Practical 2: Spam Classification

Emily Chen, Joanna Chung, Kevin Lee
Emails: emily-chen@, joannachung@, kevinlee01@college.harvard.edu
Camelot.ai usernames (Team Machine_learners): emchen, jcat1, kevin

March 11, 2018

1 Technical Approach

To construct and test our models, we first split the given "training" data set in two, yielding a training and a validation set. This new training set comprised roughly 85% of the original data set, and the remaining 15% became the validation set, on which we evaluated the different models.

First Iteration

On the first iteration of our approach, we made use of Python's Scikit-learn package and tested various types of classification models and model classes with the features already given to us:

- k-Nearest Neighbors (Model 1): We iterated through all possible combinations of the following hyperparameter values to maximize the accuracy score on our validation set:
  - k values: 1-20, 25, 50, 100, 150
  - Weight function: neighbors weighted equally or neighbors weighted by the inverse of their distance from the point
  - Distance measure: Manhattan distance or Euclidean distance

  The combination that yielded the highest accuracy score was $k = 17$, neighbors weighted by distance, and the Manhattan distance measure.
- Random Forest (Model 2): We iterated through the following hyperparameter values:
  - Number of trees: 5, 10, 25, 50, 100, 250, 500, 750, 1000
  - Maximum number of features (to consider when looking for the best split, out of the total n features): $\sqrt{n}$, $\log_2 n$, n
  - Maximum tree depth: 2, 4, 10, 25, 50, or nodes are expanded until all leaves are pure
  - Class weight: class weights inversely proportional to class frequencies in the training data or weights constant across all classes

  The combination that yielded the highest accuracy score was 1000 trees, maximum tree depth of 5, maximum number of features $\sqrt{n}$, and constant class weights.

- Logistic Regression (Model 3): We iterated through the following hyperparameter values:
    - Intercept: bias term included for the weights or no bias term
    - Number of cross-validation folds: 3-10

  Here, we perform cross-validation while constructing our model, and select 4-fold cross-validation as the optimal process. We also include a bias term in our weights and employ L_2 regularization to encourage weights of smaller magnitude as a means of preventing overfitting and making the model more generalizable to new data.
- Naive Bayes/Bayesian Naive Bayes (Model 4/5): toggled smoothing and prior
    - Smoothing parameter
    - Uniform prior or given frequencies (then it becomes a Dirichlet-Multinomial model, can be solved in closed form → training and predicting is very fast)
- Support Vector Machine (Model 6): We iterated through the $C = 0.25, 0.5, 1, 2, 5$ to maximize the accuracy score on our validation set and found the optiaml value was $C = 2$.
- Neural Network (Model 7): We iterated through the following hyperparameter values:
    - Loss activation function: identity, logistic, tanh, relu
    - L_2 penalty coefficient: 0.0001, 0.0001, 0.001, 0.01, 0.1, 1

  The combination that yielded the highest accuracy score was the logistic activation function with and penalty coefficient of $\alpha = 0.0001$.

The accuracy scores on the validation set for each of the aforementioned models are denoted in Table 1 below:

Model	Accuracy
Baseline 1: Most Frequent Class	0.39000
Baseline 2: Bigram	0.78474
Model 1: K-NN ($k = 17$)	0.87555
Model 2: Random Forest (1000 trees, max depth 50)	0.89301
Model 3: Logistic Regression	0.84934
Model 4: Naive Bayes	0.66968
Model 5: Bayesian Naive Bayes	0.66062
Model 6: SVM	0.76638
Model 7: Neural Network	0.87773

Table 1: First iteration of models and their validation set accuracy scores

After tuning the hyperparameters of our model classes during our first iteration, we found that Random Forest (with 1000 trees of max depth 50) was the best model, yielding a validation accuracy score of 0.89301 (as indicated by Table 1). Thus, we then turned to methods of feature engineering to further improve our models.

Second Iteration

On the second iteration, we turned to feature engineering to extract additional amounts of data regarding each XML file. We extracted a total of 10 additional features, which are listed in Table 2 below.

Feature	Value Set
Unique call tags	Count per call
Call-pairs	Count per pair
NETAPI32.dll.NetUserGetInfo	Recurrences
urlmon.dll	Recurrences
urlmon.dll.URLDownloadToFile	Recurrences
HKEY_LOCAL_MACHINE_SOFTWARE_Microsoft	Recurrences
HKEY_LOCAL_MACHINE_Keyboard	Recurrences
HKEY_LOCAL_MACHINE_SOFTWARE_Classes	Recurrences
get_host_by_name	Recurrences
MSVBVM60.DLL	Recurrences

Table 2: Additional features extracted from the given XML files

The first feature function we wrote is `counts_per_call`, which for each created a dictionary of all the unique calls per file and counted the number of recurrences of that call in the file. The intuition behind this was that similar viruses may have similar frequencies of similar calls.

The second feature function is `counts_per_twocall_window`,where we sought to take into account the sequence of calls, with the idea that classes of viruses would have the same strategy/procedure for carrying out their calls, and thus similar orders of calls. The feature function pairs adjacent calls, and for each unique pair, counts the frequency the pair occurs in the file.

The third feature function, `virus_specific_words`, extracted the features listed in rows through 10 of Table 2. We researched into the technical fingerprints of each virus, and added any virus-unique identifiers as a feature to be counted. For example, we found from the Infosec Institute (see citations in comments in code), that the VB virus downloads a file called MSVBVM60.DLL into the systems32, and no other type of virus or normal program does so. Thus, we parsed through the attributes for each call of with a `load_dll` class and checked whether this string was present in as the filename attribute. The other features were determined similarly, according to the nature and behavior of the viruses.

After extracting these features, we conducted the same process for selecting hyperparameter values as we did in the first iteration of models on this expanded data set. The accuracy scores on the validation set for each of the models are denoted in Table 3 below:

2 Results

The final model we used was Random Forest (with 50 trees of max depth 50) trained on the given data in addition to all the features we extracted for each XML file. This model yielded the highest

Model	Accuracy
Model 1: K-NN ($k = 11$)	0.87118
Model 2: Random Forest (50 trees, max depth 50)	0.90175
Model 3: Logistic Regression	0.86026
Model 4: Naive Bayes	0.78367
Model 5: Bayesian Naive Bayes	0.80181
Model 6: SVM	0.81659
Model 7: Neural Network	0.88646

Table 3: Second iteration of models (with feature engineering) and validation set accuracy scores

validation accuracy score of 0.90175 running on roughly 85% of the given training set.

After training this model and using it to make predictions on the given test set, we uploaded our predictions to Camelot. We obtained a test set accuracy score of 0.82105, beating both the baseline Most Frequent Class Model (0.39) and Bigrams Model (0.78474) by a significant margin. This result from our final iteration was the best score we achieved, although the first iteration of models beat the baselines already.

3 Discussion

In summary, our search for the best model included multiple different classes of models, both non-parametric and parametric. We found that the best model was the Random Forest (with 1000 trees of max depth 50) model, tested on the validation set (which we had separated from the original given training set). This gave us a validation accuracy score of 0.89301 and Camelot test accuracy score of 0.81368, bringing us over both the Most Frequent Class and Bigrams baselines. We then turned to feature engineering in hopes of further improving our score.

Next, we extracted 10 additional features per XML files (see Table 2 above and additional functions added to classification_starter.py), and ran our models on this extended dataset. Here, the best model was also Random Forest (with 50 trees of max depth 50).This improved our validation accuracy score to 0.90175 and Camelot test accuracy score to our final 0.82105.

One thing we would also look into given more time is the observation that our validation set accuracy scores were higher than the scores we obtained on the Camelot test set (across all models). We do not believe this is an issue of overfitting, as we constructed the validation set to circumvent this. Perhaps there are differences in our validation set due to the random split and the test set, though this would require further investigation.

For future experimentation, our implementation of the Neural Network and k-NN models showed promising results. We would likely want to further explore the Neural Network over the k-NN due to its greater computational efficiency as a parametric model. With further fine-tuning of the parameters and modification of the model, it may become the model with the best results. The accuracy in our second iteration was already 0.88646, very close compared to our random forest of 0.90175.