Sentiment Analysis on Product Reviews

By:

Sahiti Desu

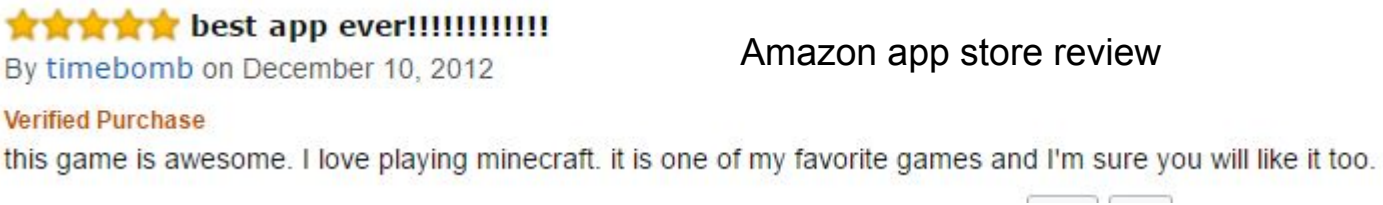
Ankit Kumar

Goal

We will build a sentiment classifier for any app store product reviews.

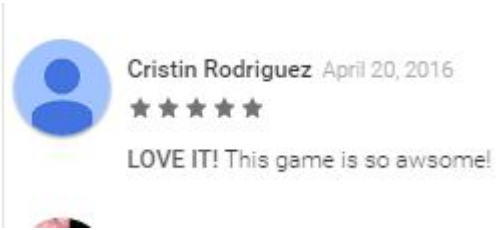
At the end of the project we will be able to process an app review and determine with good accuracy, the sentiment behind the review.

App reviews



Amazon app store review

Google app store review



In effect, the format is the same for different types of reviews for app stores across the board. [Rating, Title, Description]

Steps:

- Gather data
  - Amazon doesn’t allow you to get the reviews directly.
  - Created a web crawler. Provide product id to the crawler and it will fetch all the reviews for that particular product.
  - Defined the class label on the basis of the ratings provided (>=3 is 1)
  - Challenges faced:
    Amazon has lots of bot checking code in place, such as,
    - Honey pots
    - Robots.txt files
    - User-agent checking

    to prevent people from gathering data from their product pages

  - Circumvented all these problems to get over 5000 reviews of a given application on the amzon android app store.

Steps:

- Pre-processing data

  To extract features, we decided to get the review and the corresponding sentiment and applied various pre-processing steps in the following order:

  - Tokenized the sentences using Tweet tokenizer - due to similarity of content-type.
  - Removed punctuation marks from the reviews
  - Removed stopwords from the tokenized words.
  - Performed spell check on the filtered words by performing Edit Distance(with distance 2).
  - Did Lemmatization for the words after spell check, we used the WordNetLemmatizer.
  - Extracted the 1000 most frequent words as our features-set- based on term-frequency(see note)

Note: Performed features extraction using tf-idf by calculating those scores separately for positive reviews and negative reviews. Poor results.

What we didn’t do.. And Why!

- Used tf-idf for positive and negative reviews separately and extracted top 1000 features which has high tf idf score. After that we have created a training set with the tf idf scores as weights,
    - The tf-idf method for extracting features, fails as we lose those features that have been repeated a high number of times but are very relevant and crucial to determining product reviews

  Did an analysis on the data set that we gathered. Found that the word “great” occured more than the word “of” 781 vs 755

- Used stemming before feature extraction, but the results obtained were poor and so we used lemmatization and discarded this approach.

Steps:

- Performing the Spell Check:
  - We read a big text file, big.txt, which consists of about a million words.
    The file is a concatenation of several public domain books from Project Gutenberg and lists of most frequent words from Wiktionary and the British National Corpus.

  - We then extracted the individual words from the file (by converting every word in to lowercase, and then defined a word as a sequence of alphabetic characters, so "don't" will be seen as the two words "don" and "t").

  - Next we trained a probability model (count of each words occurs).

  - **Smoothing**, we smooth over the parts of the probability distribution, that would have been zero(that has not occurred in the bigtext) by bumping them up to the smallest possible count.
    This is achieved through the class collections.defaultdict, which is like a regular Python dict enumerated the possible corrections c of a given word w, ie(addition, insertion, deletion and alternation)

  - This was done twice for calculating edit distance of 2.

Spell check(example):

Example

known_edits2('something') is a set of just 4 words: {'smoothing', 'seething', 'something', 'soothing'}

extracted the candidate words, by first checking whether the word is present in the know words and then the word with e

edit distance 1 is present and then the edit-distance with 2 is present.

If none of them are present we return the word itself.

The word in the candidate with the max probability of that candidate word repeating in the big set of words.

Steps:

- Created a training set using the familiar Bag-of-words method.

Evaluation of Classifier:

- Why do we need evaluation?
  - Instead of seeing how fast the classifier is classifying the instances, it is always to good to evaluate the predictive capability of the classifier.
  - Eg: Consider a 2-class problem,Number of Class 0 examples = 9990,Number of Class 1 examples = 10.IIf model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %. Accuracy is misleading because model does not detect any class 1 example
  - Performance of a model may depend on other factors besides the learning algorithm:–Class distribution–Size of training and test sets

```
J48 tree = new J48();
tree.setMinNumObj(2);
J48 tree2 = new J48();
tree2.setMinNumObj(5);
J48 tree3 = new J48();
tree3.setMinNumObj(10);
J48 tree4 = new J48();
tree4.setMinNumObj(30);
NaiveBayes nb= new NaiveBayes();

SMO svm1 = new SMO();
svm1.setC(1.0);
SMO svm2= new SMO();
svm2.setC(5.0);
SMO smv3=new SMO();

RBFKernel rbf=new RBFKernel();

smv3.setKernel(rbf);
smv3.setC(1.0);
SMO smv4=new SMO();
smv4.setKernel(rbf);
smv4.setC(5.0);
IBk ibk1=new IBk(1);
IBk ibk2=new IBk(3);
IBk ibk3=new IBk(5);
IBk ibk4=new IBk(10);
```

Evaluation of Classifier:

- How did we evaluate ?
  - Step 1:Have built different classifiers by adopting different machine learning algorithms like K-nearest neighbours, Decision Trees, SVM, Naivebayes by changing the parameters for each classifiers.
  - Step 2: Evaluated the performance of the classifiers using different generators like HoldOut, 10-fold cross validation , Resample Generator. Ran these generators for 10 runs and had done reshuffling of data on each run.

- What are the different evaluation metrics that have been used?
  - Calculated the Average REsubstitution error and generalization errors for each classifier and calculated their mean and sd of those errors.
  - Computed paired Student's-t test to relatively compare the performance of the classifiers.

Code run by the classifiers

```
for (Classifier alg : classifiers) {
    for (Generator g : tdGenerators) {
        double[] resubErrors = new double[numRuns];
        double[] genErrors = new double[numRuns];
        Evaluation _evalTest;
        for (int run = 0; run < numRuns; ++run) {
            g.initializeRun();
            _evalTest = g.getEvalutaion();
            double avgPartResubErr = 0.0;
            double avgPartGenErr = 0.0;
            for (int part = 0; part < g.getNumPartitions(); ++part) {
                Instances train;
                Instances test;
                if (g instanceof CrossValidationGenerator) {
                    train = ((CrossValidationGenerator) g).getNextTraingingSet(part);
                    test = ((CrossValidationGenerator) g).getNextTestingSet(part);
                }
                else {
                    train = g.getNextTrainingSet();
                    test = g.getNextTestingSet();
                }
                alg.buildClassifier(train);
                _evalTest.evaluateModel(alg, train);
                double genErr = 0.0;
                double resubErr = 0.0;
                double errcount_train = 0;
                double errcount_test = 0;
                for (int i = 0; i < train.numInstances(); i++) {
                    double pred = alg.classifyInstance(train.instance(i));
                    String actual = Double.toString(train.instance(i).classValue());
                    String predicted = Double.toString(pred);
                    if (!actual.equals(predicted)) {
                        errcount_train++;
                    }
                }
                resubErr = (errcount_train) / train.numInstances();
                genErr = _evalTest.errorRate();
                avgPartResubErr += resubErr;
                avgPartGenErr += genErr;
            }

            resubErrors[run] = avgPartResubErr / g.getNumPartitions();
            genErrors[run] = avgPartGenErr / g.getNumPartitions();
            g.reset();
        }
```

Student t-test

Hypothesis test where the null hypothesis is that two models are the same

Compute the *t*-statistic: $$t = \frac{avgErr(M_1) - avgErr(M_2)}{\sqrt{(var\,(M_1 - M_2)/k)}}$$

Depending on test sets variance is estimated by:

$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}[err(M_1)_i - err(M_2)_i - (avgErr(M_1) - avgErr(M_2))]^2$$

Or

$$var(M_1 - M_2) = \sqrt{\left(var\frac{(M_1)}{k_1} + var\frac{(M_2)}{k_2}\right)}$$

The top variance is used if the same test sets were used for both models, bottom is used when two different test sets (with their own k1 and k2 numbers of folds)

Result

We ran the classifiers and calculated the mean and standard deviation of 5 runs.

After running paired-student-t-test, we have preliminarily short-listed 2 of the algorithms that are returning good results,

1. Naive Bayes - 81%
2. Support Vector Machine(RBF Kernel) - 76%
3. Support Vector Machine(Linear Kernel) - 72%
4. Decision Tree(depth = 10) - 63%

Right now the best option is NaiveBayesClassifier. More testing is needed to confirm.

Fin.