-validation splits. The data file contains:

1. `train.csv`: The full training set, with 20,000 examples.
2. `test.csv`: The test set, with 10,000 examples.
3. To help with cross-validation, we have split the training set into five parts `training00.csv` - `training04.csv` in the folder `CVSplits`.

All the data files are in the same format as we have used in the previous homeworks: Each row is an instance, the column `label` corresponds to the binary label, and the remaining columns correspond to the features.

2.2 Implementation and Evaluation Notes

Each algorithm has different hyper-parameters, as described below. Use 5-fold cross-validation to identify the best hyper-parameters as you did in previous homeworks. Refer to the description of cross-validation in homework 1 for more information.

An important difference between what we have seen in the previous homeworks and this one involves the metric that you will compute during cross-validation and final evaluation. The positive and negative labels are not balanced in this data. With such unequally distributed labels, we usually measure precision, recall and F-scores because the accuracy of a classifier could be misleading.

To compute these quantities, you should count the number of true positives (that is, examples that your classifier predicts as positive and are truly positive), the false positives (i.e, examples that your classifier predicts as positive, but are actually labeled negative) and the false negatives (i.e., examples that are predicted as negative by your classifier, but are actually positive).

Denote true positives, false positive and false negative as TP, FP and FN respectively. The precision (p), recall (r) and f-value F_1 are defined as:

$$p = \frac{TP}{TP + FP} \qquad r = \frac{TP}{TP + FN} \qquad F_1 = 2\frac{p \cdot r}{p + r}$$

(Sidenote: The F_1 score is defined to be the harmonic mean of the precision and the recall. That is, it is defined to be the number such that $\frac{2}{F_1} = \frac{1}{p} + \frac{1}{r}$. Reorganizing this gives us the expression above.)

For all your classifiers, you should report measure precision, recall and F_1. During cross-validation, use the average F_1 instead of average accuracy.

2.3 Algorithms to Compare

1. [30 points] **Support Vector Machine**

   Implement the stochastic sub-gradient descent version algorithm SVM as described in the class. Assume that the learning rate for the t^{th} epoch is

   $$\gamma_t = \frac{\gamma_0}{1+t}$$

   For this, and all subsequent implementations, you should choose an appropriate number of epochs and justify your choice. One way to select the number of epochs is to observe the value of the SVM objective over the epochs and stop if the change in the value is smaller than some small threshold. You do not have to use this strategy, but your report should specify the number of epochs you chose.

   Hyper-parameters:

   (a) Initial learning rate: $\gamma_0 \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

   (b) The regularization/loss tradeoff parameter: $C \in \{10^1, 10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$

2. [30 points] **Logistic regression**

   Implement the Logistic Regression learner based on your algorithm in the Question 1.

   Hyper-parameters:

   (a) Initial learning rate: $\gamma_0 \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

   (b) Tradeoff: $\sigma^2 \in \{10^{-1}, 10^0, 10^1, 10^2, 10^3, 10^4\}$

3. [30 points, Extra credit] **SVM over trees**

   In class, we saw how the bagging and random forest algorithms work. In this setting, you are going to build a different ensemble over depth-limited decision trees that are learned using the ID3 algorithm.

   First, using the training set, you need to build 100 decision trees. To construct a decision tree, you need to sample 10% of the examples *with replacement* from the training set (i.e. 1000 examples), and use this subset to train your decision tree with a depth limit d. Repeating this 100 times will get you 100 trees.

   Usually, the final prediction will be voted on by these trees. However, we would like to train an SVM to combine these predictions. To do so, you should treat the 100 trees as a *learned* feature transformation and construct a new dataset by applying the transformation. That is, suppose your trees were $tree_1, tree_2, \cdots, tree_{100}$. Each of these are functions that can predict a label for an example that is either -1 or $+1$. Instead of just predicting the label, treat them as a feature transformation $\phi(x)$ that is defined as:

   $$\phi(x) = [tree_1(x), tree_2(x), \cdots, tree_N(x)]$$

In other words, you will build an $N = 100$ dimensional vector consisting of the prediction (1 or -1) of each tree that you created. Thus, you have a *learned* feature transformation.

Now, you can train an SVM on these transformed features. (Don't forget to transform the test set before making your final evaluations.)

Hyper-parameters:

(a) Initial learning rate $\gamma_0 \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

(b) Tradeoff $C \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$

(c) Depth limit: $d \in \{5, 10\}$

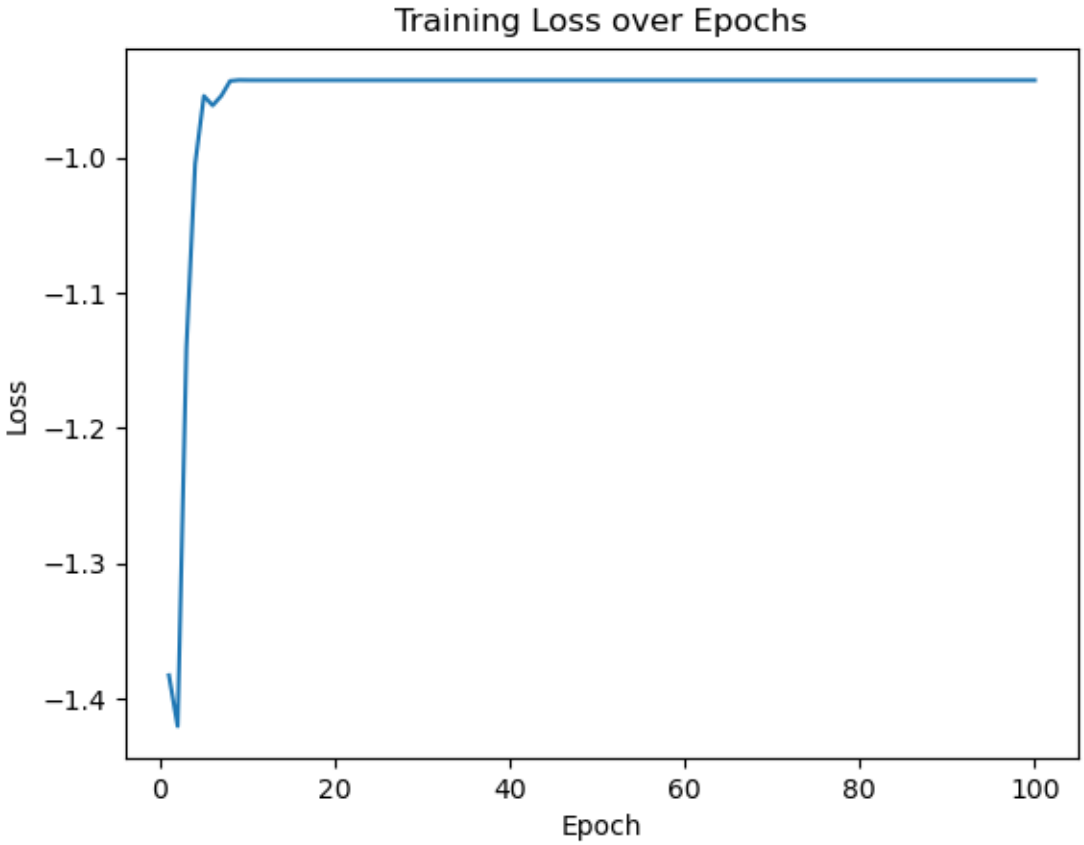
2.4 What to report

1. For each algorithm above, briefly describe the design decisions that you have made in your implementation. (E.g, what programming language, how do you represent the vectors, trees, etc.)

SVM

- I chose 100 epoch as it gave me clearer picture, even though it definitely took a lot of time.
- **Sigmoid Function:** The sigmoid function is utilized as the activation function in logistic regression, effectively mapping input values to a probability between 0 and 1.
- **Overflow Prevention:** To prevent numerical overflow, the input to the sigmoid function is clipped between -500 and 500. This prevents errors during computation caused by extremely large values.
- **Gradient Calculation:** The gradient computation includes a regularization term $C \times w$, which aids in preventing overfitting by penalizing large weight values.
- **Loss Function:** Loss is calculated using binary cross-entropy, ideal for binary classification tasks as it measures the difference between the predicted probabilities and the actual binary outcomes.
- **Batch Processing:** SGD with batch processing enables the model to update weights more frequently than in full-batch gradient descent, leading to quicker convergence and more efficient handling of large datasets.
- **Epochs and Learning Rate Decay:** The learning rate decays over epochs using the formula $\frac{\text{learning rate}}{1+\text{epoch}}$, reducing the step size gradually to aid in stabilizing the convergence of the algorithm.
- **Cross-Validation Setup:** Cross-validation is performed by iterating over specified fold paths, which helps ensure that the model generalizes well by being validated against different subsets of data.

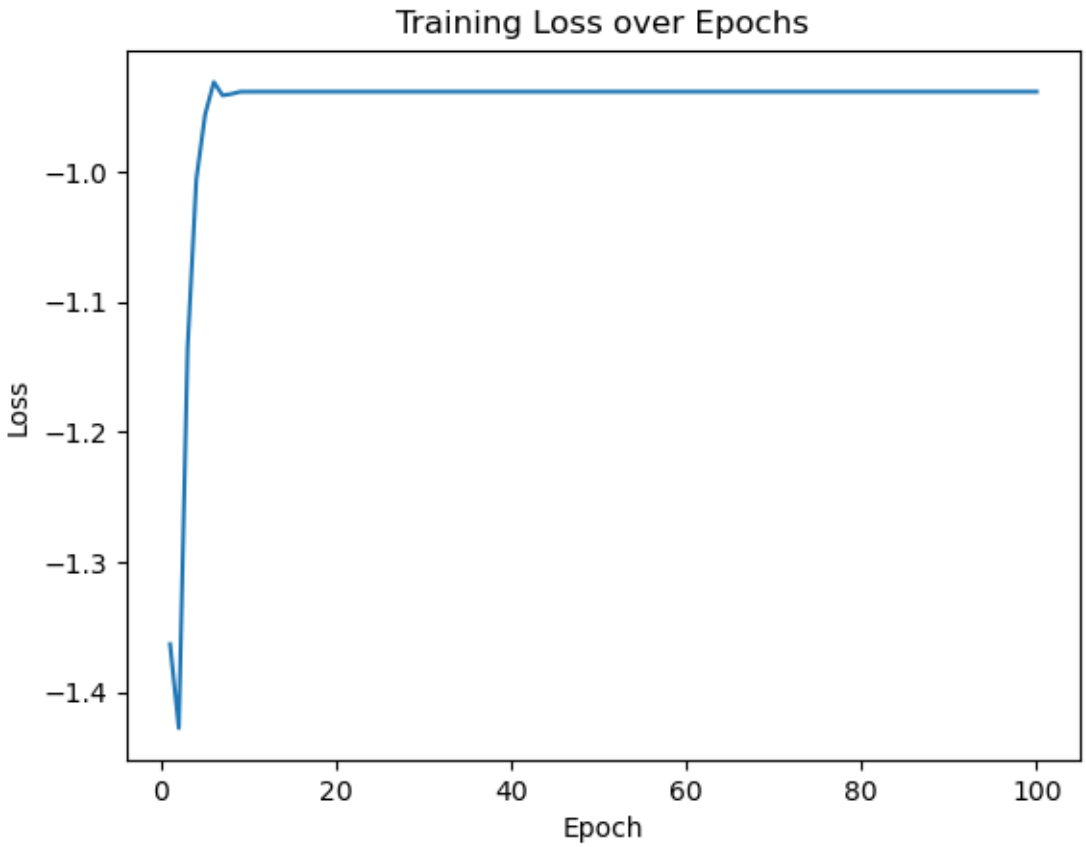
- **Hyperparameter Grid:** A systematic exploration of a grid of learning rates and regularization strengths (C) allows for the tuning of the model to find the best combination that either minimizes validation loss or maximizes performance metrics like the F1 score.



Design Choices for LR

- I chose 100 epoch as it gave me clearer picture, even though it definitely took a lot of time.
- general libraries like numpy, pandas and matplotlib were used.
- **Sigmoid Activation Function:** The sigmoid function is properly defined to prevent overflow issues with very large positive or negative inputs by clipping values to the range $[-500, 500]$.
- **Gradient Computation:** The `compute_gradient` function calculates the gradient of the loss function for logistic regression with a regularization term $C \cdot \mathbf{w}$. The term $*2 - 1$ is applied to `y_pred` to transform probabilities into the $\{-1, 1\}$ label space which is not a typical operation in logistic regression. Usually, the predicted probabilities are directly used for calculating the gradient.

- **Mini-batch SGD:** In `logistic_regression_SGD`, the data is shuffled at the beginning of each epoch and mini-batches are processed. This adds randomness to the gradient estimation and can help the algorithm escape local minima.
- **Learning Rate Decay:** The learning rate is decayed after each epoch according to the formula `learning_rate/` $= (1 +$ `epoch`$)$. This can help the algorithm to take smaller steps as it converges, preventing overshooting the minimum.



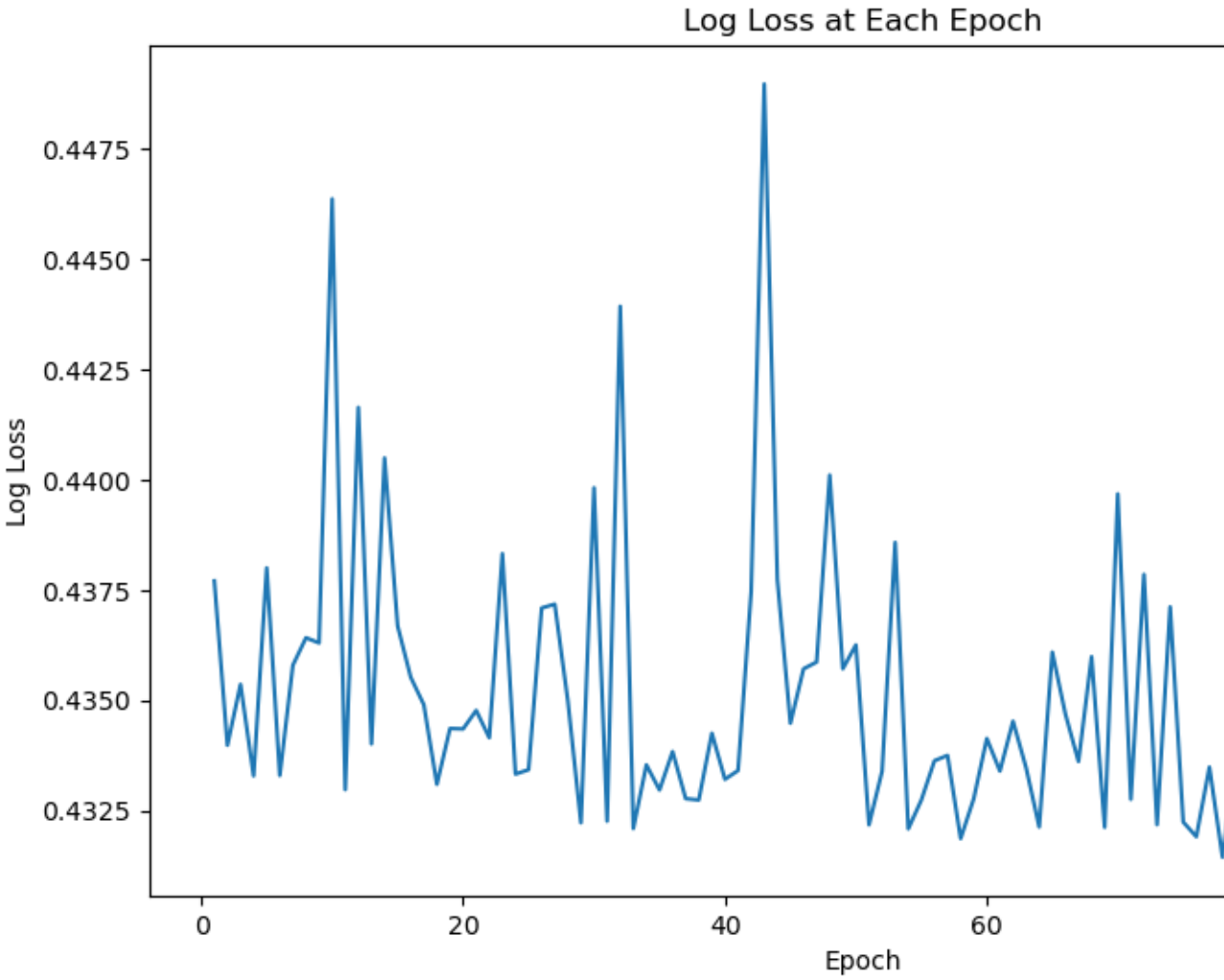
SVM Over Trees

- **Hyperparameter Search Space:**
  - **Learning Rates and Tradeoffs:** Although included in the search grid, `learning_rates` and `tradeoffs` are not parameters of the `DecisionTreeClassifier`. This seems to be a mistake, indicating a misunderstanding or a copy-paste error from templates used for models that rely on gradient-based optimization like logistic regression or SVMs.
  - **Tree Depths:** The array $[5, 10]$ specifies the tree depths to explore. Deeper trees can model more complex patterns but may overfit. Adjusting tree depth allows for finding a balance between capturing sufficient detail and avoiding overfitting.

- **Cross-Validation:**
  - **Implementation:** The model's performance is evaluated using manually conducted cross-validation with five folds. This approach helps assess generalization across different data subsets.
  - **Validation Strategy:** Each fold uses a fixed test size of 20% and varying `random_state` settings to ensure diverse validation subsets, potentially capturing a broader range of data characteristics.
- **Performance Metrics:**
  - Metrics like precision, recall, and F1-score are calculated for each validation set. The F1-score, combining precision and recall harmonically, serves as the primary metric for model selection, particularly beneficial in scenarios with imbalanced classes.
- **Model Training and Testing:**
  - **Final Model Training:** After determining the optimal hyperparameters, the final model is trained using the entire dataset, maximizing the data's utility under the best settings found.
  - **Testing:** The model's performance on a separate test set provides the ultimate measure of its ability to generalize, serving as the final validation of its effectiveness.

	Best hyper-parameters	Average Cross-validation P/R/F1	Test P/R/F1
SVM	1,0.0001	0.653,0.342,0.449	0.713,0.359,0.477
Logistic regression	1,1000	0.654,0.341,0.448	0.716,0.355,0.475
SVM over trees (optional)	1,1,5	0.663,0.338,0.446	0.704,0.361,0.477

Table 1: Results table



2. Report the best hyper-parameters, the average precision, recall and F_1 achieved by those hyperparameters during cross-validation and the precision/recall/F_1 on the test set. You can use the table 1 as a template for your reporting.

3. For the algorithms that involve minimizing loss, show a plot of the corresponding loss function at each epoch. For the SVM over trees, you should plot the loss function for the SVM.

(Note that to get this plot, you should compute the loss at the end of each epoch over the entire training set. This requires a second pass over the training data that is different from the weight update loop.)

Experiment Submission Guidelines

1. The report should detail your experiments. For each step, explain in no more than a paragraph or so how your implementation works. Describe what you did. Comment on the design choices in your implementation. For your experiments, what algorithm parameters did you use? Try to analyze this and give your observations.

2. Your report should be in the form of a *pdf* file, LaTeX is recommended.

3. *Your code should run on the CADE machines.* You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.

   You are responsible for ensuring that the grader can execute the code using only the included script.

4. Please do not hand in binary files! We will *not* grade binary submissions.

5. Please look up the late policy on the course website.