Machine Learning Engineer Nanodegree

Capstone Report

Patrick O'Sullivan
June 20th, 2019

Business Public Sentiment

Baseline Model

Naïve Bayes

I used the Naïve Bayes model as my baseline model as it is a quick and easy way to predict classes. They are based on a statistical classification technique known as Bayes Theorem. They are naïve because they assume that the variables are independent from each other.

The calculate the posterior probability of a certain event A to occur given some probabilities of prior events.

$$P(A|R) = \frac{P(R|A)P(A)}{P(R)}$$

P(A): Prior probability of a class.
P(R|A): Likelihood the probability of predictor given class.
P(R): Prior probability pf predictor.
P(A|R): Posterior probability of class A given the predictor R.

Step to preform

To use the Naïve Bayes algorithm to solve classification problems like deciding if text is positive or negative.

- Convert the dataset into a frequency table.
- Find the probabilities of the events to occur.
- Computes the posterior probability of each class.
- The class with the highest posterior probability is the prediction.

Strengths & Weaknesses

<u>Strengths:</u>

- Easy and quick to implement.
- If the conditional independence holds then it will converge quickly.
- Need less training data.
- Scalable.

- Not sensitive to irrelevant features.

Weaknesses:

- The naïve assumption of independent features is unlikely in the real world.

Implementation

In this project we will develop a Naïve Bayes model that classify text (twits) sediment as positive or negative. It will be based on the training data from the sendiment140 project (see ref. for details).

This is a supervised binary classification problem as the texts (twits) are either positive or negative. We will provide a labeled dataset to train the model.

Preview of the data

	Labels	Id	Date	Query	User	Text
0	0	2205441133	Wed Jun 17 04:44:48 PDT 2009	NO_QUERY	Julie_oh	@tiedyeina lucky you! 6 more days
1	0	2205441225	Wed Jun 17 04:44:49 PDT 2009	NO_QUERY	KatieBug1112	Morning everyone! :-D not in a good mood righ...
2	0	2205441321	Wed Jun 17 04:44:50 PDT 2009	NO_QUERY	jbh_dc	Back in rainy reston ... With a wife who has t...
3	0	2205441485	Wed Jun 17 04:44:51 PDT 2009	NO_QUERY	aaakritiLove	@jysla :S:S whats wrong, dear?
4	0	2205441608	Wed Jun 17 04:44:52 PDT 2009	NO_QUERY	JackyDouglas	Hates the rain!

Labels: The sediment 0 for positive and 4 for negative.
Id: The id of the twitter.
Date: The date of the twit.
User: The name of the twitter.
Text: The text of the twit.

Bag of words

The bag of words process is used when you have a collection of text data that needs to be processed. You take each word and count its frequency within a piece of text. Each word is treated independently, and the order is irrelevant.

I use CountVectorizer to convert the list of twits into a matrix with each twit been a row and each word been a column. The corresponding row:column value is the frequency of the occurrence of each word in that twit.

	00	000	0000r0cx	0007	000th	00am	01	01000101	01614948343	018	...	½rmï	½s	½se	½stand	½t	½tï	½ve	½y	½ï	ã¼ã
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

With the large amount of training data and the various characters people use in twits we have a lot of strange word in the column. It also means we have a lot of column 27387.

Count Vectorizer

I use the sklearn.feature_extraction.text.CountVectorizer to

- Separate the string into individual words and give each word (column) an integer ID.
- Count the occurs of each word (column) in each twit (row).
- Covert all words to lower case.
- Ignore all punctuation.
- Ignore all stop words.

Split Dataset

I split the dataset into four buckets
X_train: Training data for the text (twit) column.
y_train: Training data for the label column.
X_test: Testing data for the text (twit) column.
y_text: Testing data for the label column.

Number of rows in the total set: 17952
Number of rows in the training set: 13464
Number of rows in the test set: 4488

After splitting the dataset, we need to convert the data into the matrix format just like we did in the sample above using CountVectorizer. For the training data we need to fit first, so the model can learn a vocabulary and then transform to the matrix view. For the testing dataset we only need to transform to the matrix view.

Model

For this project I used the multinomial Naïve Bayes implementation because it is suitable for classifications with discrete features like word counts for text classification.

```
def fitNaiveBayes(self):
    self.naive_bayes = MultinomialNB()
    self.naive_bayes.fit(self.training_data, self.y_train)
```

Now that I've trained the model with the training dataset I can use the testing dataset to make predictions.

```
def predict(self):
    self.predictions = self.naive_bayes.predict(self.testing_data)
```

Evaluation

And final now that we have predictions I can evaluate the model to check the accuracy of my baseline model. There are several ways to evaluate a model.

Accuracy: is the fraction of predictions our model got right (e.g. what proportion of twits we predicted/classified as positive or negative were actually positive or negative).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision: tells us what proportion of positive identifications was correct (e.g. what proportion of actually positive twits we predicted as positive twits).

$$Precision = \frac{TP}{TP + FP}$$

Recall: tells us what proportion of positives was identified correctly (e.g. what proportion of all positive twits did we predict as positive)

$$Recall = \frac{TP}{TP + FN}$$

F1 Score: is a weighted average of precision and recall ranging from 0 to 1.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

TP: True Positive
TN: True Negative
FP: False Positive
FN: False Negative

Scores

Accuracy score: 0.7667112299465241
Precision score: 0.7834830979888746
Recall score: 0.7719224283305227
F1 score: 0.7776598003822467

Conclusion

References

1. Sentiment140: http://help.sentiment140.com/for-students/
2. Twitter Search API: https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets
3. Kaggle: https://www.kaggle.com