University of Burgundy

Masters in Computer Vision and Robotics

SSI Pattern Recognition

Homework 2

Linear Classification

by
Vamshi Kodipaka

Evaluator: Dr. Desire Sidibe

BUILDING A CLASSIFIER TO FILTER SPAM EMAILS

SPAM: The irrelevant or unsolicited messages sent over the Internet, typically to a large number of users, for the purposes of advertising, phishing, spreading malware, etc

1.1 Problem Statement:

We were given training data 'spamTrain.txt', 'spamTrainLabels.txt' and test data 'spamTest.txt', 'spamTrainLabes.txt' sets of emails.

Each email has been processed and a set of 57 features were extracted as described below.

- 48 features, in [0; 100], giving the percentage of words in a given message which match a given word on a predefined list (called vocabulary).
- 6 features, in [0; 100], giving the percentage of characters in the email that match a given character on the list.
- Feature #55: the average length of an uninterrupted sequence of capital letters.
- Feature #56: the length of the longest uninterrupted sequence of capital letters.
- Feature #57: the sum of the lengths of uninterrupted sequence of capital letters.

Expected: Design for a Classifier to Filter Spam Emails for the given problem Statement

The data is stored in *'data_training'* for training data, *'data_training_labels*' for training data labels, *'data_testing*' for test data and *'data_testing_labels*' for test data labels in the code.

NOTE:

1. Before Proceeding, Please find and observe a README.txt file in code folder.

2. Implementations steps explained here are corresponding functions in the code for easy access.

1.2.1 Finding max and mean of the average length of uninterrupted sequences of capital letters in the training set

See the function [data_max, data_mean] = getMaxMean(data, feature)

Mean and max can be calculated from 55th column of *'data_training'*.

```
------------%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%----------
Maximum of the average length of uninterrupted sequences of capital letters in the training set is 1102.5
Mean of the average length of uninterrupted sequences of capital letters in the training set is 4.9006
------------%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%----------
```

Fig. 1 Result of task 1.2.1

Output : The average length of uninterrupted sequences of capital letters

:: MEAN = 4.9 ; MAX = 1102.5

1.2.2 Find max and mean of the lengths of the longest uninterrupted sequences of capital letters in the training set

See the function [data_max, data_mean] = getMaxMean(data, feature)

Mean and max is calculated from 56th column of *'data_training'*.

```
----------%%%%%%%%%%%%%%%%%%%%%%%%----------
Maximum of the lengths of the longest uninterrupted sequences of capital letters in the training set is 9989
Mean of the lengths of the longest uninterrupted sequences of capital letters in the training set is 52.675
----------%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%----------
```

Fig.2 Result of task 1.2.2

Output: The lengths of the longest uninterrupted sequences of capital letters

:: MEAN = 52.67 ; MAX = 9989

1.2.3 Preprocessing steps

See the Function :: [data_processed, data_mean, data_std] = getPreProcessed(data , type)

1.2.3.1 To standardize the columns so they all have mean 0 and unit variance

In the above function, type = 1 performs standardize.

Normalization step for standardization:

$$for\ every\ feature\ x_i, \qquad we\ standarize\ by\ using\ mean\ and\ standard\ dev\ x_i \leftarrow \frac{x_i - \bar{x}_i}{\sigma_{x_i}}$$

Now, each feature is centered (zero mean) and standardized (unit variance).

The standardized data for train is stored in *'data_train_stand'*, and for test in *'data_test_stand'*.

1.2.3.2 Transforming the features by log(x_{ij} + 0.1)

In previous function, type = 2 for transformed features by above relation.

Now, transformed data is stored in *'data_training_trans'*, and for test in *'data_testing_trans'*.

1.2.3.3 Binarize the features using I(xij > 0)

In previous function, type = 3 is for binarizing features.

The binarized data for test is stored in '*data_binary'*, and for test in *'data_test_binary'*.

1.2.4 Fitting a Logistic Regression Model for each version of data

First-of-all, a column of ones is added to train and test data for bias term.

To fit the logistic regression model to data, we need to get the regularization parameter, λ.
Already we know that, regularization parameter to avoid over fitting[2]. So we use it here.
We use cross-validation (training data) to get the regularization parameter.

This is implemented by function
lambda = getlambda(data_training, data_training_lab)

Here implementation of this function, where it takes the training data and training data labels and it automatically calculates training error and low validation error and gives lambda as output.

Usually, Cross-validation is performed by taking:

- 80% of training data to train (called training data, here) and
- 20% of training data to validate (called validation data, here)

We define a range of lambda to get the minimum validation and train error for data.

Procedural Steps:

1. For each lambda value
   a. Firstly, get the regression params
      This is implemented by function
      regs_para = findRegParma(data, labels, lambda)

      To get the regression params(w), we use MATLAB built-in function 'fminunc' to get optimal value of regression parameters. So, function requires a cost function to optimize.

      A function when given the training set and a particular 'w', computes the cost function and the gradient with respect to 'w' for the dataset [2].

      This cost function is implemented by [e grad] = costFunction_regu(X, y, w, lambda)

      The cost function is defined as,

$$E(w) = -\sum_{n=1}^{N}\{y_n \log(\phi_n) + (1-y_n)\log(1-\phi_n)\} + \frac{\lambda}{2}\sum_{k=1}^{M} w_k^2$$

We do not regularize the bias term w_0.

where, $\phi_n = y(\phi_n) = \sigma\left(w^T \phi(x_n)\right)$ with, $\sigma(x) = \frac{1}{1+e^{-a}}$ is sigmoid function.

The gradient of the cost function is a vector where the i^{th} element is defined as:

if $i = 0$

$$\frac{\partial E(w)}{\partial w_0} = \sum_{n=1}^{N} (y(\phi_n) - y_n)\phi_n$$

else

$$\frac{\partial E(w)}{\partial w_i} = \sum_{n=1}^{N} (y(\phi_n) - y_n)\phi_n + \lambda w_i$$

b. Now, we predict the class for validation set and training set. [y] = predict(X,w)

where we pass $w^T X$ to the sigmoid function to get the prediction.

As sigmoid function values are between 0 and 1, values can be interpreted as probabilities.

$$\sigma(s) > 0.5 \rightarrow s \; belongs \; to \; class - I$$
$$\sigma(s) < 0.5 \rightarrow s \; belongs \; to \; class - II$$

c. Now, compare with the true validation labels and train labels to get validation error and train error respectively. And this, error is stored for each lambda

2. After getting all errors for all lambda values, the minimum error is chosen and that particular index lambda value is returned by the program.

After getting regu-param, we compute the reg-parms on the full training data and get the reg-params. [as explained above in (1.a)].

***

Getting Training-Error:

After getting the regression parameters from training data, we apply it on training data to get training error. error_percent = findError(data, labels, regs_para)

Here, we predict the class by using training data and reg-parmas and assign a class as explained in (1.b). Atlast, we compare predicted class to the true class as we need to find the error in prediction.

Getting Test-Error: The same method above is applied on test data to get the test error.

1.2.5 Results on logistic regression

1.2.5.1 How to choose lambda for standardized data

Testing various values of lambda to get the validation and train errors. The result is as shown in Fig. 3

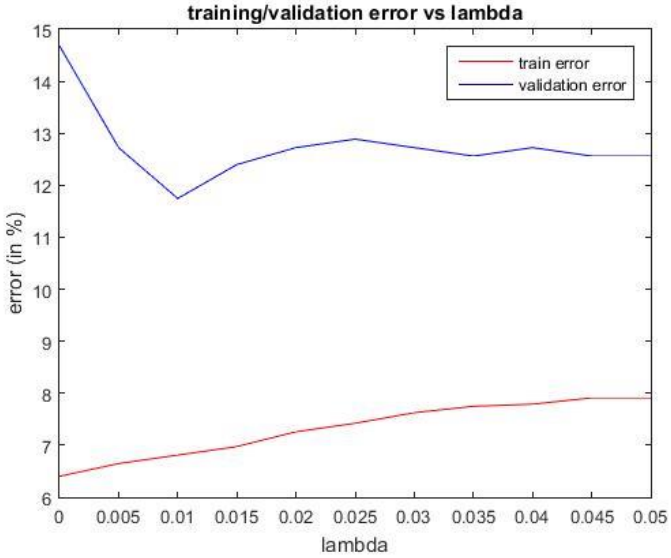

Fig. 3 Train and validation error for standardized data (to get lambda)

From the Fig.3, for lambda value of 0.01, the validation and train errors are optimal.
So lambda we choose for standardized data is 0.01.

	Train Error (in %)	Validation Error (in %)
Lambda = 0.01	6.81	11.74

1.2.5.2 How to chose lambda for transformed data

(Same as above procedure)::
The result is in Fig. 4.

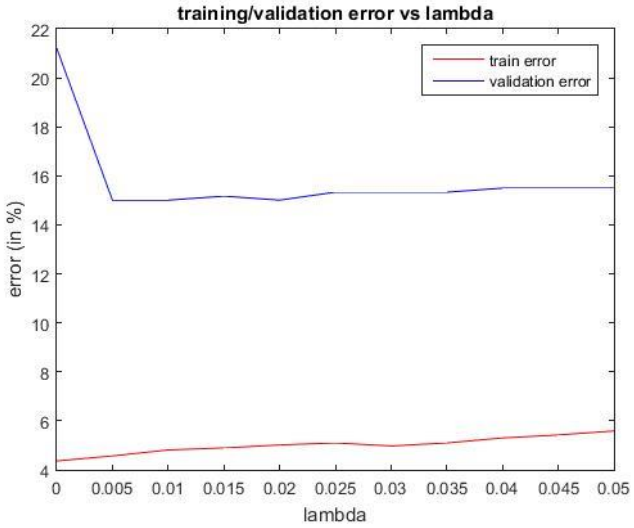

Fig. 4 Train and validation error for transformed data (to get lambda)

From the Fig.4, we can infer that for the lambda value of 0.005, the validation and train errors are optimal. So lambda for standardized data is chosen as 0.005.

	Train Error (in %)	Validation Error (in %)
Lambda = 0.01	4.56	15

1.2.5.3 How to chose lambda for binarizing data

(Same procedure as above)
The result is as shown in Fig. 5.

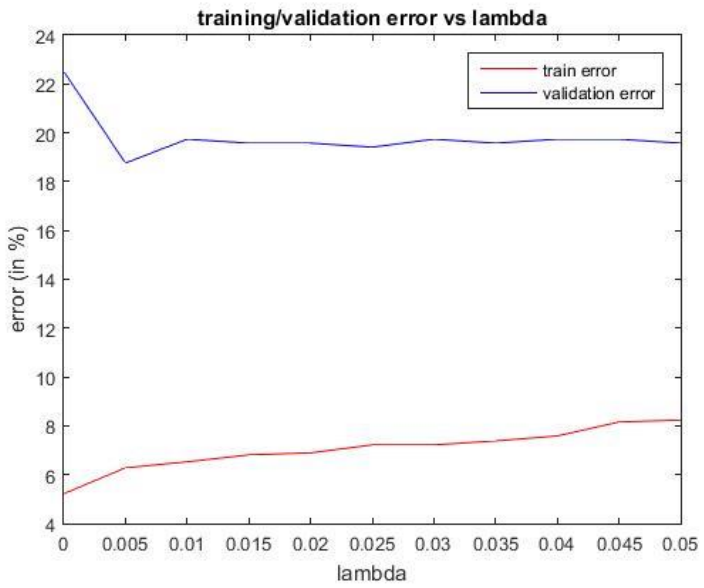

Fig. 5 Train and validation error for binarized data (to get lambda)

From the Fig.5, we can infer that for the lambda value of 0.005**, the validation and train errors are optimal.** So lambda chosen for standardized data is **0.005.**

	Train Error (in %)	Validation Error (in %)
Lambda = 0.01	6.28	18.76

1.2.6 Naive Bayes Classifier

The Naive Bayes Classifier is implemented by function
testing_lab_naive = getNaiveBayes(data_training, data_training_lab, data_test)

Theme: when we were given a test data to classify it, we will check the probability of the class-I given test data(X), $p(C_1|X)$ and probability of class-II given test data, $p(C_2|X)$ and assign a class which has greater probability of two.

$$p(C_1|X) \propto p(X|C_1)p(C_1)$$

where, $p(C_1)$ can be found from training data as,

$$p(C_1) = \frac{1}{N}\sum_{n=1}^{N} t_{n_{C_1}} = \frac{N_1}{N}$$

And $$p(X|C_1) = p(X_1, X_2, \dots, X_m \,|C_1)$$

by assumption of i.i.d, we get

$$p(X|C_1) = p(X_1|C_1)p(X_2|C_1) \ldots . p(X_m|C_1)$$

so,

$$p(C_1|X) \propto p(X_1|C_1)p(X_2|C_1) \ldots . p(X_m|C_1)p(C_1)$$

$$p(C_1|X) \propto \prod_{n=1}^{m} p(X_n|C_1)\, p(C_1)$$

here, each $p(X_i|C_1)$ can be a normal distribution $\mathcal{N}(X_i;\, \mu_1, \sigma_1) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-1}{2}\left(\frac{(X_i-\mu_1)}{\sigma^2}\right)^2}$

The same applies to class II to get $p(C_2|X)$

$$if\ p(C_1|X) \geq\ p(C_2|X) assign\ class\ I$$
$$else\ assign\ class\ II$$

1.2.6 Overall Train and Test Errors

DataType	λ	Train Error (in %)	Test Error (in %)			
			With preprocessing of test data	With preprocessing of test data using train data (only for method1)	Naive Bayes (my implementation)	Naive Bayes (Matlab)
Standadized Data	0.01	8.32	8.65	8.789	18.94	18.88
Transformed Data	0.005	5.18	6.05	-	18.099	18.099
Binarized Data	0.005	6.98	7.55	-	38.76	-

This table gives overall train and test errors using logistic-reg & Naive Bayes for all sets of data.

Note:
A common good practice in classification is to do feature normalization is of the features, i.e., center the data subtracting the mean & normalize it dividing by the variance (or standard deviation too). We do this for achieving two things:

1. Avoiding extra small model weights for the purpose of numerical stability.

2. Ensure quick convergence of optimization algorithms like e.g. Conjugate Gradient so that the large magnitude of one feature dimension w.r.t. the others doesn't lead to slow convergence.

If testing data is expected to vary from the training data, we can use the test data for normalization. This might slightly go into the direction of domain adaption. However, the applicability of the approach highly depends on whether our problem setting admits such a treatment (e.g. often in classification, we only have one test point at a time at our disposal, which renders the test mean approach inapplicable). So, in such case we take the mean and standard deviation from training data and apply it on testing data.

But, in this data, the transformed feature data (using log(x+0.1)) gives less error, as data in certain columns have high order of information, so when we apply logarithm to such data, it would give us less numbers as any other columns of data. So, now all the data would be in proper order to get better results.

References

[1] Problem statement of Homework2 by Dr. Desire Sidibe
[2] Lecture slides and Class Notes of Dr. Desire Sidibe
[3] Naïve Bayes Classifier: https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[4] Andrew Ng's Machine Learning: https://www.youtube.com/watch?v=qSF4eCa0j88
[5] Mr.GopiKrishna's Explanations:: https://github.com/gopi231091/Build-a-classifier-to-filter-spam-emails/tree/master/MatlabCode_Gopi

Thanks

Vamshi Kodipaka