Case Study 1

Email Spam Classification

Assigned: Wednesday, October 9, 2019

Pre-submission deadline: Friday, October 25, 2019 at 11:59pm

Due: Friday, November 1, 2019 at 11:59pm

A typical email spam filtering solution these days involves multiple layers of filters, including blacklisting IP addresses, global content-based filtering, and personalized content-based filtering. In this case study, you will construct a personalized content-based email spam filter using supervised learning techniques.

DATA DESCRIPTION

You are given a set of training emails in CSV format. The emails have been pre-processed into a set of features that are useful for classifying emails to this recipient as spam or not spam (ham). Each feature corresponds to a word and denotes the proportion of all words in the email that match the given word.

Each row consists of 1 email, with the first 30 entries corresponding to the features, all in the range $[0, 1]$. The last entry in each row is the target class, which is either 0 (not spam) or 1 (spam). The training data initially provided to you consists of 1500 emails.

At the pre-submission deadline for the case study, you will have the opportunity to submit your classifier code for evaluation on a second set of 1500 emails. The features and labels for these 1500 emails will be made available to you after this evaluation, resulting in 3000 total labeled emails you can use to train your classifier. See the Pre-submission (Optional) section for additional details on this pre-submission.

OBJECTIVE

Your task is to train a classifier to predict whether an email is a spam email given the features. Your classifier will be evaluated on a held-out test data set. It is recommended to use well-written and tested implementations of machine learning algorithms for this case study. While the code that we have been using in class from the textbook website is useful for learning and illustration purposes, it is not as fully featured or robust as a well-established machine learning package such as `scikit-learn`. To help you get started with `scikit-learn`, see the Python script `classifySpam.py` provided with the case study on Blackboard.

Building an accurate machine learning system is an iterative refinement process that involves

- Understanding the properties of the data by conducting exploratory data analysis and visualization.
- Understanding the underlying assumptions behind different machine learning algorithms to identify whether a particular algorithm is well-suited to an application.
- Pre-processing the data in a manner so that is most appropriate to match the assumptions of the chosen algorithm.
- Examining intermediate variables, decision boundaries, and outputs of a machine learning algorithm beyond just the classification accuracy to refine the trained model.
- Experimenting with a large number of models and algorithms.

While it is certainly possible to build an extremely accurate classifier just based on extensive experimentation, it would probably be a better use of your time to be selective in how you choose algorithms and to carefully examine the outputs of an algorithm to improve it.

GRADING: 75 POINTS TOTAL

Students should form groups of 2 or 3 students, all at the same level (BS, MS, or PhD). BS students at the Lorain campus may form groups with BS students at the UToledo Main Campus.

Classification accuracy: 30 points

Your classifiers will be evaluated for accuracy using two metrics on a held-out test set (not available to you):

- AUC: Area under the Receiver Operating Characteristic (ROC) curve.
- True positive rate at 1% false positive rate.

This will require you to use your judgment to select a model that provides a good balance of accuracy and model complexity (to avoid overfitting). Accuracy on each metric is worth 15 points.

We will only train your classifier once, so if your classifier has any random component, e.g. for random initialization of weights for the multi-layer Perceptron, you may want to *set the seed* for the random number generator in your classifier to ensure that you are submitting your best algorithm. Within `scikit-learn`, you can do this by setting the `random_state` parameter to a fixed integer seed value.

Your grade for accuracy on each metric will be determined by the accuracy of your classifier *compared to the rest of the class*, i.e. the most accurate algorithm receives 15 points, the next most accurate algorithm receives 14 points, etc. If two algorithms are extremely close in accuracy, they may be assigned the same grade. Also, if there is a large gap in accuracy between two groups' algorithms, there may be also be a gap in the grades (e.g. no group assigned a grade of 13 points because there is a large gap between the model that received 14 points and the model that received 12 points).

Your algorithm will also be compared to a simple baseline that is not expected to be a competitive solution. In this case study, the baseline will be the Gaussian naïve Bayes model, fit directly to the data with no pre-processing. The accuracy of the baseline model will be considered as 0 points, so the accuracy of your algorithm ***must exceed that of the baseline*** in order to receive a grade for accuracy! See the `classifySpam.py` file posted on Blackboard with the assignment for code implementing the baseline.

Note: If your code generates an error and does not run correctly, then your grade for accuracy will be a 0. This may occur due to your code not following the given specification (see Source Code Specifications section) or due to faulty assumptions that result in a non-functional classifier. ***We will not make any attempts to modify your code to correct it!*** You will have the opportunity to pre-submit your code for evaluation about one week prior to the deadline to try to catch such errors (see Pre-submission (Optional) section).

Report: 45 points

Submit a report of not more than 5 pages that includes descriptions of

- Your final choice of classification algorithm, including the values of important parameters required by the algorithm. Communicate your approach in enough detail for someone else to be able to implement and deploy your spam filtering system. (5 points)
- Any pre-processing, such as exploratory data analysis, normalization, feature selection, etc. that you performed, and how it impacted your results. (10 points)
- How you selected and tested your algorithm and what other algorithms you compared against. *Explain why you chose an algorithm and justify your decision!* It is certainly OK to blindly test many algorithms, but you will likely find it a better use of your time to be selective based on the specifics of this data set and application. (10 points)

- Recommendations on how to evaluate the effectiveness of your algorithm if it were to be deployed as a personalized spam filter for a user. What might be a good choice of metric, and what are the implications on the classifier? How might you solicit feedback from users to evaluate and improve your spam filter? (10 points)

Your report will be evaluated on both technical aspects (35 points, distributed as shown above) and presentation quality (10 points). Think of your submitted model as a representation of your "final answer" for a problem, while your report contains your "rough work." If you tried several approaches that failed before arriving at a model that you are satisfied with, describe the failed approaches in your report (including results) and what you learned from the failed approaches to guide you to your final model.

You may include additional descriptions, figures, tables, etc. in an appendix beyond the 5-page limit, but content beyond page 5 may not necessarily be considered when grading the report.

SUBMISSION DETAILS

Although you are expected to work in groups of 2 or 3 students, each student is required to submit their case study individually on Blackboard. The contents of the submissions for all group members should be the same.

For this case study, submit the following to Blackboard:

- A file named `classifySpam.py` containing your source code.
- A Word document or PDF file containing your report.

Do not place contents into a ZIP file or other type of archive! Blackboard automatically renames your files with your username to facilitate grading but does not open archives to rename those files.

Source Code Specifications

Your source code must contain the following function with *no modifications to the function header*:

```
def predictTest(trainFeatures,trainLabels,testFeatures):
```

This function trains a classifier using the training features and labels to predict the labels of the test features. Since you do not have access to the held-out test set, you can test your function by splitting the training data set provided to you into training and test sets, by cross-validation, etc.

If you have any other code in your `classifySpam.py` file, please place it under the following `if` block:

```
if __name__ == "__main__":
```

This prevents the code from running when we import the `predictTest()` function from your `classifySpam.py` file. For an example, see the structure in the `classifySpam.py` file on Blackboard with the assignment. We will use the script `evaluateClassifier.py` (also posted on Blackboard) to evaluate the accuracy of your classifier.

Important note: Since this case study is designed to simulate a deployment of an actual machine learning system, it is very important to ensure that your model can be applied in practice, where you have no access to the test data. *Thus, you are not allowed to use elements of the test data to train your model*. ***Be careful that you are not accidentally violating this rule!*** For example, the simple normalization approach we have used in class of first normalizing the features then splitting into training and test is a violation because we have accessed elements of the test data in order to compute our normalizations.

Pre-submission (Optional)

Groups may optionally participate in a pre-submission that mimics the actual grading process for classification accuracy. *This is purely for informational purposes and will not count towards your final grade!* If your group elects to participate, you should submit their code *on Blackboard the same way you would submit your actual submission*, being sure to ***follow the source code specifications***!

We will run your classifier on a second (held out) data set containing 1500 emails and report the accuracy metrics for each group on a leaderboard visible to all groups. Groups may use this leaderboard to gauge their standing within the class. *The leaderboard will be anonymous* (no group or student names), but each group who submitted their code will be informed of the accuracy of their submission. The 1500 emails used for evaluating this pre-submission will be made available to all groups following the evaluation, regardless of whether they participated in the pre-submission.

Note: the held-out data set used for evaluating the pre-submission is ***different*** from the held-out data used for the final accuracy evaluation. However, the final accuracy evaluation will be conducted using the same process as for the pre-submission, so groups may elect to use this as a "test run." In particular, it allows groups to test if their code has any issues that prevent it from working properly prior to the final evaluation.