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

We are providing solutions because supervised learning is easier than unsupervised learning, and so we think having solutions available can help you learn. However, the solution file is meant for you alone and we do not give permission to share these solution files with anyone. Both distributing solution files to other people or using solution files provided to you by other people are considered academic misconduct. Please see UBC's policy on this topic if you are not familiar with it:
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,959`
`http://www.calendar.ubc.ca/vancouver/index.cfm?tree=3,54,111,960`

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.

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).

   Answer: The first derivative is $f'(w) = 2\alpha w - \beta$ and the second derivative is $f''(w) = 2\alpha$, which implies convexity since $\alpha > 0$.

2. $f(w) = -\log(\alpha w)$ with $\alpha > 0$ and $w > 0$ ("negative logarithm")

   Answer:
   $$f'(w) = -\frac{1}{w},$$
   $$f''(w) = \frac{1}{w^2},$$
   which implies convexity because $w > 0$.

3. $f(w) = \|Xw - y\|_1 + \frac{\lambda}{2}\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized robust regression).

   Answer: We have $\|w\|_1$ is convex because it's a norm, and $\lambda \geq 0$ so $\lambda\|w\|_1$ is convex. The function $\|Xw - y\|_1$ is convex because we're composing the (convex) L1-norm $\| \cdot \|_1$ with an affine function $Xw - y$. Finally $f(w)$ is the sum of these two convex functions so it's convex.

4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).

Answer: The function $-y_i w^T x_i$ is linear, so we just just need to show that the log-sigmoid function $g(z) = \log(1 + \exp(z))$ is convex to show that each term is convex. It will then follow because the sum of convex functions is convex. To show that the log-sigmoid functions is convex, note that

$$g'(z) = \frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)},$$

and

$$g''(z) = -\frac{1}{(1+\exp(-z))^2}\frac{d}{dz}\exp(-z) = \frac{\exp(-z)}{(1+\exp(-z))^2)} = \frac{1}{1+\exp(-z)}\frac{\exp(-z)}{1+\exp(-z)} = h(-z)h(z),$$

where h is the sigmoid function. Since the sigmoid function is always positive, we've shown that $g(z)$ is convex.

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).

   Answer: First we note that $w^T x_i - y_i$ is a linear function and the composition of the convex $|\cdot|$ with a linear function would be convex. The number ϵ is constant so is convex. Since 0 is also convex, and the max of convex functions is convex, the max inside the sum is convex. Since $\lambda > 0$ and squared-norm are convex, $(\lambda/2)\|w\|^2$ is convex. For here, f is a sum of functions we've already shown are convex, so f is convex.

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

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 I humbly suggest `lammy`, which is what my 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: With L2-regularization, the validation error decreases to 0.074 but the number of non-zeroes stays at 101. Gradient descent iterations go down from 121 to 36 so decreases by 85 (the precise numbers may vary a bit). Training error went from 0 to 0.002.

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: With L1-regularization, the validation error decreases to 0.052 and the number of non-zeroes decreases to 71. Solving requires 78 iterations. Training error is 0.

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: With L0-regularization, the validation error decreases to 0.038 and the number of non-zeroes decreases to 24. You might have gotten a slightly different answer depending on versions of Python and libraries. We've seen validation errors of 0.018. Training error is 0.

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: L2 does not introduce any sparsity (zeros), while L1 introduces some and L0 even more. The sparsity seems to help with overfitting as the validation error goes down for each subsequent method. However, L0-regularization requires nested loops resulting in a much slower algorithm.

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: We end up with exactly the same results (at least, when printed out to 3 decimal places), which is satisfying.

2.6 $\text{L}\frac{1}{2}$ 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 "$\text{L}\frac{1}{2}$ regularization" (in quotation marks because the "$\text{L}\frac{1}{2}$ 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.

   Answer:

   $$f(w) = \frac{1}{2}(w - 2)^2 + \lambda\sqrt{|w|} .$$

   (The second data term goes away because it is independent of w.)

2. If $\lambda = 0$, what is the solution, i.e. $\arg\min_w f(w)$?

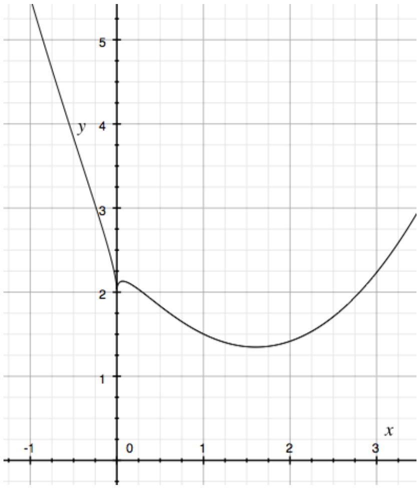
   Answer: $w = 2$

3. If $\lambda \to \infty$, what is the solution, i.e., $\arg\min_w f(w)$?

   Answer: $w = 0$

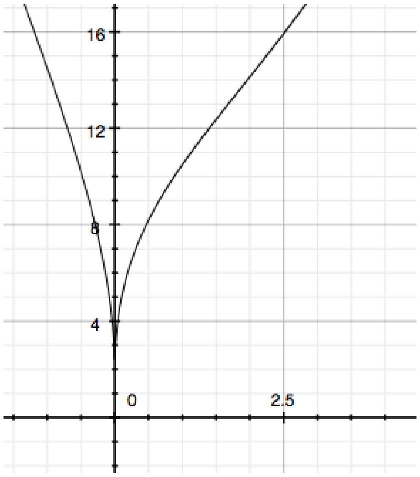
4. Plot $f(w)$ when $\lambda = 1$. What is $\arg\min_w f(w)$ when $\lambda = 1$? Answer to one decimal place if appropriate.

   Answer: See below. The solution is somewhere around $x = 1.6$.



5. Plot $f(w)$ when $\lambda = 10$. What is $\arg\min_w f(w)$ when $\lambda = 10$? Answer to one decimal place if appropriate.

   Answer: The solution is exactly 0. See below.



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.

   Answer: For sufficiently large finite values of λ we get the coefficient to be exactly 0, so it seems more like L1.

7. Is least squares with L$\frac{1}{2}$ regularization a convex optimization problem? Briefly justify your answer.

   Answer: No, from the plot when $\lambda = 1$ we can see already that it's not convex, even for this very simple case of $d = 1$ and $n = 1$ (for all intents and purposes). So the problem is in general not convex.

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 $y_i \in \{1, 2, \dots, k\}$. 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: This model bases its decision of maximizing the inner-product, $w_c^T \hat{x}$. For class 1 we have

$$w_1^T \hat{x} = (+2)1 + (-1)1 = 1.$$

For class 2 we have

$$w_2^T \hat{x} = (+2)1 + (-2)1 = 0.$$

For class 3 we have

$$w_3^T \hat{x} = (+3)1 + (-1)1 = 2.$$

So this model would predict '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: This decreases the error from around 0.13 down to around 0.07.

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: The loss for one training example is:

$$-w_{y_i}^T x_i + \log\left(\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)\right)$$

The derivative with respect to a particular W_{cj} is given by

$$-I(y_i = c)x_{ij} + \frac{\exp(w_c^T x_i)}{\sum_{c'=1}^{k} \exp(w_{c'}^T x_i)} x_{ij}.$$

For the first term, we got the indicator function because the gradient is zero except when $c = y_i$; that's the only time where w_c actually appears in the loss. For the second term, we applied the chain rule to the log, and then the chain rule again to the inner term to get the x_{ij} on the outside. This expression can be simplified even further by factoring out x_{ij} and noticing that the softmax probability appears in the second term,

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

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: The validation error depends on how precisely you solve the optimization problem, but it decrease to something very small like 0.01 or 0.02.

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: I get exactly the same results for one-vs-all, but a validation error of 0.016 for softmax (instead of 0.008). There's no sense reading too much into these numbers, I think. For example if you change the number of iterations of optimization then you get slightly different results. But, they are quite similar overall which is encouraging.

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?

   Answer: Forming the basis Z costs $O(n^2d)$, to compute $O(n^2)$ values of the $O(d)$-cost distance function. Training the model involves T iterations of gradient descent. Each iteration involves computing the function value and gradient over all n examples. To evaluate the function for one example, the dominant cost is computing $v_{c'}^T z_i$ for all k values of c', each of which costs $O(n)$. Thus evaluating the function for one example costs $O(nk)$, and the gradient has the same cost. Putting everything together gives $O(n^2d + n^2kT)$.

2. What is the cost of classifying the t test examples?

Answer: At test time the largest costs are computing $\tilde{Z}$ and computing $\tilde{Z}W$. Computing $\tilde{Z}$ costs $O(ndt)$ to compute the nt distance functions. Computing $\tilde{Z}W$ takes $O(tnk)$ using a standard implementation of matrix multiplication (technically, we could do this operation slightly faster using fast matrix multiplication methods.). The total cost is thus $O(ndt + tnk)$.

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.

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?

   Answer: "Relevance" can be fundamentally ambiguous and can also be non-intuitive (e.g., non-causal).

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?

   Answer: Doesn't model dependencies between variables (mutual information may also be tough to compute for continuous features/labels).

3. What is a setting where you would use the L1-loss, and what is a setting where you would use L1-regularization?

   Answer: L1-loss: outliers, L1-regularization: irrelevant features.

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

   Answer: L1- and L2- are convex, L2- is unique, L0- and L1- are sparse.

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?

   Answer: As λ goes up the solution becomes more sparse, the training error goes up, and the approximation error goes down.

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.

   Answer: Apply the method to bootstrap samples, take the union.

7. Suppose a binary classification dataset has 3 features. If this dataset is "linearly separable", what does this precisely mean in three-dimensional space?

   Answer: There exist hyper-planes where all the points in one class are on one side, and all the points in the other class are on the other side.

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?

   Answer: Non-convex/non-continuous (hard to minimize unless linearly separable)

9. What are “support vectors” and what’s special about them?

   Answer: Support vectors are training examples that have nonzero loss in an SVM. They are special in that they are the only examples needed to “support” the fit; all other examples could be removed with the same result.

10. What is a disadvantage of using the perceptron algorithm to fit a linear classifier?

    Answer: Only converges for linearly separable, doesn’t find max-margin, degenerate objective function.

11. Why we would use a multi-class SVM loss instead of using binary SVMs in a one-vs-all framework?

    Answer: One-vs-all doesn’t calibrate the w_c so that the correct class is the largest one.

12. How does the hyper-parameter σ affect the shape of the Gaussian RBFs bumps? How does it affect the fundamental tradeoff?

    Answer: Controls width (variance) of the bumps. Smaller σ yields lower training error and higher approximation error.