Assignments

PA - 0

Attached Files: code.py (1.991 KB)

The purpose of this assignment is to get you started with Python and the required libraries. Please download the code.py file.

Your task is to

1. Modify the code.py file (according to the instructions in the file itself),

2. Run the code (after you modify it, of course); the code will generate a result.txt file

3. Submit both the new code.py file and the result.txt file. Do not zip the files; attach both files. Do not change the names of the files.

PA - 1

Attached Files: code.py (2.726 KB)
datasets_x.pkl (23.055 MB)
datasets_y.pkl (525.952 KB)
load_datasets.py (2.849 KB)

Programming assignment 1. Topic: decision trees.

Please download the attached files. Follow the instructions in the code.py file. Your tasks

1. Modify the code.py as described in the instructions therein.

2. Run the code.py file. It will generate a results.pkl file.

3. Attach the modified code.py file and the results.pkl file it generates.

4. When run, the code.py file should print the accuracies on the train and test sets. What kinds of trends do you observe? Anything expected? Anything surprising? Discuss your findings IN THE COMMENT SECTION BELOW. One insightful paragraph (about 200 words or so) should be sufficient.

Note: When you download the files, their names might be altered, either because blackboard decided to change the file names internally, or because you have the same-name files in your downloads folder. For example, code.py might be downloaded as code.py(1), and so on. Make sure to rename them to what they are supposed to be.

PA - 2

Attached Files: binarized_xs.pkl (3.257 MB)
binarized_ys.pkl (70.973 KB)

Topic: naive Bayes and parameter estimation.

We learned in class about naive Bayes, MLE estimates, and Bayesian estimates. One issue we explored is the role of the alpha parameter in scikit-learn code: it is used as the Beta (or Dirichlet) prior, and the assumption is that the prior is always assumed to be uniform. The larger the alpha value, the stronger the prior is, and the smaller the alpha value, the weaker the prior is. When alpha = 0, it is equivalent to MLE estimates. Note that scikit-learn does not keep track of the posterior $p(\theta \mid D)$

It simply computes $P(x_{next} \mid D)$

In this assignment, you will explore how well your model fits to the training data and how it performs on the test data for various values of alpha. We expect that when alpha is small, it should fit better to the training data (the weaker the assumption, the more the data speaks), but how do you expect it to perform on the test data for small and large values of alpha?

Of course, for performance, we need to choose a metric. In this case, we will measure directly what the naive Bayes model optimizes for. That is, the joint log likelihood. Mathematically, it is $\sum_{d \in D} \ln(P(y_d, \overrightarrow{x_d}))$

BE CAREFUL: it is NOT $\sum_{d \in D} \ln(P(\overrightarrow{x_d}))$ and it is NOT $\sum_{d \in D} \ln(P(y_d \mid \overrightarrow{x_d}))$. Please see OneNote page titled PA-2 for more explanation.

You will be using Bernoulli Naive Bayes: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html.

It requires binarized data. Your TAs already binarized the features and classes for you; simply download the attached binarized_xs.pkl and binarized_ys.pkl files. These are pickle files that you need to load. These files contain 10 datasets (their inputs and outputs).

Your tasks:

1. Download the attached datasets.
2. Load the datasets.
3. For each dataset:
    1. Split the dataset into train-test split, using train_test_split method from Scikit-learn, with test_size set as 1./3 (NOT 0.3, NOT 0.33), and the random state should be set to the last four digits of your A#; drop the leading zeros. For example, if your A number is A12345678, your random state should be 5678; if your A number is 12340078, your random state should be 78. Note that the random state should be set to an integer; not a string, or other formats.
    2. For 15 different alpha values from $10^{-7}, 10^{-6}, ..., 10^{0}, ..., 10^{6}, 10^{7}$
        1. Create a new Bernoulli Naive Bayes model with that alpha
        2. Fit the model to the training set
        3. Compute joint log likelihood for the training set; store it somewhere
        4. Compute joint log likelihood for the testing set; store it somewhere
4. Store your results in a train_jll 2D array, and in test_jll 2D array, where the rows correspond to datasets, and columns correspond to different alpha values, and each entry corresponds to the joint log likelihood. That means, these arrays' shapes should be (10x15).
5. Create a tuple (train_jll, test_jll) and pickle it as result.pkl.
6. Discuss your results in the comments section of the submission. A paragraph (200 words or so) should be sufficient.
7. Submit your code, named as code.py file and your result.pkl file.

Notes:

1. No template code is provided in this assignment. You can use one of the templates provided in earlier assignments. Make sure you use the datasets from this assignment.
2. For log, use ln.
3. I recommend you to read the naive Bayes code from scikit-learn; it is open-source and it is a great resource. It might even have a few functions that you find useful for this assignment. You do not need to implement what is already implemented in scikit-learn; you can use any function, including internal ones. However, make sure it does what the assignment asks for.
4. If you have questions, ask on Piazza. However, do NOT share code or solutions on Piazza.
5. Optional: I recommend you to plot, for each dataset, the joint log likelihood for train and test as a function of alpha; do NOT submit the plots, but they should help you understand your results better. You can use matplotlib for plotting them.
6. Optional: Feel free to experiment with alpha = 0 on this dataset and other datasets; do NOT submit alpha = 0 results.

PA - 3

Topic: Logistic regression

We learned in class that regularized logistic regression balances between the model fit and model complexity. For model fit, we maximized conditional log-likelihood (CLL). For model complexity, we used either the sum of the squares of the weights (i.e, L2 penalty) or the sum of the absolute values of the weights (i.e, L1 panelty). We also learned that L1-regularization can result in weights that are exactly zero, in essence performing feature selection, whereas L2-regularization will result in small but not zero weights.

In this assignment, we will test the following:

1. Is and how model complexity related to overfitting to the data?

2. Feature selection capabilities of L1 versus L2 regularization.

Download the datasets we provided in PA-2 (the two files). We will use scikit-learn logistic regression: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html The C parameter of the constructor controls the complexity of the model.

To test overfitting vs model complexity:

Split the data into train and test split first. Then, for a range of C values, create an l2-regularized logistic regression classifier, fit it to the training data, compute the CLL on training set and test set, compute the model complexity, and plot training and test set CLLs as a function of model complexity (i.e., CLL in the y axis, model complexity in the x axis). Record the results in an array.

To test feature selection:

Split the data into train and test split first. Then, for a range of C values, create a logistic regression classifier, fit it to the training data, and then compute the number of weights that are exactly zero, and plot it as a function of C.

Mathematical details:

CLL is equal to $\sum_{d \in D} \ln(P(y_d \mid \overrightarrow{x_d}))$

.

Model complexity of L2-regularized logistic regression: $w_0^2 + \sum_{i=1}^{k} w_i^2$

Model complexity of L1-regularized logistic regression: $|w_0| + \sum_{i=1}^{k} |w_i|$

For w0, use the intercept_ attribute of the classifier. If it is an array, it should have a single scalar in it; use the scalar value. For other weights, use the coef_ attribute of the classifeir. It is an array of arrays, but the outer array should have a single array in it; use that array.

Task details:

1. Download the attached datasets from PA-2.
2. Load the datasets.
3. For each dataset:
    1. Split the dataset into train-test split, using train_test_split method from Scikit-learn, with test_size set as 1./3 (NOT 0.3, NOT 0.33), and the random state should be set to the last four digits of your A#; drop the leading zeros. For example, if your A number is A12345678, your random state should be 5678; if your A number is 12340078, your random state should be 78. Note that the random state should be set to an integer; not a string, or other formats.
    2. For 15 different C values from $10^{-7}, 10^{-6}, ..., 10^{0}, ..., 10^{6}, 10^{7}$
        1. Create a new L2-regularized Logistic regression classifier, with penalty="l2", C is set to the appropriate value, random_state=42. Do NOT set the other arguments of the constructor.
        2. Fit the model to the training set
        3. Compute the model complexity; store it somewhere
        4. Compute the number of zero weights; store it somewhere
        5. Compute CLL for the training set; store it somewhere
        6. Compute CLL for the testing set; store it somewhere
        7. Create a new L1-regularized Logistic regression classifier, with penalty="l1", C is set to the appropriate value, random_state=42. Do NOT set the other arguments of the constructor.
        8. Fit the model to the training set
        9. Compute the number of zero weights; store it somewhere
4. Store the model complexity and CLL results in l2_model_complexity 2D array, in l2_train_cll 2D array, and in l2_test_cll 2D array, where the rows correspond to datasets, and columns correspond to different C values. That is, these arrays' shapes should be (10x15). Note that these are computed only for L2-regularized model; not the L1. Plot 10 graphs, where the graph title is the dataset number, the y-axis has the train_cll and test_cll, and the x-axis is the model_complexity. Pick your favorite one and save it as "favorite_complexity_vs_overfit.png" file.
5. Store the number of zero weights in l2_num_zero_weights 2D array and l1_num_zero_weights 2D array, where the rows correspond to datasets, and columns correspond to different C values. That is, these arrays' shapes should be (10x15). Plot 10 graphs, where the graph title is the dataset number, the y-axis has the l2_num_zero and l1_num_zero, and the x-axis is the exponent of the C. Pick your favorite one and save it as "favorite_feature_selection.png" file.
6. Create a tuple (l2_model_complexity, l2_train_cll, l2_test_cll, l2_num_zero_weights, l1_num_zero_weights) and pickle it as result.pkl.
7. Discuss your results and figures in the comments section of the submission. A paragraph (200 words or so) should be sufficient.
8. Submit your code, named as code.py file, your two favorite figures, and your result.pkl file.

Notes:

1. No template code is provided in this assignment. You can use one of the templates provided in earlier assignments or from your own solution to PA-2. Make sure you use the datasets from assignment PA-2.
2. For log, use ln.
3. If you have questions, ask on Piazza. However, do NOT share code or solutions on Piazza.