Machine Learning & Data Mining
Set 2

Caltech CS/CNS/EE 155
January 17th, 2019

1 Comparing Different Loss Functions [30 Points]

Relevant materials: lecture 3

Problem A [3 points]: Squared loss is often a terrible choice of loss function to train on for classification problems. Why?

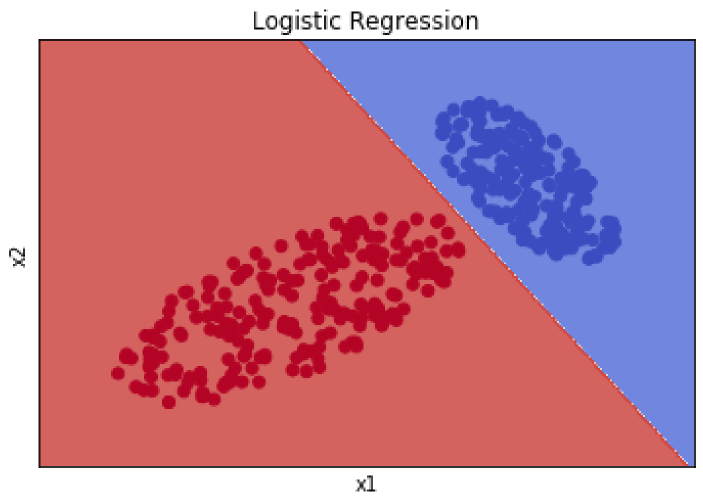
Solution A: *This is because squared error adjusts too hard for points that are very far away from the desired decision boundary. Furthermore, the squared loss function is a real valued function so it's output is fundamentally different from pure classification.*

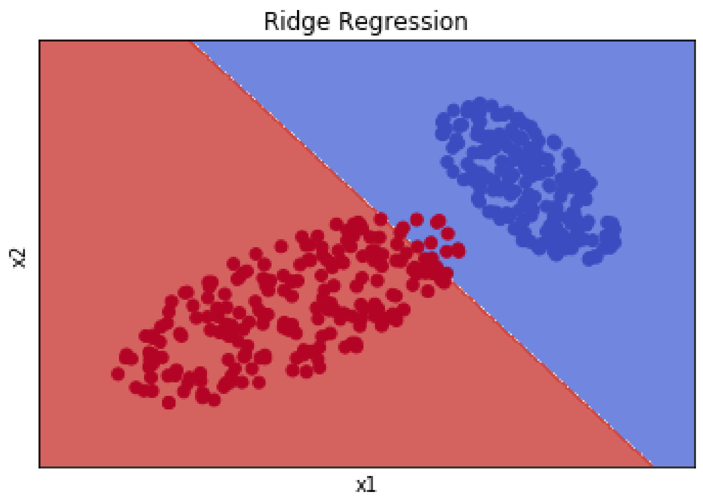
Problem B [9 points]: A dataset is included with your problem set: `problem1data1.txt`. The first two columns represent x_1, x_2, and the last column represents the label, $y \in \{-1, +1\}$.

For each loss function/model, plot the data points as a scatter plot and overlay them with the decision boundary defined by the weights of the trained linear classifier. Include both plots in your submission. The template notebook for this problem contains a helper function for producing plots given a trained classifier.

What differences do you see in the decision boundaries learned using the different loss functions? Provide a qualitative explanation for this behavior.

Solution B:





Logistic regression gets a decision boundary that splits the two clusters of data points. Squared loss, on the other hand, has a decision boundary that cuts through one of the clusters. The reason this occurs is one of the clusters is more spread out in the x direction. This means the squared loss function will adjust heavily to the points that are at the far edge of the spread cluster so the decision boundary shifts toward the spread cluster rather than having a more accurate decision boundary.

Problem C [9 points]: Leaving squared loss behind, let's focus on log loss and hinge loss. Consider the set of points $S = \{(\frac{1}{2}, 3), (2, -2), (-3, 1)\}$ in 2D space, shown below, with labels $(1, 1, -1)$ respectively.

Given a linear model with weights $w_0 = 0, w_1 = 1, w_2 = 0$ (where w_0 corresponds to the bias term), compute the gradients $\nabla_w L_{\text{hinge}}$ and $\nabla_w L_{\text{log}}$ of the hinge loss and log loss, and calculate their values for each point in S.

Solution C:

$\nabla_w L_{hinge} = -y\mathbf{x}$ *when* $1 - y\mathbf{w}^T\mathbf{x} > 0$
Otherwise, $\nabla_w L_{hinge} = 0$
$\nabla_w L_{log} =$

$$\frac{-y\mathbf{x}}{1 + e^{y\mathbf{w}^T\mathbf{x}}}$$

Points	*Log Gradient*	*Hinge Gradient*
(1/2, 3)	*(-0.38, -0.19, -1.13)*	*(-1, -0.5, -3)*
(2, -2)	*(-0.12, -0.24, 0.24)*	*(0, 0, 0)*
-(3, 1)	*(-0.95, 2.86, -0.95)*	*(0, 0, 0)*

Problem D [4 points]: Compare the gradients resulting from log loss to those resulting from hinge loss. When (if ever) will these gradients converge to 0? For a linearly separable dataset, is there any way to reduce or altogether eliminate training error without changing the decision boundary?

Solution D: *Both gradients converge to 0 as* $y\mathbf{w}^T\mathbf{x}$ *goes to greater or equal to 1 for hinge loss and inifinity for log loss because the gradient of* L_{hinge} *will go to zero because its loss function will go to zero and the gradient of* L_{log} *will go to zero beacuse its denomintor will go to zero. You can reduce* L_{log} *by making* $\mathbf{w}$ *itself times a large scalar multiple. This will make the gradient approach* $\ln(1) = 0$ *but never reach it while maintaining the decision boundary.* L_{hinge} *can be reduced and eliminated because making* $\mathbf{w}$ *itself times a large scalar multiple will make* $y\mathbf{w}^T\mathbf{x} > 1$ *for every point which means loss will be 0. Again the decision boundary is preserved in this case.*

Problem E [5 points]: Based on your answer to the previous question, explain why for an SVM to be a "maximum margin" classifier, its learning objective must not be to minimize just L_{hinge}, but to minimize $L_{\text{hinge}} + \lambda\|w\|^2$ for some $\lambda > 0$.

(You don't need to prove that minimizing $L_{\text{hinge}} + \lambda\|w\|^2$ results in a maximum margin classifier; just show that the additional penalty term addresses the issues of minimizing just L_{hinge}.)

Solution E: *We need the* $\lambda\|w\|^2$ *term becasue* L_{hinge} *can classify the points sufficiently and it will have a margin, but it isn't guarenteed to be the maximum margin. The margin is* $2/\|w\|^2$ *so maximizing the margin is the same thing as minimizing* $\|w\|^2$*. Thus, we need the* $\lambda\|w\|^2$ *term to guarentee we get the maximal margin when* L_{hinge} *is minimized.*

2 Effects of Regularization

Relevant materials: Lecture 4

Problem A [4 points]: In order to prevent over-fitting in the least-squares linear regression problem, we add a regularization penalty term. Can adding the penalty term decrease the training (in-sample) error? Will adding a penalty term always decrease the out-of-sample errors? Please justify your answers. Think about the case when there is over-fitting while training the model.

Solution A: *Adding a penalty term cannot decrease the training error because is will make the model less complex. This means it cannot fit the data as perfectly so the training error will usually increase and cannot decrese. Lessening complexity can never make the model fit the data better so training error will never decrease. Out of sample error, on the other hand, usually decreases. This doesn't mean it always decreases. It is possible to have a model that classifies data perfectly so adding regularization to that model couldn't lower out of sample error and could even increase if the model is so simple that it is underfit. In fact making regularization very strong will add bias to your model and almost certainly increase out of sample and in sample error.*

Problem B [4 points]: ℓ_1 regularization is sometimes favored over ℓ_2 regularization due to its ability to generate a sparse w (more zero weights). In fact, ℓ_0 regularization (using ℓ_0 norm instead of ℓ_1 or ℓ_2 norm) can generate an even sparser w, which seems favorable in high-dimensional problems. However, it is rarely used. Why?

Solution B: *The ℓ_0 norm is rarely used because it is not continuous. It's a flat and discontinuous regularization penalty. This means it is very hard to optimize.*

Implementation of ℓ_2 regularization:

Problem C [16 points]: Do the following for both training data sets (wine_training1.txt and wine_training2.txt) and attach your plots in the homework submission (use a log-scale on the horizontal axis):

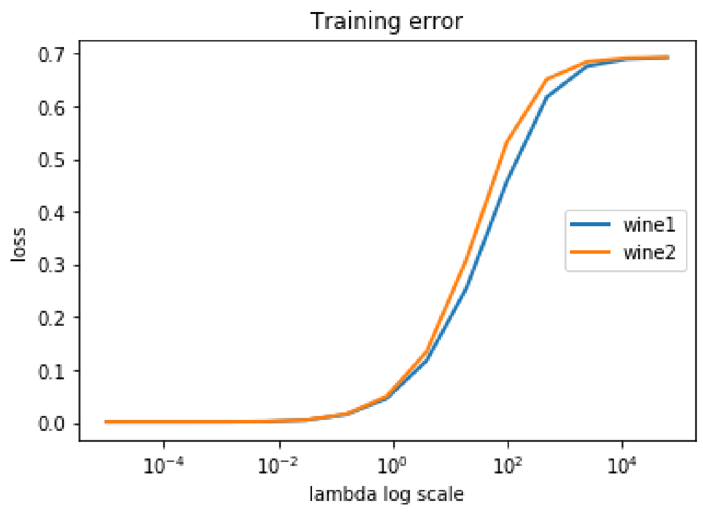
i. Plot the average training error (E_{in}) versus different λs.

ii. Plot the average test error (E_{out}) versus different λs using wine_testing.txt as the test set.

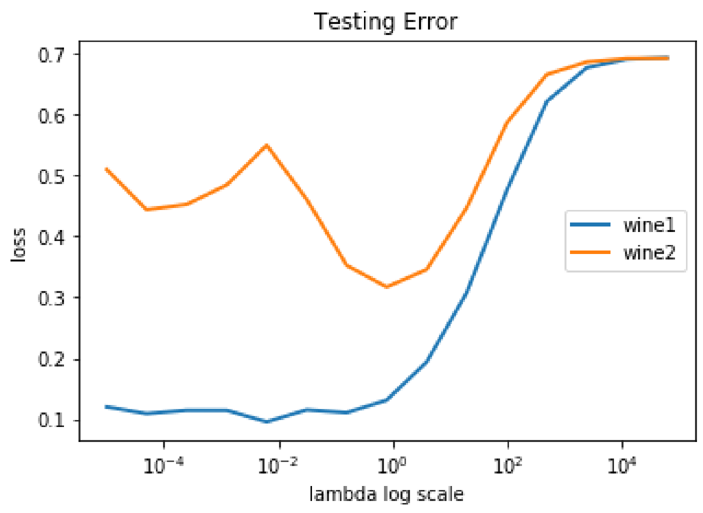
iii. Plot the ℓ_2 norm of **w** versus different λs.

You should end up with three plots, with two series (one for wine_training1.txt and one for wine_training2.txt) on each plot. Note that the E_{in} and E_{out} values you plot should not include the regularization penalty — the penalty is only included when performing gradient descent.

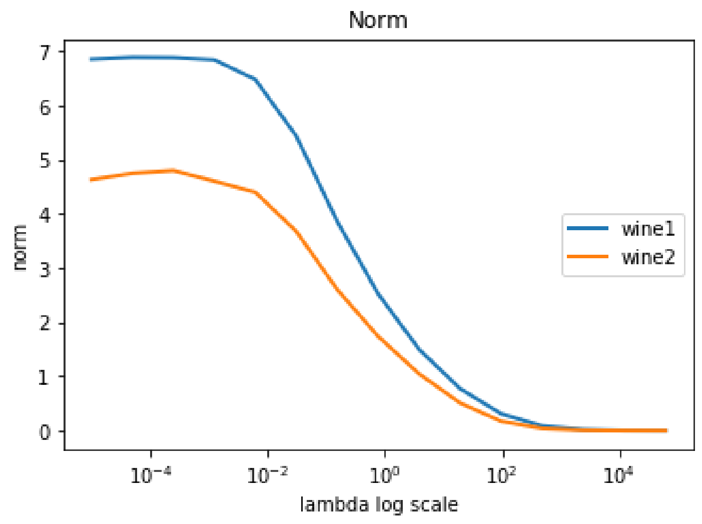
Solution C:



The E_in of the two data sets



The E_out of the two data sets



The norm of the two data sets

Problem D [4 points]: Given that the data in wine_training2.txt is a subset of the data in wine_training1.txt, compare errors (training and test) resulting from training with wine_training1.txt (100 data points) versus wine_training2.txt (40 data points). Briefly explain the differences.

Solution D: *The training errors are largely the same. This makes sense because the model should be able to minimize error when regularization is low regardless of the number of points. When regularization increases the number of points becomes less relevent because the model becomes so simple that the training error sky rockets. The testing error shows the model trained with more points generalizes better. This makes sense because, in general, the more points you have to train on, the better your model performs out of sample becasue the model can train on a larger set of points that more accurately reflects the true data. We can see an example of overfitting with sample set 2 because we see training error remain the same but testing error go up. You're more likely to overfit on a smaller amount of points. Testing error converges with very large reularization stength becasue both models become very simple and the training benefits no longer are relevant for data set 1.*

Problem E [4 points]: Briefly explain the qualitative behavior (i.e. over-fitting and under-fitting) of the training and test errors with different λs while training with data in wine_training1.txt.

Solution E: *As lambda increases past around 1 we can see underfitting start to occur as the training and testing error both start to increase in a similar manner. There is evidence of slight overfitting at small lambdas because* *we see testing error increases and training error remains at 0. The smallest testing error is around lambda =* 10^-1 *and smaller. Underfitting occurs after this.*

Problem F [4 points]: Briefly explain the qualitative behavior of the ℓ_2 norm of $\mathbf{w}$ with different λs while training with the data in wine_training1.txt.

Solution F: *The* ℓ_2 *norm of* $\boldsymbol{w}$ *is similar for very small theta, around 7, but as* λ *approaches around .01 the norm decreases dramatically until it hits 0 at large lambda. This makes sense because large norms are getting punished more heavily as* λ *increases.*

Problem G [4 points]: If the model were trained with wine_training2.txt, which λ would you choose to train your final model? Why?

Solution G: *I would choose* $\lambda = 1$ *because it has the lowest testing error which means the model with that level of regularization generalizes the best. We can clearly see the dip in testing error near that* λ. *It is the best balance between complexity and overfitting.*

3 Lasso (ℓ_1) vs. Ridge (ℓ_2) Regularization

Relevant materials: Lecture 3

For this problem, you may use the scikit-learn (or other Python package) implementation of Lasso and Ridge regression — you don't have to code it yourself.

The two most commonly-used regularized regression models are Lasso (ℓ_1) regression and Ridge (ℓ_2) regression. Although both enforce "simplicity" in the models they learn, only Lasso regression results in sparse weight vectors. This problem compares the effect of the two methods on the learned model parameters.

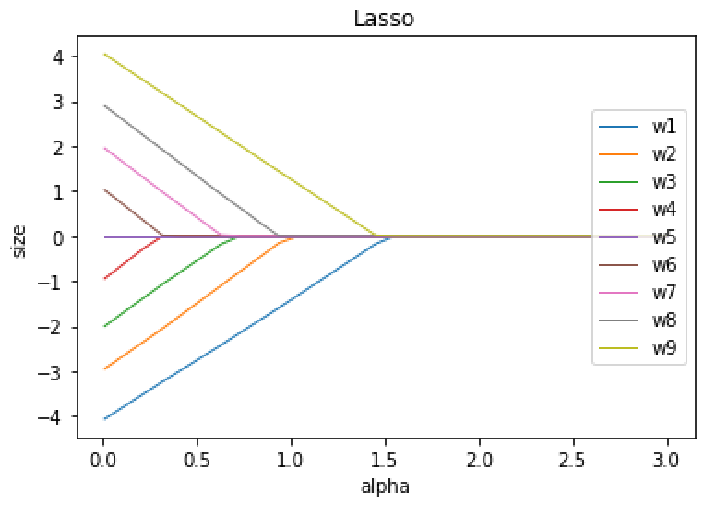
Problem A [12 points]: The tab-delimited file problem3data.txt on the course website contains 1000 9-dimensional datapoints. The first 9 columns contain $x_1, \ldots, x_9$, and the last column contains the target value y.

i. Train a linear regression model on the problem3data.txt data with Lasso regularization for regularization strengths α in the vector given by `numpy.linspace(0.01, 3, 30)`. On a single plot, plot each of the model weights $w_1, ..., w_9$ (ignore the bias/intercept) as a function of α.

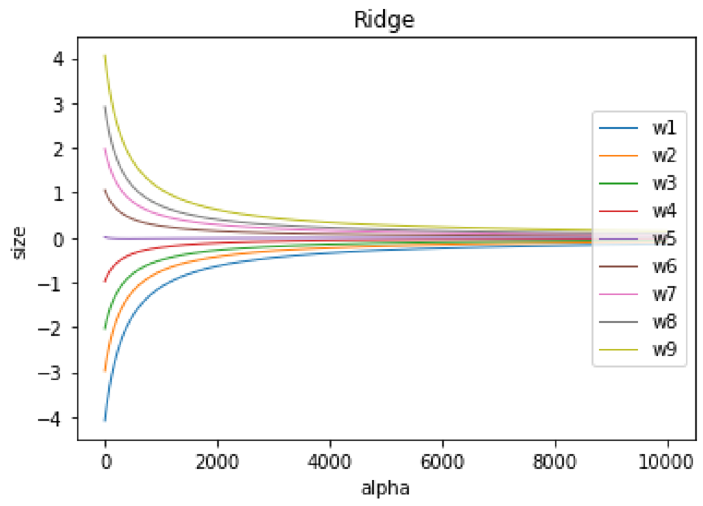
ii. Repeat **i.** with Ridge regression, and this time using regularization strengths $\alpha \in \{1, 2, 3, \ldots, 1e4\}$.

iii. As the regularization parameter increases, what happens to the number of model weights that are exactly zero with Lasso regression? What happens to the number of model weights that are exactly zero with Ridge regression?

Solution A:



Weights using lasso regression



Weights usisng Ridge regression As regularization increases the number of weights that are zero for lasso regression goes up and eventually becomes all the weights. With ridge regression none of the weights go to exactly zero, they all just approach zero.

Problem B [18 points]:

i. In the case of 1-dimensional data, Lasso regression admits a closed-form solution. Given a dataset containing N datapoints each with d features, where $d = 1$, solve for

$$\arg\min_{w} \|\mathbf{y} - \mathbf{x}w\|^2 + \lambda\|w\|_1,$$

where $\mathbf{x} \in \mathbb{R}^N$ is the vector of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values corresponding to these datapoints. Just consider the case where $d = 1$, $\lambda \geq 0$, and the weight w is a scalar.

This is linear regression with Lasso regularization.

ii. In this question, we continue to consider Lasso regularization in 1-dimension. Now, suppose that when $\lambda = 0$, $w \neq 0$. Does there exist a value for λ such that $w = 0$? If so, what is the smallest such value?

iii. Given a dataset containing N datapoints each with d features, solve for

$$\arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|_2^2$$

where $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the matrix of datapoints and $\mathbf{y} \in \mathbb{R}^N$ is the vector of all output values for these datapoints. Do so for arbitrary d and $\lambda \geq 0$.

This is linear regression with Ridge regularization.

iv. In this question, we consider Ridge regularization in 1-dimension. Suppose that when $\lambda = 0$, $w \neq 0$. Does there exist a value for $\lambda > 0$ such that $w = 0$? If so, what is the smallest such value?

Solution B: *i. We get the subgradient to get*
$0 = (-2\mathbf{x}^T)(y - \mathbf{x} * w) \pm \lambda$
Solving for w we get $w = \frac{2\mathbf{x}^T\mathbf{y} \pm \lambda}{\mathbf{x}^T\mathbf{x}}$
ii. Yes, as we can see there exists a lambda value that makes w = 0. We can see $\pm 2\mathbf{x}^T\mathbf{y}$ *makes w = 0 depending on the sign of* $\mathbf{x}^T\mathbf{y}$*. So the smallest* λ *value that makes w = 0 is* $|2\mathbf{x}^T\mathbf{y}|$
iii. Solving the gradient with respect to w we get $0 = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}w) + 2\lambda w$
Thus, we get $w = (\mathbf{X}^T\mathbf{X} + \lambda I_d)^{-1}\mathbf{X}^T\mathbf{y}$
iv. No, there isn't a $\lambda > 0$ *value for which w = 0. We know* $(\mathbf{X}^T\mathbf{X} + \lambda I_d)$ *is only 0 when* λ *is the negative eigenvalue of* $\mathbf{X}^T\mathbf{X}$*. This obviously cannot be the case because* $\lambda > 0$ *is a requirement for this problem. Thus, when we multiply two non zero matrices we will always get a non zero result.*