CPSC 340 Assignment 4 (due Wednesday, Mar 6 at 11:55pm)

Zitong Wang k5j0b, Jingyuan Hu z3o0b

Instructions

Rubric: {mechanics:5}

IMPORTANT!!! Before proceeding, please carefully read the general homework instructions at `https://www.cs.ubc.ca/~fwood/CS340/homework/`. The above 5 points are for following the submission instructions. You can ignore the words "mechanics", "reasoning", etc.

We use blue to highlight the deliverables that you must answer/do/submit with the assignment.

1 Convex Functions

Rubric: {reasoning:5}

Recall that convex loss functions are typically easier to minimize than non-convex functions, so it's important to be able to identify whether a function is convex.

Show that the following functions are convex:

1. $f(w) = \alpha w^2 - \beta w + \gamma$ with $w \in \mathbb{R}, \alpha \geq 0, \beta \in \mathbb{R}, \gamma \in \mathbb{R}$ (1D quadratic).
2. $f(w) = -\log(\alpha w)$ with $\alpha > 0$ and $w > 0$ ("negative logarithm")
3. $f(w) = \|Xw - y\|_1 + \frac{\lambda}{2}\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized robust regression).
4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).
5. $f(w) = \sum_{i=1}^{n} [\max\{0, |w^T x_i - y_i|\} - \epsilon] + \frac{\lambda}{2}\|w\|_2^2$ with $w \in \mathbb{R}^d, \epsilon \geq 0, \lambda \geq 0$ (support vector regression).

General hint: for the first two you can check that the second derivative is non-negative since they are one-dimensional. For the last 3 you'll have to use some of the results regarding how combining convex functions can yield convex functions which can be found in the lecture slides.

Hint for part 4 (logistic regression): this function may seem non-convex since it contains $\log(z)$ and log is concave, but there is a flaw in that reasoning: for example $\log(\exp(z)) = z$ is convex despite containing a log. To show convexity, you can reduce the problem to showing that $\log(1 + \exp(z))$ is convex, which can be done by computing the second derivative. It may simplify matters to note that $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

Answer:

1. **Since:** $f'(w) = 2\alpha w - \beta$, $f''(w) = 2\alpha$ where $\alpha > 0$, therefore we have $f''(w) > 0$. $f(w)$ is a convex function.
2. **Since:** $f'(w) = -\frac{1}{\alpha w}$ and $f''(w) = \frac{\alpha}{(\alpha w)^2}$, we have $\alpha > 0, w > 0$ therefore, the second derivative $f''(w) > 0$.
3. **Since:** $\|Xw - y\|_1$ and $\|w\|_1$ are all norms, they are all convex function; by multiplying with $\frac{\lambda}{2}$, a non-negative constant, $\frac{\lambda}{2}\|w\|$ is also convex. The sum of 2 convex function are convex as well, therefore $f(w)$ is a convex function.

4. **Let** $z = -y_i w^T x^i$ such that we have: $f(z) = log(1 + exp(z))$, $f'(z) = \frac{exp(z)}{1+exp(z)} = \frac{1}{1+exp(-z)}$ and $f''(z) = \frac{exp(-z)}{(1+exp(-z))^2}$. Since $z = -y_i w^T x^i$ is a linear function that it is always convex. By plugging into $exp(z)$, it is always greater tha zero, therefore $f''(w) > 0$ meaning the function is convex.

5. **Since:** $w^T x_i - y_i$ is a linear function and ϵ is a contant greater than zero, we should have $|w^T x_i - y_i|$ is also convex. Also, by definition, the Max of a convex function is also a convex, therefore $max\{0, |w^T x_i - y_i|\} - \epsilon$ is also convex. Since $\|w\|_2^2$ is euclidean norm and $\frac{\lambda}{2}$ is a positive constant that $\frac{\lambda}{2}\|w\|_2^2$ is still a convex function, therefore $f(w) = \sum_{i=1}^{n}[\max\{0, |w^T x_i - y_i|\} - \epsilon] + \frac{\lambda}{2}\|w\|_2^2$ is a convex function.

2 Logistic Regression with Sparse Regularization

If you run `python main.py -q 2`, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. 'Standardize' the columns of X and add a bias variable (in *utils.load_dataset*).
3. Apply the same transformation to $Xvalidate$ (in *utils.load_dataset*).
4. Fit a logistic regression model.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the validation set.

Logistic regression does reasonably well on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent, if you have enough data and know which features are relevant). In this question, you will modify this demo to use different forms of regularization to improve on these aspects.

Note: your results may vary a bit depending on versions of Python and its libraries.

2.1 L2-Regularization

Rubric: {code:2}

Make a new class, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w by minimizing

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)),$$

your new function *logRegL2* should compute w by minimizing

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \frac{\lambda}{2}\|w\|^2.$$

Hand in your updated code. Using this new code with $\lambda = 1$, report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

Note: as you may have noticed, `lambda` is a special keyword in Python and therefore we can't use it as a variable name. As an alternative we humbly suggest `lammy`, which is what Mike's niece calls her stuffed animal toy lamb. However, you are free to deviate from this suggestion. In fact, as of Python 3 one can now use actual greek letters as variable names, like the λ symbol. But, depending on your text editor, it may be annoying to input this symbol.

Answer:

logRegL2 Training error 0.002

logRegL2 Validation error 0.074

nonZeros (number of selected features): 101

Number of gradient descent iterations: 35

For code submission, please refer to file **linear_model.py** inside code repository.

2.2 L1-Regularization

Rubric: {code:3}

Make a new class, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_1.$$

Hand in your updated code. Using this new code with $\lambda = 1$, report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

You should use the function *minimizers.findMinL1*, which implements a proximal-gradient method to minimize the sum of a differentiable function g and $\lambda\|w\|_1$,

$$f(w) = g(w) + \lambda\|w\|_1.$$

This function has a similar interface to *findMin*, **EXCEPT** that (a) you only pass in the the function/gradient of the differentiable part, g, rather than the whole function f; and (b) you need to provide the value λ. To reiterate, your `funObj` **should not contain the L1 regularization term**; rather it should only implement the function value and gradient for the training error term. The reason is that the optimizer handles the non-smooth L1 regularization term in a specialized way (beyond the scope of CPSC 340).

Answer:

logRegL1 Training error 0.000

logRegL1 Validation error 0.052

nonZeros (features selected): 71

Number of gradient descent iterations: 77

For code submission, please refer to file **linear_model.py** inside code repository.

2.3 L0-Regularization

Rubric: {code:4}

The class *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda \|w\|_0.$$

The `for` loop in this function is missing the part where we fit the model using the subset *selected_new*, then compute the score and updates the *minLoss/bestFeature*. Modify the `for` loop in this code so that it fits the model using only the features *selected_new*, computes the score above using these features, and updates the *minLoss/bestFeature* variables. Hand in your updated code. Using this new code with $\lambda = 1$, report the training error, validation error, and number of features selected.

Note that the code differs a bit from what we discussed in class, since we assume that the first feature is the bias variable and assume that the bias variable is always included. Also, note that for this particular case using the L0-norm with $\lambda = 1$ is equivalent to what is known as the Akaike Information Criterion (AIC) for variable selection.

Also note that, for numerical reasons, your answers may vary depending on exactly what system and package versions you are using. That is fine.

Answer:

Training error 0.000

Validation error 0.018

Number of features selected: 24

For code submission, please refer to file **linear_model.py** inside code repository.

2.4 Discussion

Rubric: {reasoning:2}

In a short paragraph, briefly discuss your results from the above. How do the different forms of regularization compare with each other? Can you provide some intuition for your results? No need to write a long essay, please!

Answer:

For validation error: L0 $<$ L1 $<$ L2, for training error: L2 $>$ L1 $=$ L0 and for number of features selection: L2 $>$ L1 $>$ L0.

Therefore, we can conclude that: L2 has the highest validation error but it does not necessarily means that it would perform badly on real world problem, since it uses all features and has least iterations. L1 has less feature selection as well as lower validation error. Among all 3, L0 seems to have the best performance since it has the lowest validation error and least feature selection.

2.5 Comparison with scikit-learn

Rubric: {reasoning:1}

Compare your results (training error, validation error, number of nonzero weights) for L2 and L1 regularization with scikit-learn's LogisticRegression. Use the `penalty` parameter to specify the type of regularization. The parameter `C` corresponds to $\frac{1}{\lambda}$, so if you had $\lambda = 1$ then use `C=1` (which happens to be the default anyway). You should set `fit_intercept` to `False` since we've already added the column of ones to X and thus there's no need to explicitly fit an intercept parameter. After you've trained the model, you can access the weights with `model.coef_`.

Answer:

For scikit-learn's LogisticRegression, I obtained:

For L1:

Training error 0.000

Validation error 0.052

Number of features selected: 71

For L2:

Training error 0.002

Validation error 0.074

Number of features selected: 101

Discussion:

By comparing with Scikit-learn's implementations. Our implementation has basically the same training error and validation as them, and the number of selected features are the same for both L1 and L2.

2.6 L$\frac{1}{2}$ regularization

Rubric: {reasoning:4}

Previously we've considered L2 and L1 regularization which use the L2 and L1 norms respectively. Now consider least squares linear regression with "L$\frac{1}{2}$ regularization" (in quotation marks because the "L$\frac{1}{2}$ norm" is not a true norm):

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 + \lambda \sum_{j=1}^{d} |w_j|^{1/2} .$$

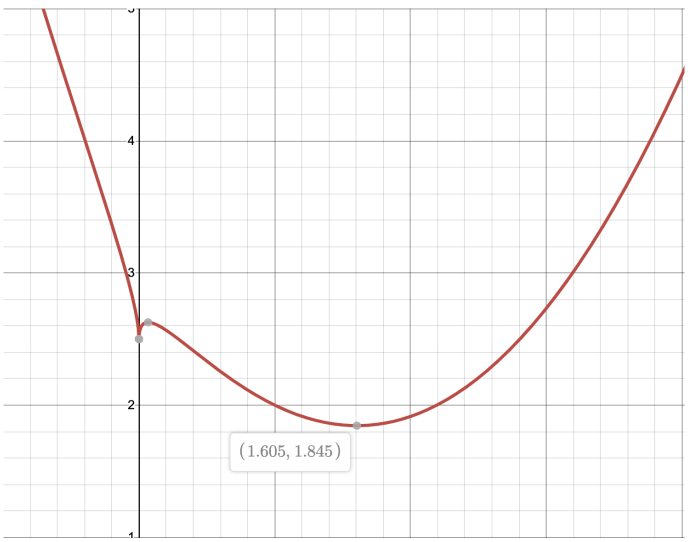
Let's consider the case of $d = 1$ and assume there is no intercept term being used, so the loss simplifies to

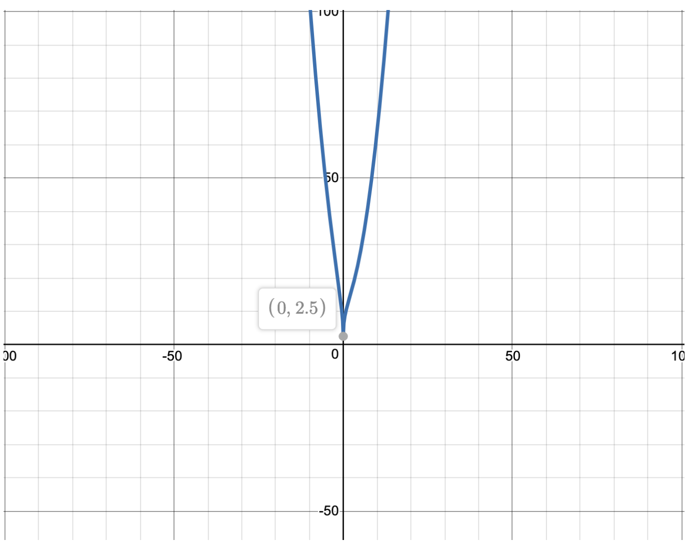
$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2 + \lambda\sqrt{|w|} .$$

Finally, let's assume $n = 2$ where our 2 data points are $(x_1, y_1) = (1, 2)$ and $(x_2, y_2) = (0, 1)$.

1. Plug in the data set values and write the loss in its simplified form, without a summation.
2. If $\lambda = 0$, what is the solution, i.e. $\arg\min_w f(w)$?
3. If $\lambda \to \infty$, what is the solution, i.e., $\arg\min_w f(w)$?
4. Plot $f(w)$ when $\lambda = 1$. What is $\arg\min_w f(w)$ when $\lambda = 1$? Answer to one decimal place if appropriate.
5. Plot $f(w)$ when $\lambda = 10$. What is $\arg\min_w f(w)$ when $\lambda = 10$? Answer to one decimal place if appropriate.
6. Does L$\frac{1}{2}$ regularization behave more like L1 regularization or L2 regularization when it comes to performing feature selection? Briefly justify your answer.
7. Is least squares with L$\frac{1}{2}$ regularization a convex optimization problem? Briefly justify your answer.

Answer:

1. By plugging in $(x_1, y_1) = (1, 2)$ and $(x_2, y_2) = (0, 1)$ we have $f(w) = \frac{1}{2}((w-2)^2 + 1) + \lambda\sqrt{|w|}$
2. By setting $\lambda = 0$ we have: $f(w) = \frac{1}{2}((w-2)^2 + 1)$, therefore, taking first derivative would be: $f'(w) = (w-2)$ by setting first derivative to 0 we have $argmin_w f(w) = 2$
3. By taking first derivative, we have: $f'(w) = (w-2) + \frac{1}{2}\lambda|w|^{-1/2}$, by letting λ goes to ∞, $argmin_w f(w)$ would be equaled to zero (i.e select no feature).
4. 

   For $\lambda = 1$ we have $argmin_w f(w) = 1.6$
5. 

   For $\lambda = 10$ we have $argmin_w f(w) = 0$
6. By summarizing the information above, $L\frac{1}{2}$ regularization behaves more like L1 regularization. As λ tends to be large (i.e $\lambda = 10$), $w = 0$ (i.e select no feature). Just like L1 regularization.
7. It is not a convex optimization problem, since $\sqrt{|w|}$ is not in convex form.

3 Multi-Class Logistic

If you run `python main.py -q 3` the code loads a multi-class classification dataset with $y_i \in \{0, 1, 2, 3, 4\}$ and fits a 'one-vs-all' classification model using least squares, then reports the validation error and shows a plot of the data/classifier. The performance on the validation set is ok, but could be much better. For example, this classifier never even predicts that examples will be in classes 0 or 4.

3.1 Softmax Classification, toy example

Rubric: {reasoning:2}

Linear classifiers make their decisions by finding the class label c maximizing the quantity $w_c^T x_i$, so we want to train the model to make $w_{y_i}^T x_i$ larger than $w_{c'}^T x_i$ for all the classes c' that are not y_i. Here c' is a possible label and $w_{c'}$ is row c' of W. Similarly, y_i is the training label, w_{y_i} is row y_i of W, and in this setting we are assuming a discrete label $y_i \in \{1, 2, \ldots, k\}$. Before we move on to implementing the softmax classifier to fix the issues raised in the introduction, let's work through a toy example:

Consider the dataset below, which has $n = 10$ training examples, $d = 2$ features, and $k = 3$ classes:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 2 \\ 3 \\ 3 \\ 3 \end{bmatrix}.$$

Suppose that you want to classify the following test example:

$$\hat{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

Suppose we fit a multi-class linear classifier using the softmax loss, and we obtain the following weight matrix:

$$W = \begin{bmatrix} +2 & -1 \\ +2 & -2 \\ +3 & -1 \end{bmatrix}$$

Under this model, what class label would we assign to the test example? (Show your work.)

Answer:

From matrix W, we have: $w_1^T = [2 \;\; -1]$, $w_2^T = [2 \;\; -2]$ and $w_3^T = [3 \;\; -1]$

Therefore, we have:

$\hat{x}w_1 = 1 * (2) + 1 * (-1) = 1$

$\hat{x}w_2 = 1 * (2) + 1 * (-2) = 0$

$\hat{x}w_3 = 1 * (3) + 1 * (-1) = 2$,

by multiplying with $w_{y_i=3}$ receive the best score. There for $\hat{x}$ should be classified as class 3.

3.2 One-vs-all Logistic Regression

Rubric: {code:2}

Using the squared error on this problem hurts performance because it has 'bad errors' (the model gets penalized if it classifies examples 'too correctly'). Write a new class, *logLinearClassifier*, that replaces the squared loss in the one-vs-all model with the logistic loss. Hand in the code and report the validation error.

Answer:

logLinearClassifier Validation error 0.070

For code submission, please refer to file linear_model.py in code repository.

3.3 Softmax Classifier Gradient

Rubric: {reasoning:5}

Using a one-vs-all classifier can hurt performance because the classifiers are fit independently, so there is no attempt to calibrate the columns of the matrix W. As we discussed in lecture, an alternative to this independent model is to use the softmax loss, which is given by

$$f(W) = \sum_{i=1}^{n} \left[-w_{y_i}^T x_i + \log\left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i) \right) \right],$$

Show that the partial derivatives of this function, which make up its gradient, are given by the following expression:

$$\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij} [p(y_i = c \mid W, x_i) - I(y_i = c)],$$

where...

- $I(y_i = c)$ is the indicator function (it is 1 when $y_i = c$ and 0 otherwise)
- $p(y_i = c \mid W, x_i)$ is the predicted probability of example i being class c, defined as

$$p(y_i = c \mid W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)}$$

Answer:

We first take the partial derivative of $\log(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i))$

We have: $\frac{\partial f}{\partial W_{cj}} \log(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} x_{ij} = p(y_i = c \mid W, x_i) x_{ij}$

Since we are taking the partial derivative with resepect to a specific class c

Therefore $\frac{\partial f}{\partial W_{cj}}(-w_{y_i}^T x_i) = -x_{ij}$ and it only exist if $y_i = c$

By combing procedures above, $\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij} (\log(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)) - 1)$

Where the constant 1 term only exist when $y_i = c$ and $p(y_i = c \mid W, x_i) = \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)}$

Therefore, $\frac{\partial f}{\partial W_{cj}} = \sum_{i=1}^{n} x_{ij} [p(y_i = c \mid W, x_i) - I(y_i = c)]$

3.4 Softmax Classifier Implementation

Rubric: {code:5}

Make a new class, *softmaxClassifier*, which fits W using the softmax loss from the previous section instead of fitting k independent classifiers. Hand in the code and report the validation error.

Hint: you may want to use `utils.check_gradient` to check that your implementation of the gradient is correct.

Hint: with softmax classification, our parameters live in a matrix W instead of a vector w. However, most optimization routines like `scipy.optimize.minimize`, or the optimization code we provide to you, are set up to optimize with respect to a vector of parameters. The standard approach is to "flatten" the matrix W into a vector (of length kd, in this case) before passing it into the optimizer. On the other hand, it's inconvenient to work with the flattened form everywhere in the code; intuitively, we think of it as a matrix W and our code will be more readable if the data structure reflects our thinking. Thus, the approach we recommend is to reshape the parameters back and forth as needed. The `funObj` function is directly communicating with the optimization code and thus will need to take in a vector. At the top of `funObj` you can immediately reshape the incoming vector of parameters into a $k \times d$ matrix using `np.reshape`. You can then compute the gradient using sane, readable code with the W matrix inside `funObj`. You'll end up with a gradient that's also a matrix: one partial derivative per element of W. Right at the end of `funObj`, you can flatten this gradient matrix into a vector using `grad.flatten()`. If you do this, the optimizer will be sending in a vector of parameters to `funObj`, and receiving a gradient vector back out, which is the interface it wants – and your `funObj` code will be much more readable, too. You may need to do a bit more reshaping elsewhere, but this is the key piece.

Answer:

For Softmax Classifer, we have its Validation error = 0.008

For code submission, please refer to file linear_model.py in code repository.

3.5 Comparison with scikit-learn, again

Rubric: {reasoning:1}

Compare your results (training error and validation error for both one-vs-all and softmax) with scikit-learn's `LogisticRegression`, which can also handle multi-class problems. One-vs-all is the default; for softmax, set `multi_class='multinomial'`. For the softmax case, you'll also need to change the solver. You can use `solver='lbfgs'`. Since your comparison code above isn't using regularization, set `C` very large to effectively disable regularization. Again, set `fit_intercept` to `False` for the same reason as above (there is already a column of 1's added to the data set).

Answer:

For scikit-learn's One vs All Logistic Regression, we obtained:

Training error = 0.084

Validation error = 0.070

For scikit-learn's implementation of Softmax Classifer, we obtained:

Training error = 0.000

Validation error = 0.008

It is worth noticing that, ours and scikit-learn's implementation of softmax classifier has training error = 0 and our implementation of softmax classifier and One-vs-all logistic regression tend to have the same result as scikit-learn's.

3.6 Cost of Multinomial Logistic Regression

Rubric: {reasoning:2}

Assume that we have

- n training examples.
- d features.
- k classes.
- t testing examples.
- T iterations of gradient descent for training.

Also assume that we take X and form new features Z using Gaussian RBFs as a non-linear feature transformation.

1. In $O()$ notation, what is the cost of training the softmax classifier with gradient descent?
2. What is the cost of classifying the t test examples?

Hint: you'll need to take into account the cost of forming the basis at training (Z) and test ($\tilde{Z}$) time. It will be helpful to think of the dimensions of all the various matrices.

Answer:

1. $O(n^2d + Tkn^2)$

   Since X is a (n x d) matrix while new features Z is a (n x n) matrix, therefore, forming training Z would have cost $O(n^2d)$. We have T iterations of gradient descent and w is a (n by k) matrix, such that gradient descent will cost $O(Tkn^2)$. So total would be $O(n^2d + Tkn^2)$.

2. $O(n^2k)$

   Since Z is a (n x n) matrix while w is a (n x k) matrix, therefore the dominating cost of estimating t examples is $O(n^2k)$.

4 Very-Short Answer Questions

Rubric: {reasoning:12}

1. Suppose that a client wants you to identify the set of "relevant" factors that help prediction. Why shouldn't you promise them that you can do this?
   It is hard to define what is relevant in real world problem and it varies among dataset

2. Consider performing feature selection by measuring the "mutual information" between each column of X and the target label y, and selecting the features whose mutual information is above a certain threshold (meaning that the features provides a sufficient number of "bits" that help in predicting the label values). Without delving into any details about mutual information, what is a potential problem with this approach?
   One possible reason would be multicollinearity, since $(-10000)x_1 + (10000)x_2$ where $x_1 = x_2$ and both

of them are actually not relevant (but now selected), or some feature are not influential as a single one, but affects prediction when interacting with others. Another reason is that recall the "Taco Tuesday" example, if taco led you sick but it happens to be eaten at Tuesday, then 'Tuesday' will take into account as 'Relevant' but it in fact it is not, therefore it would lead to a problem.

3. What is a setting where you would use the L1-loss, and what is a setting where you would use L1-regularization?
   L1-loss is robust to outliers data points (less sensitve to outliers), such that it would fit better with outliers' samples. L1 regularization is less sensitive to irrelevant features that can be used for feature selection.

4. Among L0-regularization, L1-regularization, and L2-regularization: which yield convex objectives? Which yield unique solutions? Which yield sparse solutions?
   $L1, L2$ yields convex objectives; $L2$ unique solutions; $L0, L1$ yieds sparse solutions

5. What is the effect of λ in L1-regularization on the sparsity level of the solution? What is the effect of λ on the two parts of the fundamental trade-off?
   As λ increases, the level of sparsity will also increases, which will lead to higher training error while lower approximation error

6. Suppose you have a feature selection method that tends not generate false positives but has many false negatives (it misses relevant variables). Describe an ensemble method for feature selection that could improve the performance of this method.
   Use bootstrap approach to reduce false negative (Apply the method to bootstrap samples of the training data, and only take the features selected in all bootstrap samples.)

7. Suppose a binary classification dataset has 3 features. If this dataset is "linearly separable", what does this precisely mean in three-dimensional space?
   There is a separating hyperplane/perfect linear classifier (of $\mathbb{R}^2$) in $\mathbb{R}^3$ that separates all data point of label 0 and 1 (binary)

8. When searching for a good w for a linear classifier, why do we use the logistic loss instead of just minimizing the number of classification errors?
   Logistic loss function has all the good properties (convex and differentiable, non-degenerate)

9. What are "support vectors" and what's special about them?
   Support vectors are the points closest to the line, they can separate the points the most (maximize the margin for linear-separable dataset)

10. What is a disadvantage of using the perceptron algorithm to fit a linear classifier?
    If a perfect classifer exists (i.e. for linearly separable datasets) than you can find it in finite steps, but if not then it can be costly and inaccurate

11. Why we would use a multi-class SVM loss instead of using binary SVMs in a one-vs-all framework?
    Using Binary classifier does not train the model so that the correct class has the largest value. (Can think of this as a some sort of "Greedy algorithm" that might not be global "optimal")

12. How does the hyper-parameter σ affect the shape of the Gaussian RBFs bumps? How does it affect the fundamental tradeoff?
    Large σ leads to less narrow bump (since $g(\epsilon) = exp(-\frac{\epsilon^2}{2\sigma^2})$) and hence simplier model, so may lead to a higher training error but lower approximation error