CPSC 340 Assignment 4 (due Friday, Nov 2 at 11:55pm)

Instructions

Rubric: {mechanics:3}

The above points are allocated for following the general homework instructions on the course homepage.

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 $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

1. $\frac{\delta^2 f(w)}{\delta w^2} = 2 > 0$ so, this is convex
2. $\frac{\delta^2 f(w)}{\delta w^2} = \frac{1}{w^2}$ and since w>0 the second derivative >0 so this is convex
3. Since all norms are convex and $\lambda > 0$ and a positive constant times a convex function is convex this expression is convex
4. Set $z = -y_i w^T x_i$ then $\frac{\delta^2 f(w)}{\delta w^2} = \frac{e^z}{(1+e^z)^2}$ since this is greater than zero the function is convex
5. $\frac{\lambda}{2}\|w\|_1$ is convex since all norms are convex and lambda is non-negative. The first part is with the sum is convex since it can be written as $\sum_{i=1}^n [\max\{0, |w^T x_i - y_i|\} - \epsilon] = \sum_{i=1}^n |w^T x_i - y_i|\} - \epsilon]$ since norm is always greater equal to 0. Then the sum of norms which are convex functions are also convex thus the expression is convex

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.

nonZeros stand for the number of features used in the regression

logReg Training error 0.000 logReg Validation error 0.084 # nonZeros: 101

logRegL2: The number of gradient descent iterations is: 36 logRegL2 Training error 0.002 logRegL2 Validation error 0.074 # nonZeros: 101

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).

The number of gradient descent iterations is: 78 logRegL1 Training error 0.000 logRegL1 Validation error 0.052 # nonZeros: 71

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.

Training error 0.000 Validation error 0.020 # nonZeros: 25

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!

L2 as discussed in the class did not use feature selection but minimized the weights. It achieved a higher validation error than without-regulation since it decreased the overfitting. L1 did some feature selection yet it achieved a perfect training score and a lower validation error than L2 since it further decreased overfitting by feature selection. L0 took a lot of time to run because of the iterations yet it achieved a perfect training error and a whopping 2% validation error owing to the increase in regularization.

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_`.

scikit L2 nonZeros: 101 Training error 0.002 Validation error 0.074

My results logRegL2 Training error 0.002 logRegL2 Validation error 0.074 # nonZeros: 101

The results are the same for L2-regularization

scikit L1 # nonZeros: 71 Training error 0.000 Validation error 0.052

My results logRegL1 Training error 0.000 logRegL1 Validation error 0.052 # nonZeros: 71

The results are the same for L1-regularization

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

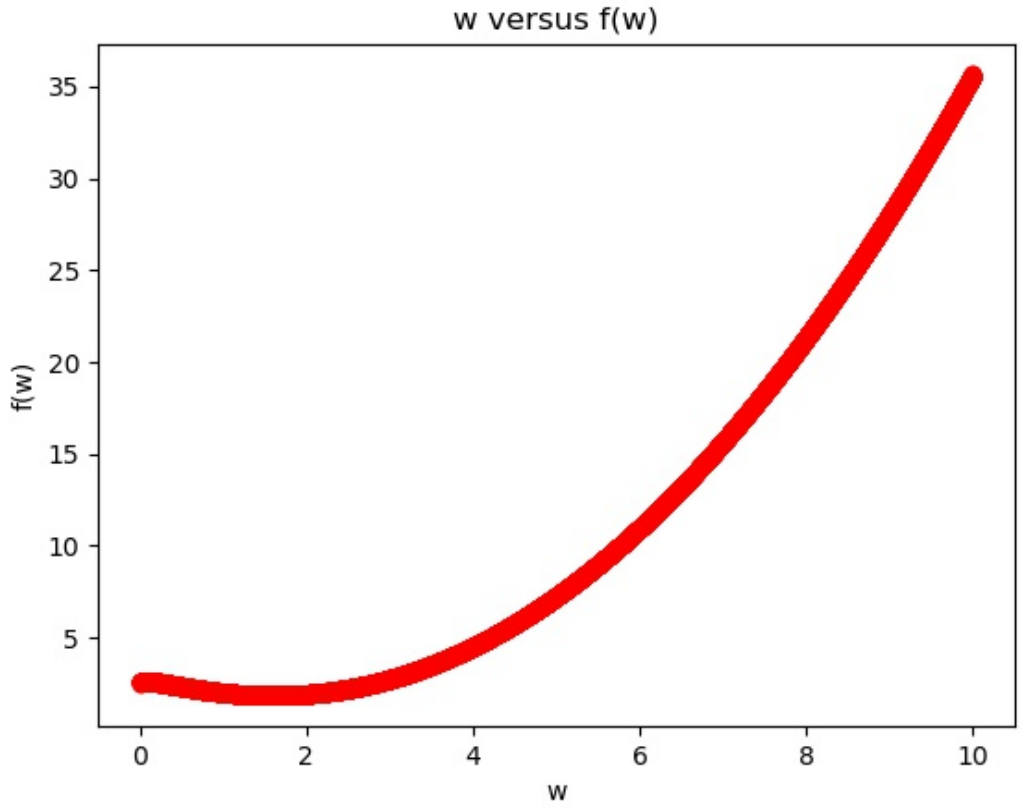
$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2 + \lambda \sqrt{|w|}\,.$$

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

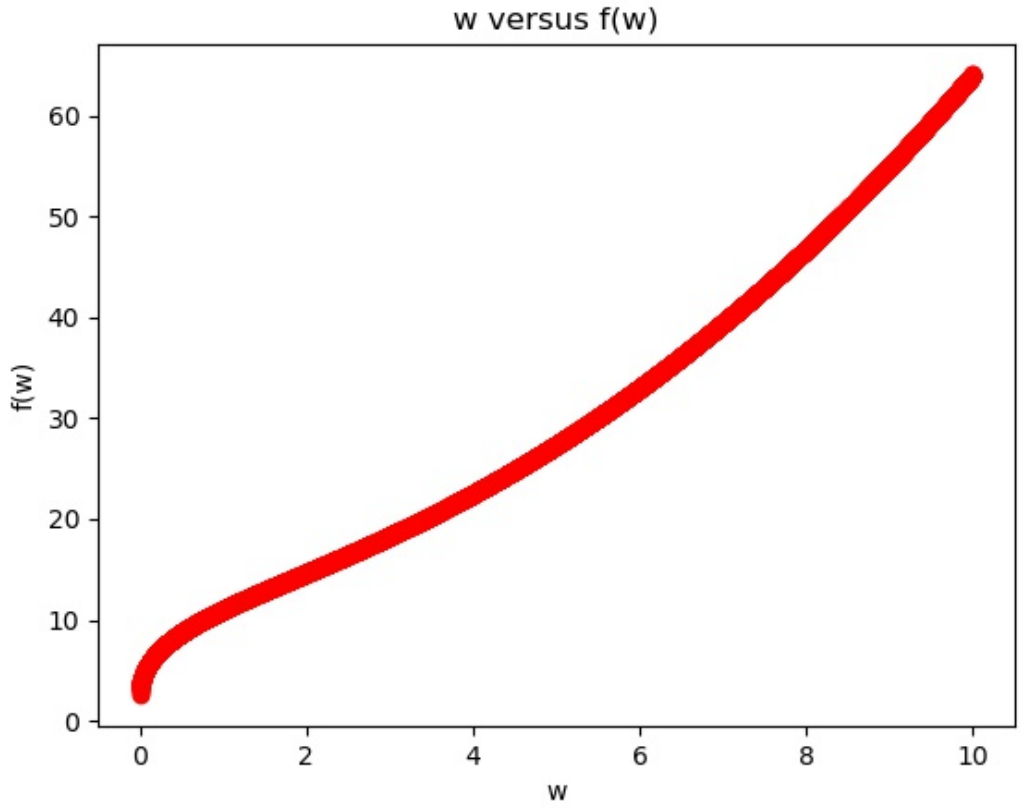
1. Plug in the data set values and write the loss in its simplified form, without a summation.

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.

1. $\frac{1}{2}[(w-2)^2 + 2\lambda\sqrt{|w|} + 1]$
2. we are solving $\frac{1}{2}((w-2)^2 + 1)$ then the gradient is w-2 setting it equal to zero we get $argmin f(w) = 2$
3. as lambda goes to infinity it becomes infinitely costly to use any weight thus argmin f(w) = 0 in this case
4. when lambda =1, min happens at w= 1.6



5. when lambda =10, min happens at w= 0



6. It behaves like L1 since it set the weight to 0 once lambda is 10. L2 never sets weights to zero(feature selection)

7. The first part is convex as shown previously in the first question since the second derivative is 2 >0. Now the second part $\lambda\sqrt{|w|}$ say $|w| = zforsomenon - negativez$ Then the second derivative of this function is $\frac{-\lambda}{4z^(2/3)}$ which is less than 0 thus this part is concave. Thus this is not a convex optimization problem

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.)

The different values we get by dot producting the rows by this example respectively is 1, 0, 2. Since 2 is greater than 1 and 0 we can deduce that the test example will be classified as the second class(assuming the classes are 0-indexed)

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

logLinearClassifier Training error 0.084 logLinearClassifier Validation error 0.070

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

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

$f(w) = \sum_{i=1}^{n} \sum_{j=1}^{d} -w_{y_{i,j}} x_{i,j} + log(\sum_{c=1}^{k} exp(\sum_{j=1}^{d} w_{c,j} x_{i,j})$ then

$\frac{\delta f(w)}{\delta w_{c,j}} = \sum_{i=1}^{n} -I(y = c) x_{i,j} + \frac{\frac{\delta}{\delta w_{c,j}} exp(\sum_{j=1}^{d} w_{c,j} x_{i,j})}{\sum_{c=1}^{k} exp(\sum_{j=1}^{d} w_{c,j} x_{i,j})}$

define $\gamma = \sum_{j=1}^{d} w_{c,j} x_{i,j}$

$= \sum_{i=1}^{n} x_{i,j}(-I(y = c) + \frac{exp(\gamma)}{\sum_{c=1}^{k} exp(\gamma)})$

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.

Training error 0.000 Validation error 0.008

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).

Setting the C value to np.inf I get Training error 0.000 Validation error 0.016

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.

1. $O(n^2d + n^3kT)$ because we first create the matrix Z by n^2d operations since each element in n has d features and the values are populated by using each n against other n's and taking the norm. Secondly, we populate our gradient matrix in n^3kT times since fun object runs n^2 times to create the gradient and each element has n features because of the radial basis. This happens T times.
2. $O(t^2d + t^2k)$ because we first create the RBF matrix by t^2d operation as explained before. Then we multiply t test examples which is a matrix of size t x t and weights which is a matrix of size t x k which takes t^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?

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?
3. What is a setting where you would use the L1-loss, and what is a setting where you would use L1-regularization?
4. Among L0-regularization, L1-regularization, and L2-regularization: which yield convex objectives? Which yield unique solutions? Which yield 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?
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.
7. Suppose a binary classification dataset has 3 features. If this dataset is "linearly separable", what does this precisely mean in three-dimensional space?
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?
9. What are "support vectors" and what's special about them?
10. What is a disadvantage of using the perceptron algorithm to fit a linear classifier?
11. Why we would use a multi-class SVM loss instead of using binary SVMs in a one-vs-all framework?
12. How does the hyper-parameter σ affect the shape of the Gaussian RBFs bumps? How does it affect the fundamental tradeoff?

1. We do not have infinite data so it is hard to know which exact features are relevant. We could use L0 and that would give us a sparse matrix of features yet that would not be unique. We could use L1 this would not be as sparse as L0 regularization and not unique. And even elastic net is unique and sparse but it is not as sparse as L0. Thus, we cannot say for sure which features are relevant.
2. If each feature was predicting the y making an independent error of the rest of the features then we would expect our error to go down as we have more features. But if these features do not have independent errors then our error will not decrease even though each feature is above the prediction threshold.
3. I would use L1-loss when there are outliers in the dataset and I want my regression to be robust against them. I would use L1-regularization when I want to select relevant features and I want to decrease overfitting
4. Convex objectives: L2 and L1 regularization Unique solution: L2 regularization Sparse solution: L0 and L1 regularization
5. As λ increases the sparsity level goes up because we are punishing the error function more for picking up weights. As we increase λ the training error goes up but test error goes down. I would think the test error starts increasing after a while along with the training error because of over-regularization(we cannot even use relevant information because of too high lambda)
6. Since it does not produce false positives we can deduce that the explained feature selection method probably uses L0 regularization because L1 produces false positives. Thus we can use a method similar to boLasso for L0 norm such as we use bootstrapping and do feature selection using L0-regularization

in each bootstrapped sample. Then we can either use all these features or using these features we can pick the ones that survive a regression with L1 regularization

7. This means that we can fit a plane between these points in the feature space such that the points in each half-spaces are of the same class
8. Because this leads to punishing some correct guesses for being too correct. Meaning if the correct label for an instance is 1 and our score is 100. This will add 99^2 to the loss function
9. Support vectors occur with the hinge loss when $y_i w^T x_i < 1$. They determine where the boundary is and if something is not a support vector its loss is zero and it does not contribute
10. Perceptron can only classify linearly separable set of data points. Thus if our dataset is not linearly separable, perceptron will not be able to perfectly classify them
11. Because when we use binary SVM, each weight is trying independently to predict its weight right, so we cannot guarantee that our method will generate the right weights collectively
12. σ determines the width of the Gaussian RBF bumps. As σ increases the width increases. Increasing σ increases training error yet reduces overfitting thus reduces the test error