



A Comparison of Machine Learning Algorithms for Binary Classification of Free-text Fields Using SAS, R, and Python

Michael S. Rimler

Allison Hainline

Jim Vasquez

Issue at hand

- Analyzing text data via machine learning algorithms, specifically to analyze the data in a binary classification of free-text data, leveraging the findings in Rimler and Pitlyk (2019)

Objective

- Assess, compare, and highlight pros/cons of various methodologies for developing a binary classifier on free-text data in...
 - ...Python
 - ...R
 - ...SAS

Our findings across the languages



Python



Relatively easy to implement in all stages

Code is similar across classifiers

Computational performance is faster

R



Good first step for R programmers

Many options for fitting functions

Allows flexibility for the experienced programmer, may be overly complicated for a beginner

Computational performance a bit slower than Python

SAS



Time consuming and not as easy to code

High consumption of both time and memory

Perhaps greater success with the capabilities of SAS Enterprise Miner

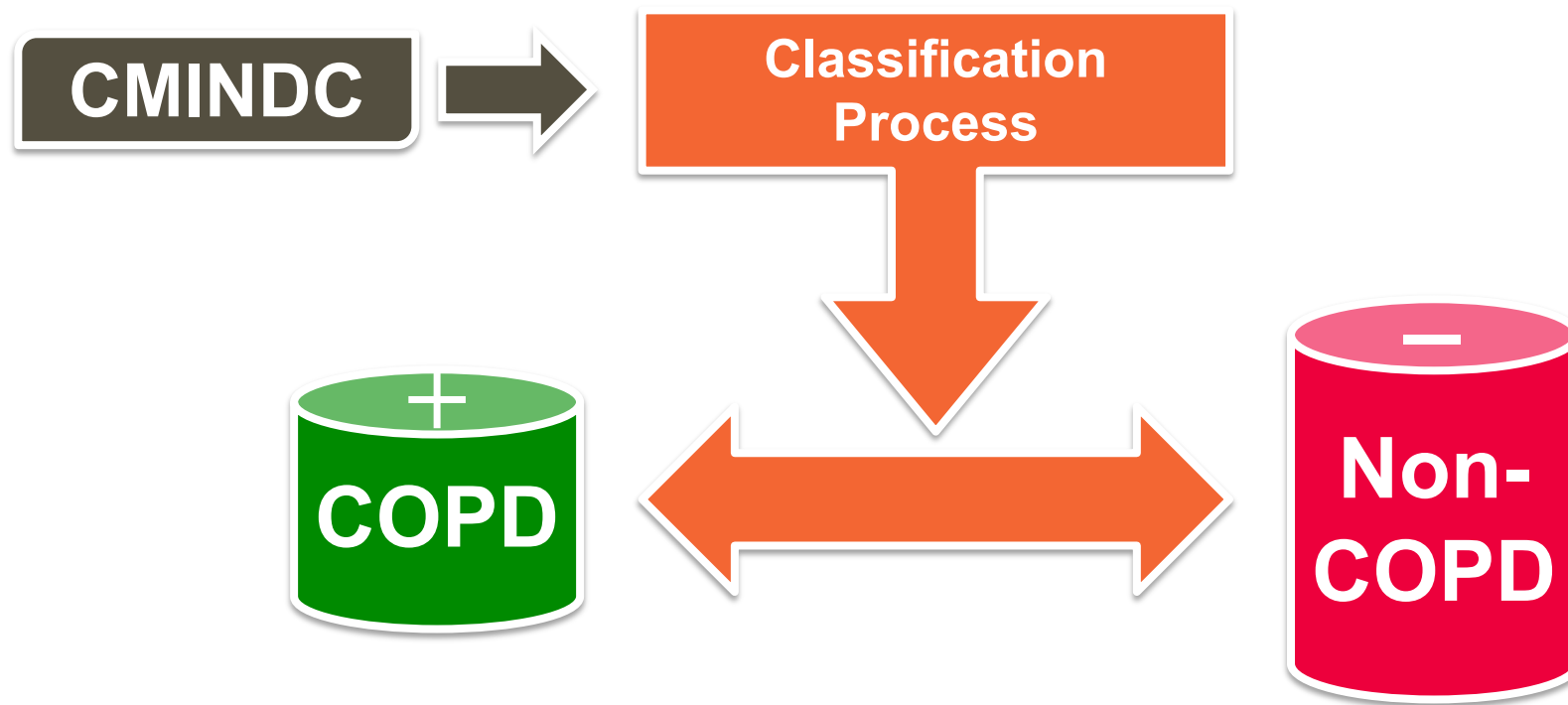


DATA BACKGROUND

The Problem Statement



Binary classification of free-text concomitant medication indication



Overall and Unique

106 Clinical Trials

- Therapeutic Area: Respiratory
- Indication of IP: COPD
- Routes of Administration: Nasal, Inhalation, Oral, IV, and others
- 18,481 unique values of CMINDC from 212,079 unredacted records
- 3.1% of unique values labelled COPD
- 20.5% of these do not contain *exact* string match for
 - COPD
 - CHRONIC OBSTRUCTIVE PULMONARY DISEASE
 - CHRONIC BRONCHITIS
 - EMPHYSEMA
- 2.3% contain 'COPD' and are not COPD (e.g., NON-COPD)

BINARY CLASSIFICATION OF FREE-TEXT DATA

Development of classifiers



Note: This process is not unique to our problem and is likely to be more complex when the objective is an **optimized classifier** ready for deployment.

STRATIFICATION INTO TRAINING AND TEST SETS

Stratification of overall data

- Python's **train_test_split** from Scikit-Learn **model_selection**
- Training → 80% (169,663)
 - 15,988 unique values of CMINDC
 - 495 (3.1%) are labelled COPD
- Test → Remaining 20% (60,501)
 - 6,143 unique values of CMINDC
 - 195 (3.2%) are labelled COPD
 - 69 of these are not in Training dataset (new positives)
 - Represents 12.2% of global list of unique CMINDC terms labelled as COPD

PREPPING THE DATA FOR TRAINING AND TEST SETS

- **Feature engineering:** creating characteristics of the data to leverage for predictive modeling
- **TF-IDF**
 - **Term Frequency:** the number of times that the term (word) appears in each record
 - **Inverse Document Frequency:** measures the incidence of records in the entire database which contain the word
- Further improvements for classification
 - **Edit Distance Measures:** metrics which characterize how different two text strings are from one another
 - **N-grams:** a method that looks at incidences of word -tuples, such as word pairs or word triples

MODEL SELECTION WITH TRAINING DATA



TUNE INTO YOUR DATA

- **Hyperparameter tuning:** tweaking the underlying model parameters on training data to improve model calibration
- **Cross-validation:** reduces the risk of **over-fitting** by facilitating out-of-sample testing with training data
- **LOGISTIC REGRESSION CLASSIFIERS**
 - Classifier based on logistic regression
- **RANDOM FOREST CLASSIFIERS**
 - Combines multiple decision tree methods to produce a score used to classify a record
 - Since singular decision tree methods are prone to overfitting, random forest methodology aims to correct for this
- **NAÏVE BAYES CLASSIFIERS**
 - Leverages Bayes' theorem to generate a probabilistic score using naïve independence assumptions between the features of the data

PREDICTING CLASSES WITH TEST DATA



- We calibrate the model for a given classifier then we can feed the test data into the fitted model
- The test data must be prepared similar to the training data, including any engineered features such as TF-IDF weightings
 - With a dictionary of words derived from the training data, it is possible that new vocabulary words appear in the test data and receive zero weight
- After calculating the predicted probability of COPD classification using the selected threshold, we assess the through statistics of the classification method such as accuracy, precision, and recall



PYTHON IMPLEMENTATION

- TF-IDF feature engineering:
 - Use CountVectorizer from Scikit-Learn's feature_extraction
 - Identify unique words in the (training) dataset
 - Generate a variable (vector) for each word containing the frequency of that word in the record
 - Apply TfidfTransformer from the same library to normalize the counts based on the TF-IDF algorithm

```
count_vect = CountVectorizer()

tfidf_transformer = TfidfTransformer()

X_train_counts = count_vect.fit_transform(x_train)

X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

- For the three classifiers of interest, the code within Python is almost identical
- The only difference is loading the proper module and defining the classifier (clf):

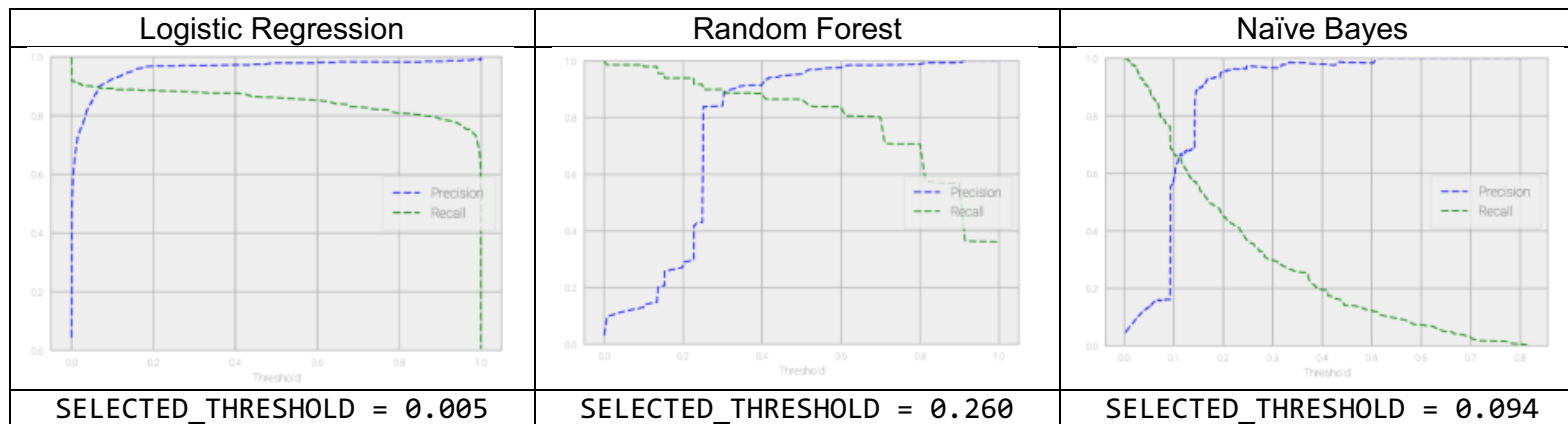
Logistic Regression	<pre>from sklearn.linear_model import LogisticRegressionCV from sklearn.model_selection import cross_validate clf = LogisticRegressionCV()</pre>
Random Forest	<pre>from sklearn.ensemble import RandomForestClassifier clf = RandomForestClassifier()</pre>
Naïve Bayes	<pre>from sklearn.naive_bayes import MultinomialNB clf = MultinomialNB()</pre>

Note: Since we are not optimizing the algorithms, we simply employ the default parameterization of each respective function.

DETERMINING A CLASSIFICATION THRESHOLD PROBABILITY



- Establish a threshold probability (score) for record classification
 - Increase recall at the cost of lower precision
 - i.e., reduce the number of false negatives at the expense of higher false positives



Python Results



Performance Comparison of Supervised Binary Classifiers on Test Data using Python

Classifier	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
100% Accuracy String	N/A	0.999	0.993	0.769	0	5
Logistic Regression	0.005	0.989	0.760	0.959	59	8
Random Forest	0.260	0.995	0.910	0.938	18	12
Naïve Bayes	0.094	0.990	0.895	0.790	18	41



R IMPLEMENTATION

- Several R packages are equipped with TF-IDF functionality. We selected the `text2vec` package.
 - **First step:** separate the phrases into *tokens* (individual words) which are used to create our vocabulary (similar to Python's `CountVectorizer`)
 - **Second step:** use TF-IDF weighting to transform our vocabulary into a document-term matrix that is normalized by the number of times each token appears in the dictionary
- Use the same TF-IDF model to transform test data

```
it_train = itoken(...)

vocab = create_vocabulary(it_train)

vectorizer = vocab_vectorizer(vocab)

dtm_train = create_dtm(it_train,
                       vectorizer)

tfidf = TfIdf$new()

X_train_tfidf = fit_transform(dtm_train,
                              tfidf)
```

```
X_test_tfidf = transform(dtm_test_tfidf,
                          tfidf)
```

Logistic regression

- Unlike Python, the three classifiers are implemented using different R packages
- **Logistic Regression:** from the `glmnet` package, use `cv.glmnet` function to perform 5-fold cross validation, returning the optimal lambda value

Logistic Regression

```
lambda_vals = 10^seq(3, -2, by = -.1) # Lambda values to search
cv_glm_fit = cv.glmnet(x = X_train_tfidf, y = y_train$COPD,
                      family = 'binomial', alpha = 0, lambda = lambda_vals, nfolds = 5)
opt_lambda = cv_glm_fit$lambda.min
glm_fit = cv_glm_fit$glmnet.fit
pred_class = predict(glm_fit, s = opt_lambda, newx = X_test_tfidf, type='class')
```

Random forest

- **Random Forest:** use `ranger` {`ranger`} and `train` {`caret`} functions to fit and cross-validate our model.

Random Forest

```
control <- trainControl(method="repeatedcv", number=5, repeats=3, search="random")
x = as.matrix(X_train_tfidf)
COPD_rf <- train(x=as.matrix(X_train_tfidf), y=y_train$COPD, method="ranger",
                 metric='Accuracy', tuneLength=15, trControl=control)
```

- We also used the `random` option in the `trainControl` function, which allows for random grid searching across all possible values of the `mtry` parameter
- Once the model has been trained, we predict classes using the test data with the `predict` {`caret`} function

Random Forest

```
pred_class <- predict(COPD_rf, newdata = as.matrix(X_test_tfidf))
```

Naïve Bayes

- **Naïve Bayes:** we use the `multinomial_naive_bayes` function from the `naivebayes` package

Naïve Bayes

```
mnb <- multinomial_naive_bayes(x = as.matrix(X_train_tfidf),  
                                y = y_train$COPD)
```

- Once the model has been trained, we predict classes using the test data with the default threshold probability 0.5

Naïve Bayes

```
pred_prob <- predict(mnb, newdata = as.matrix(X_test_tfidf),  
                     type = "class")
```

R Results



Performance Comparison of Supervised Binary Classifiers on Test Data using R

Classifier	Threshold	Accuracy	Precision	Recall	False Positives	False Negatives
100% Accuracy String	N/A	0.999	0.993	0.769	0	0
Logistic Regression	0.5	0.994	0.924	0.872	14	25
Random Forest	0.5	0.997	0.978	0.938	4	12
Naïve Bayes	0.5	0.971	0.525	0.954	168	9



SAS IMPLEMENTATION

- No package in SAS, derive TF-IDF from original code

TF

- Create the dictionary of words from the set of unique terms in the training data
- Count the incidence of each word in each record

```
do ii = 1 to %eval(&numwords); * Each word in record;  
  do jj = 1 to %eval(&vocabsize.); * Each term in vocab;  
    *** Count frequency of a word in a record *;  
    if words(ii) = terms(jj) then flags(jj) = flags(jj) + 1;  
  end;  
end;
```

IDF

- Count how many records in the training dataset contain term
- Use L2 to normalize within each record

```
do jj = 1 to %eval(&vocabsize.);  
  *** Count number of documents a word appears *;  
  if flags(jj) > 0 then idfs(jj) + 1;  
  else idfs(jj) = idfs(jj);  
end;  
if eof; * Keep last record (total num docs) *;
```

– Logistic Regression:

Logistic Regression

```
proc logistic data=DS8;  
  class tfidf1-tfidf%eval(&vocabsize.);  
  model copdyn(event='1')= tfidf1-tfidf%eval(&vocabsize.) / RIDGING = NONE;  
  score data=testdata out=predprobs;  
run;  
quit;
```

- Note: we have set the option RIDGING = NONE based on notes to the log.

FAIL!!!

- Enterprise Miner likely has a solution
- PROC HPFOREST
- PROC HPBNET

Notable Points

- The test dataset must be prepared the same as the training data
 - It has 9288 additional variables, each representing a word in the training data dictionary
 - Use same IDF values to calculate the TF-IDF
 - Therefore, completely new vocabulary words receive zero TF-IDF weight and yield no explanatory/fit power
- Due to memory limitations, we were unsuccessful in executing PROC LOGISTIC
 - The Hessian becomes quite large and our instance of SAS could not handle the memory requirements
 - If memory wasn't a limiting factor, we estimate the model would take at least 24h to converge (vs. a few minutes in Python)
- Also unable to produce results using PROC HPFOREST (for random forest classification) or for naïve Bayes
 - These classifiers seem only available via SAS Enterprise Miner or other publicly available bespoke code

CONCLUSION

Our findings across the languages



Python



Relatively easy to implement in all stages

Code is similar across classifiers

Computational performance is faster

R



Good first step for R programmers

Many options for fitting functions

Allows flexibility for the experienced programmer, may be overly complicated for a beginner

Computational performance a bit slower than Python

SAS



Time consuming and not as easy to code

High consumption of both time and memory

Perhaps greater success with the capabilities of SAS Enterprise Miner

-
- **Optimization of algorithms:** Real-world application would require optimization in the training phase and performing hyperparameter tuning to improve trained algorithm
 - **Ensemble models:** Consider ensemble methods, which may yield improved performance on the optimized classifiers
 - **Deployment:** Need to deploy the classifier to production on newly collected free-text data

Michael Rimler

GlaxoSmithKline

michael.s.rimler@gsk.com

Allison Hainline

GlaxoSmithKline

allison.e.hainline@gsk.com

Jim Vasquez

GlaxoSmithKline

james.x.vasquez@gsk.com