CS 534: Machine Learning
Emory University
Fall 2024
Prof. Eugene Agichtein

Homework # 2

Mitchell Scott
(mtscot4)

1. **Written(10 pts) Bias-Variance Trade-off of LASSO** While it is hard to write the explicit formula for the bias and variance of using LASSO, we can quantify the expected general trend. Make sure you justify the answers to the following questions for full points:

   (a) (Written) What is the general trend of the bias as λ increases?
   Solution.

   (b) (Written) What about the general trend of the variance as λ increases?
   Solution.

   (c) (Written) What is the bias at $\lambda = 0$?
   Solution.

   (d) (Written) What about the variance at $\lambda = \infty$?
   Solution.

2. **Code+Written(40 pts) Spam classification using Naive Bayes and Standard Logistic Regression**

   (a) **(Code) You will explore the effects of feature preprocessing and its impact on Naive Bayes and Standard (unregularized) logistic regression. Write the following functions to preprocess your data. You are free to use the `sklearn.preprocessing` module. Note that they functions should be independent of one another and should not build on each step/call each other. You should assume only the features are passed, in and not the target. These functions should accept numpy 2darray as input, return the preprocessed train and test set in numpy 2D array format (i.e., two return values).**

      i. **function `do_nothing(train, test)` that takes a train and test set and does no preprocessing.**

      ii. **function `do_std(train, test)` that standardizes the columns so they all have mean 0 and unit variance. Note that you want to apply the transformation you learned on the training data to the test data. In other words, the test data may not have a mean of 0 and unit variance.**

      iii. **function `do_log(train, test)` that transforms the features using $\log(x_{ij} + 0.1)$ or a smoothed version of the natural logarithm.**

iv. **function `do_bin(train, test)` that binarize the features using $\mathbb{I}_{(x_{ij}>0)}$. (Note that $\mathbb{I}$ denotes the indicator function). In other words, if the feature has a positive value, the new feature is a 1, otherwise, the value a 0.**

Solution. Implimented in '`q2.py`'.

(b) **(Code) Write a Python function `eval_nb(trainx, trainy, testx, testy)` that fits a Naive Bayes model to the training. You can use `sklearn.naive_bayes` module for this part. The function should accept as input numpy 2d arrays, and return a dictionary containing the accuracy and AUC for the training and test sets and the predicted probabilities for the test set:**
`return "train-acc": train_acc, "train-auc": train_auc, "test-acc": test_acc, "test-auc": test_auc, "test-prob": test_prob.`
The values for accuracy and AUC should be scalar numeric values, while test-prob should be a numpy 1-d array with the predicted probability for the positive class 1, for the test.

Solution. Implimented in '`q2.py`'.

(c) **(Written) Fit a Naive Bayes model to each of the four preprocessing steps above using the code in 2b. Each preprocessing should be performed independently (i.e., use each of the functions you created in 2a on the original dataset). Report the accuracy rate of and AUC of NB on the training and test sets across the 4 preprocessing steps in a table.**

Solution.

(d) **(Code) Write a Python function `eval_lr(trainx, trainy, testx, testy)` that fits a ordinary (no regularization) logistic regression model. The function should return a dictionary containing theaccuracy and AUC for the training and test sets and the predicted probabilities for the test:**

`{"train-acc": train_acc, "train-auc": train_auc, "test-acc": test_acc, "test- auc": test_auc, "test-prob": test_prob}.`
Note that the values for accuracy and AUC should be scalar numeric values, while test-prob should either be a numpy 1-d array with the predicted probability of positive class 1 for the test set. The output will be the same format as 2b.

Solution. Implimented in '`q2.py`'.

(e) **(Written) Fit ordinary (no regularization) logistic regression model with each of the four preprocessing steps above using the code in 2d. Report the accuracy rate and AUC on the training and test sets for the 4 preprocessing steps in a table**

Solution.

(f) **(Written) Plot the receiver operating characteristic (ROC) curves for the test data. You should generate 3 plots:**

- **One plot containing the 4 Naive Bayes model curves representing each of the preprocessing steps.**

- **One plot containing the 4 logistic regression model curves representing each of the preprocessing steps.**
- **One plot containing the best Naive Bayes model and the best logistic regression model curve.**

Solution.

(g) **(Written) Given your results in 2c, 2e, and 2f, discuss how the preprocessing affects the models (Logistic and Naive Bayes) with regards to ROC, AUC, and accuracy. Also, comment on how Naive Bayes performance compares with logistic regression**

Solution.

3. **(50 pts) Exploring Model Selection Strategies for Logistic Regression with Regularization We will be using the SPAM dataset from the previous part for this problem. You can preprocess the data however you see fit, either based on the results of the previous problem or by introducing another preprocessing method. The only requirement is that it is consistent throughout the rest of this problem. For this problem, you are not allowed to use the `sklearn.model_selection` module. All the specified functions should be in the file '`q3.py`'.**

(a) **(Written) How did you preprocess the data for this problem? Why?**

Solution.

(b) **(Code) Implement the Python function `generate_train_val(x, y, valsize)` that given the validation size splits the data randomly into train and validation splits. The function should return a dictionary**
`return {"train-x": tr_x, "train-y": tr_y, "val-x": ts_x, "val-y": ts_y}.`
The values for '`train-x`' and '`val-x`' are expected to be numpy 2d arrays of the same dimension as x, and split into the associated training and validation features. The values for '`train-y`' and '`val-y`' are expected to be a subset of y split accordingly. Note that each time this function is run, the splits could be different.

Solution.

(c) **Code) Implement the Python function `generate_kfold(x, y, k)` that given the k, will split the data into k-folds. The function should return a single numpy 1-d array, containing the k that each item index it belongs to (e.g., array([0,1,2,. . . ,2,1,1])) for $k = 3$, which indicates item 0 belongs to fold 0, and item 1 belongs to fold 1, etc.**

Solution.

(d) **(Code) .**

Solution.

(e) **(Code) Implement the Python function`eval_holdout(x, y, valsize, logistic)` which takes in the input (e.g., 3000 training samples from spam.train.dat), the input labels, and the validation size and (1) uses 3b to split x and y into train and validation, and (2) evaluates the performance using the logistic regressionmodel passed in**

as . You can assume the logistic regression classifier will be created using `sklearn.linear_model.LogisticRegression` and initialized before passed to your method, so you can invoke it as usual. Your function should return a dictionary containing the accuracy and AUC for the training and validation sets using keys '`train-acc`', '`train-auc`', '`val-acc`', '`val-auc`'.

Solution.

(f) **(Code) Implement function `eval_kfold(x, y, k, logistic)` which takes in number of folds k, (1) uses 3c to split the data, and (2) evaluates the performance using the input classifier logistic instantiated as logistic regression model. You can assume the logistic regression classifier will be created using `sklearn.linear_model.LogisticRegression`. Your function should return a dictionary containing the accuracy and AUC for the training and validation sets using the following keys: '`train-acc`', '`train-auc`', '`val-acc`', '`val-auc`'. The accuracy and AUC for this part should be a averaged across the k folds.**

Solution.

(g) **(Code) Implement the Python function `eval_mccv(x, y, valsize, s, logistic)` that takes in the validation size and the sample size s and uses the Monte Carlo cross-validation approach with s rounds (i.e., use the validation/hold-out technique from 3d, s times). The output should be the same format as 3e.**

Solution.

(h) **Written) Fit Ridge and LASSO using the K-fold cross validation approach with $k = 5, 10$, and varying alpha (regularization weight). Report the best setting (combination of k and regularization weight alpha)**

Solution.

(i) **(Written) Fit Ridge and LASSO using the Monte Carlo Cross-validation (MCCV) approach with $s = 5, 10$ rounds and different valsize ratios, and varying alpha (regularization weight). Report the best setting (combination of valsize, s, alpha).**

Solution.

(j) **(Written) Using the best parameters identified in 3g and 3h, re-train the regularized logistic regression models (Ridge, Lasso) using all the training data and report the performance on the test set in terms of AUC and accuracy in a table. You should have 4 models to compare: Ridge and Lasso trained using best parameters from k-fold and Ridge and Lasso using best parameters from MCCV. Compare how the model selection techniques compare to one another w.r.t. AUC and accuracy, predicted vs. actual test error, and computational complexity (running time).**

Acknowledgements

I would like to acknowledge that I attended both TA Swati's and TA Ziyang's office hours over the past week.