Naive Bayes

Jason Shipp

July 9, 2018

Concept

In an ideal world the value of any given feature would be totally independent from the value of another feature for a given data point. If this were the case we could use Bayes Law/Rule to compute the missing / predicted value with certainty after working out the subsequent relationship between the features. As it stands however this is not the case, and frequently there are complex underlying relationships in our data that are impossible to model on computers. It can, however, be usefult to pretend that these relationships are independent and apply Bayes Law anyway. This is where Naive Bayes comes in!

Multinomial Naive Bayes

Math Involved

Bayes Rule

$$\text{Let } x = (x_1, x_2, ..., x_n) p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \text{Read } p(y|x) \text{as the probility of y given that x has occurred} \tag{1}$$

Naive Bayes Assumption

$$C_k = argmaxp(C_k) \prod_{i=1}^{n} p(x_k|C_k) \tag{2}$$

Code

Scikit-Learn

```python
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, GaussianNB
from sklearn import metrics, preprocessing

#SCIKIT-LEARN PORTION
#THE FOLLOWING CODE IS ADAPTED FROM RITCHIE NG AND DOES NOT BELONG TO THE REPO OWNER
#Resource: https://www.ritchieng.com/machine-learning-multinomial-naive-bayes-vectorization/
#Read in the data into a pandas dataframe
dataFile = 'https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv'
features = ['label', 'message']
sms = pd.read_table(dataFile, header=None, names=features)

#If you want to see it's size
#texts.shape

#If you want to see its top 5 or 10 or whatever
#texts.head()

#If you want to see the unique labels
#texts.label.unique()

#Lets convert the discrete "lable" into a numerical value we can work with
#Some examples will show you doing this as a .map and then doing it manually but this
#Should work in the general sense
le = preprocessing.LabelEncoder()
sms['labelNum'] = le.fit_transform(sms['label'])


X = sms.message
y = sms.labelNum

#Split into training and test states, you can set a random_state to have repeatable
#occurrences
X_train, X_test, y_train, y_test = train_test_split(X, y)
print(y_test.head())
#Here we are going to use a "Count Vectorizer" to create a bag of words model
#since we are dealing with textual data
#This will be a "sparse matrix" and its best to let sklearn/pandas handle this as
#it knows how to store it in an effecient manner
vect = CountVectorizer()
X_train_dtm = vect.fit_transform(X_train)
X_test_dtm = vect.transform(X_test)

#Let's create a model
nb = MultinomialNB()
nb.fit(X_train_dtm, y_train)

#Here is where we make an actual prediction on our test data, easy right
y_pred = nb.predict(X_test_dtm)

#How'd we do?
print('Multivariate Naive Bayes')
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print('Precision: ', metrics.precision_score(y_test, y_pred))
print('F1 Score: ', metrics.f1_score(y_test, y_pred))

#Top left is True Negative predictions, top right is False Positives
#Bottom left is False Negatives, bottom right is True Positives
print(metrics.confusion_matrix(y_test, y_pred))
```

```

#If you want to test the probability a sentence will have a certain category you can
#Use this, however this really isn't a very good use of naive bayes
#y_pred_prob = nb.predict_proba(X_test_dtm)[:, 1]

#You can also use Gaussian NB if our matrix wasn't sparse.
'''
gnb = GaussianNB()
gnb.fit(X_train_dtm, y_train)
y_pred = nb.predict(X_test_dtm)
print('Gaussian Naive Bayes')
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print('Precision: ', metrics.precision_score(y_test, y_pred))
print('F1 Score: ', metrics.f1_score(y_test, y_pred))
'''

#TENSORFLOW Portion
#This is usually not the use for Tensorflow, but you can kind of force it if you want
class NaiveBayesClassifier:
    dist = None

    def fit(self, X, y):
        ydist = np.unique(y)
        count = np.array([
            [x for x,t in zip(X,y) if t == c]
            for c in ydist])

        mean, var = tf.nn.moments(tf.constant(count), axes=[1])
        self.dist = tf.distributions.Normal(loc=mean, scale=tf.sqrt(var))

    def predict(self, X):
        assert self.dist is not None
        nb_classes, nb_features = map(int, self.dist.scale.shape)
        prob = tf.reduce_sum(
                self.dist.log_prob(
                    tf.reshape(
                        tf.tile(X, [1, nb_classes]), [-1, nb_classes, nb_features])),
                    axis=2)
        priors = np.log(np.array([1.0/nb_classes] * nb_classes))
        joint_likelihood = tf.add(priors, cond_probs)
#TODO
        norm_factor = tf.reduce logsumexp(joint_likelihood, axis=1,
                keep_dims=True)
        log_prob = joint_likelihood - norm_factor
        return tf.exp(log_prob)

#TODO Use classifier, later
```