Northeastern University, Khoury College of Computer Science

CS 6220 Data Mining — Assignment 5

Due: 02/22/2024(100 points)

Dharun Suryaa Nagarajan
https://github.com/dharun4772/CS6220
nagarajan.dh@northeastern.edu

Naıve Bayes, Bayes Rules

The original performance of acoustic classification for Parkinson's Disease leverages speech recordings from controlled subject responses from variety of questions. The task in the competition was to detect whether or not a person X had Parkinson's disease from a sampling of data. As of 2018, the state of the art classifiers have achieved 90% correct classification on a held out dataset, both for subjects who had Parkinson's and those who did not (at equal rates). So, when classifier Y sees person X, it works correctly 90% of the time.:

1. Let's say that we run a clinic. This clinic leverages this classifier, which has 90% accuracy. Also, let us say that we know that our current patient load is that 10% of the population have Parkinsons and 90% of the population do not. Let's also say that we're seeing patient X, and the classification algorithm has detected that they have Parkinson's disease. What's the probability that indeed X has Parkinson's disease?:
   $P(A) : probability\ of\ having\ parkinsons.$
   $P(B) : probability\ of\ predicting\ parkinsons.$
   $P(A|B) : probability\ of\ having\ parkisons\ given\ predicting\ parkinsons.$
   $P(B|A) : probability\ of\ predicting\ parkisons\ given\ person\ having\ parkinsons.$
   $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$
   $P(B|A) = 0.9$
   $P(A) = 0.1$
   $P(B) = P(B|A) * P(A) + P(B|A^{-}) * P(A^{-})$
   $P(B) = 0.9 * 0.1 + 0.1 * 0.9$
   $P(B) = 0.18$
   P(A—B) = (0.9*0.1)/(0.18) = 0.5 = 50%

The Sum of Conditional Probabilities

In class, we reviewed three main rules in Bayesian probability inference: Conditional Probability, Bayes Theorem, Total Probability.
A well-known outcome of the three sets of rules is the fact that the sum of all the conditional probabilities equals one.

2. **Prove that:**

$$\sum P(A_i|B) = 1$$

Applying Conditional probability:

$$\frac{\sum P(A_i \cap B)}{P(B)} = 1$$
$$\frac{\sum P(B \cap A_i)}{P(B)} = 1$$

Applying Bayes theorem:

$$\frac{\sum P(B|A_i) * P(A_i)}{P(B)} = 1$$

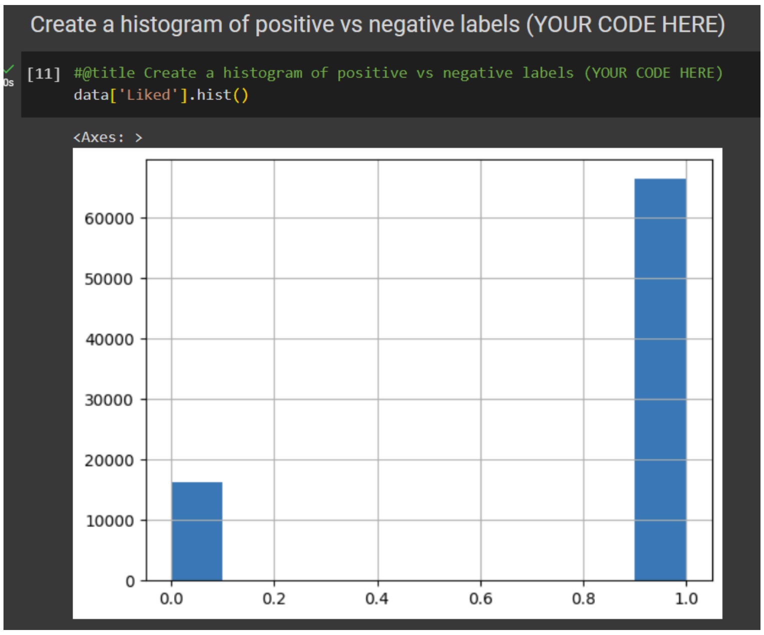
Total probability:

$$\frac{P(B)}{P(B)} = 1$$
$$1 = 1$$

Hence Proved.

Mining Reviews in Small-ish Datasets

3. With the Amazon magazines dataset, do the following. (Feel free to use the scikit-learn library):
Plot a histogram of the data with the number of positive and negative reviews.

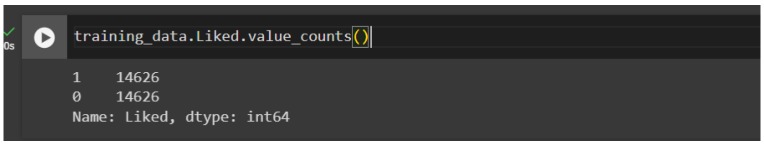

Create a histogram of positive vs negative labels (YOUR CODE HERE)

```
[11] #@title Create a histogram of positive vs negative labels (YOUR CODE HERE)
     data['Liked'].hist()
```

```
<Axes: >
```

Figure 0.1: Number of positive and negative reviews

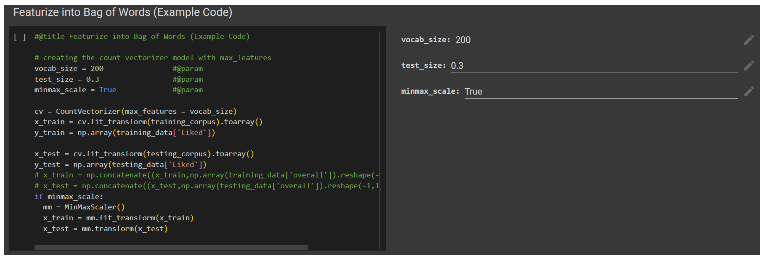
Balance the data so that you are training with equal probabilities, $P(Liked) = P(NotLiked) = 0.5$. We will use the original distribution for the evaluation dataset.



```
training_data.Liked.value_counts()

1    14626
0    14626
Name: Liked, dtype: int64
```

Figure 0.2: Balanced Training data

Use machine learning models to predict whether or not a review as written will result in a ”Liked” rating. Try changing different parameters, including the maximum vocabulary size. Also try normalization.
Answer: After trying normalisation and different values for the Max vocabulary size, the best results were showcased by Vocab size 200, test-size 0.3 and normalised data.



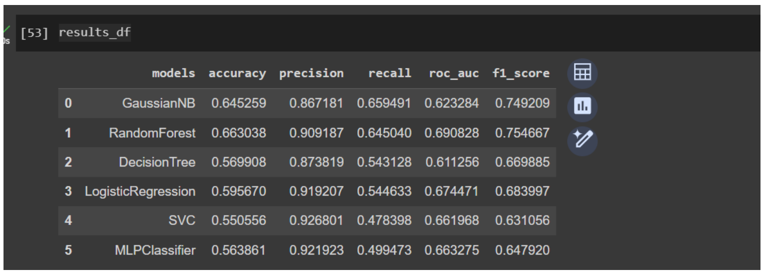
Featurize into Bag of Words (Example Code)

```
#@title Featurize into Bag of Words (Example Code)

# creating the count vectorizer model with max_features
vocab_size = 200                #@param
test_size = 0.3                 #@param
minmax_scale = True             #@param

cv = CountVectorizer(max_features = vocab_size)
x_train = cv.fit_transform(training_corpus).toarray()
y_train = np.array(training_data['Liked'])

x_test = cv.fit_transform(testing_corpus).toarray()
y_test = np.array(testing_data['Liked'])
# x_train = np.concatenate((x_train,np.array(training_data['overall']).reshape(-
# x_test = np.concatenate((x_test,np.array(testing_data['overall']).reshape(-1,1
if minmax_scale:
  mm = MinMaxScaler()
  x_train = mm.fit_transform(x_train)
  x_test = mm.transform(x_test)
```

vocab_size: 200
test_size: 0.3
minmax_scale: True

Figure 0.3: Parameters

Try out Na¨ıve Bayes, Decision Trees, Random Forests, and Logistic Regression machine learning algorithms and Print out your best accuracy, precision, and recall numbers for each algorithm.



```
[53] results_df
```

	models	accuracy	precision	recall	roc_auc	f1_score
0	GaussianNB	0.645259	0.867181	0.659491	0.623284	0.749209
1	RandomForest	0.663038	0.909187	0.645040	0.690828	0.754667
2	DecisionTree	0.569908	0.873819	0.543128	0.611256	0.669885
3	LogisticRegression	0.595670	0.919207	0.544633	0.674471	0.683997
4	SVC	0.550556	0.926801	0.478398	0.661968	0.631056
5	MLPClassifier	0.563861	0.921923	0.499473	0.663275	0.647920

Figure 0.4: Output

Answer: The result for the various different models after hyper-tuning them are shown above. Out of these **Random Forest** comes out as the best model with a Precision of 0.91, F1-score of 0.75, recall of 0.65, ROC-AUC as 0.69, accuracy as 0.66 . In terms of Precision MLP Classifier is the highest with a score 0.92

How do they compare? Which algorithms are over-fitting?
Answer:

a) Naive Bayes: Simple, fast. With sparse features, Naive Bayes can perform quite well as it’s well-suited for text classification tasks. But it does assume independece between features which is not evident in most of the scenarios.

b) Decision Trees: Over-fitting. Decision trees can become overly complex and may not generalize well to unseen data.

c) Random Forests: Built multiple decision trees and aggregated their predictions. With additional hyper parameter tuning, reduction in over-fitting compared to individual decision trees and performs the best across all metrics.

d) Logistic Regression: Simple and efficient, but assumed a linear relationship between features and the logarithm of the odds, which does not hold true. Have applied L1 technique to avoid over-fitting.

e) Support Vector Machines (SVM): SVM also suffers from over-fitting but it is effective in high-dimensional spaces, especially when the number of dimensions is greater than the number of samples.They are computationally expensive.

f) MLP Classifier (Multi-layer Perceptron): Can capture complex patterns in data due to its multilayer architecture. Effective for tasks involving non-linear relationships. Over-fitting happens here too, due to model complexity. They have the best precision compared to all algorithms.