IT159IU: Artificial Intelligence
Lab#6&7: Spam Filter

Lab Objective: *Create a Naïve Bayes Classifier. Use this classifier, and Sklearn's premade classifier to make an SMS spam filter.*

About Naïve Bayes

Naïve Bayes classifiers are a family of machine learning classification methods that use Bayes' theorem to probabilistically categorize data. They are called naïve because they assume independence between the features. The main idea is to use Bayes' theorem to determine the probability that a certain data point belongs in a certain class, given the features of that data. Despite what the name may suggest, the naïve Bayes classifier is not Bayesian method. This is because naïve Bayes is based on likelihood rather than Bayesian inference.

While naïve Bayes classifiers are most easily seen as applicable in cases where the features have, ostensibly, well defined probability distributions (such as classifying sex given physical characteristics), they are applicable in many other cases. While it is generally a bad idea to assume independence naïve Bayes classifiers are still very effective, even when we can be confident there is nonzero covariance between features.

The Classifier

You are likely already familiar with Bayes' Theorem, but we will review how we can use Bayes' Theorem to construct a robust machine learning model.

Given the feature vector of a piece of data we want to classify, we want to know which of all the classes is most likely. Essentially, we want to answer the following question

$$\text{argmax}_{k \in K} P(C = k|\mathbf{x}) \tag{1.1}$$

where C is the random variable representing the class of the data. Using Bayes' Theorem, we can reformulate this problem into something that is actually computable. We find that for any $k \in K$ we have

$$P(C = k|\mathbf{x}) = \frac{P(C = k)P(\mathbf{x}|C = k)}{P(\mathbf{x})}$$

Now we will examine each feature individually and use the chain rule to expand the following expression

$$P(C = k)P(\mathbf{x}|C = k) = P(x_1,...,x_n,C = k)$$

$$= P(x_1|x_2,\ldots,x_n,C = k)P(x_2,\ldots,x_n,C = k) = \ldots$$
$$= P(x_1|x_2,\ldots,x_n,C = k)P(x_2|x_3,\ldots,x_n,C = k)\cdots P(x_n|C = k)P(C = k),$$

and applying the assumption that each feature is independent we can drastically simplify this expression to the following

$$P(x_1|x_2,\ldots,x_n,C = k)\cdots P(x_n|C = k) = \prod_{i=1}^{n} P(x_i|C = k)$$

Therefore, we have that

$$P(C = k|\mathbf{x}) = \frac{P(C = k)}{P(\mathbf{x})}\prod_{i=1}^{n} P(x_i|C = k)$$

which reforms Equation 1.1 as

$$\text{argmax}_{k\in K} P(C = k)\prod_{i=1}^{n} P(x_i|C = k) \quad (1.2)$$

We drop the $P(\mathbf{x})$ in the denominator since it is not dependent on k.

This problem is approximately computable, since we can use training data to attempt to find the parameters which describe $P(x_i|C = k)$ for each i,k combination, and $P(C = k)$ for each k. In reality, a naïve Bayes classifier won't often find the actual correct parameters for each distribution, but in practice the model does well enough to be robust. Something to note here is that we are actually computing $P(C = k|\mathbf{x})$ by finding $P(C = k,\mathbf{x})$. This means that naïve Bayes is a generative classifier, and not a discriminative classifier.

AI Lab: SMS Spam Filter with Naïve Bayes Classifier

- Build a **Naïve Bayes classifier from scratch**
- Use **Scikit-learn's Naïve Bayes classifier** for comparison
- Apply both models on an **SMS Spam dataset from : https://raw.githubusercontent.com/justmarkham/pycon-2016-tutorial/master/data/sms.tsv**
- Evaluate performance (accuracy, precision, recall)

A spam filter is just a special case of a classifier with two classes: spam and not spam (or ham). We can now describe in more detail how we are to calculate Equation 1.2 given that we know what the features are. We can use a labeled training set to determine $P(C = \text{spam})$ the probability of spam and $P(C = \text{ham})$ the probability of ham. To do this, we will assume that the training set is a representative sample and define

$$P(C = \text{spam}) = \frac{N\text{spam}}{N\text{samples}} \quad (1.3)$$

and

$$P(C = \text{ham}) = \frac{N\text{ham}}{N\text{samples}} \quad (1.4)$$

Using a bag of words model, we can create a simple representation of $P(x_i|C = k)$ where x_i is the i^{th} word in a message, and therefore **x** is the entire message. This results in the simple definition of

$$P(x_i|C = k) = \frac{N_{\text{occurrences of } x_i \text{ in class } k}}{N_{\text{words in class } k}}. \qquad (1.5)$$

Note that the denominator in Equation 1.5 is not the number of unique words in class *k*, but the total number of occurrences of any word in class *k*. In the case, we have some word x_u that is not found in the training set, we can may choose $P(x_u|C = k)$ so that the computation is not effected, e.g., letting $P(x_u|C = k) = 1$ for unique words.

A First Model

When building a naïve Bayes classifier, we need to choose what probability distribution we believe our features to have. For this first model, we will assume that the words are a categorically distributed random variable. This means the random variable may take on say *N* different values, each value has a certain probability of occurring. This distribution can be thought of as a Bernoulli trial with *N* outcomes instead of 2.

In our situation we may have *N* different words which we expect may occur in a spam or ham message, so we need to use the training data to find each word and its associated probability. In order to do this, we will make a DataFrame that will allow us to calculate the probability of the occurrence of a certain word x_i based on what percentage of words in the training set were that word x_i. This DataFrame will allow us to compute more easily Equation 1.5, assuming the words are categorically distributed. While we are creating this DataFrame, it will also be a good opportunity to compute Equations 1.3 and 1.4.

Throughout the lab we will use an SMS spam dataset contained in *SMSSpamCollection.csv*. We will use portions of data to check our progress. And other files that you can and cannot modify as follow.

Files you'll edit:

`naiveBayesClassifier.py`	Where all your classification algorithms will reside.

Files you should look at but NOT edit:

`spam_filter.py`	The main file that runs NB to solve the spam filter

Training The Model

Problem 1. Create a class NaiveBayesFilter, with an `__init__()` method as defined in NaiveBayesFilter. Add a fit() method which takes as arguments X, the training data, and y the training labels. In this case, X is a pandas.Series containing strings that are SMS messages. For each message in X count the number of occurrences of each word and record this information in a DataFrame.

The final form of the DataFrame should have a column for each unique word that appears in any message of X as well as a Label column, you may also include any other columns you think you'll need. Each row of the DataFrame corresponds to a row of X, and records the number of occurrences of each word in a given message. The Label column records the label of the message.

Save this DataFrame as self.data.

Predictions

Now that we have implemented the fit() method, we can begin to classify new data. We will do this with two methods, the first will be a method that calculates $P(S|\mathbf{x})$ and $P(H|\mathbf{x})$, and the other will determine the more likely of the two and assign a label. While it may seem like we should have $P(C = S|\mathbf{x}) = 1 - P(C = H|\mathbf{x})$, we do not. This would only be true if we assume the S and H are independent of $\mathbf{x}$. It is clear that we shouldn't make this assumption, because we are trying to determine the likelihood of S and H based on what $\mathbf{x}$ tells us. Therefore, we must compute both $P(C = S|\mathbf{x})$ and $P(C = H|\mathbf{x})$.

Hint

Create a class `NaiveBayesFilter` with the following:

- `__init__()` to initialize internal attributes
- `fit(X, y)`:
  - Process each SMS message in `X` (which is a series of text strings)
  - Count the number of occurrences of each word
  - Combine the results into a DataFrame with columns representing words and a `Label` column
  - Store the result in `self.data`

```python
def __init__(self):
    self.data = pd.DataFrame()
    self.vocabulary = set()  # returns tuple of unique words
    self.p_spam = 0  # Probability of Spam
    self.p_ham = 0  # Probability of Ham
    # Initiate parameters
    self.parameters_spam = {unique_word: 0 for unique_word in self.vocabulary}
    print('parameters_spam: ', self.parameters_spam)
    self.parameters_ham = {unique_word: 0 for unique_word in self.vocabulary}
    print('parameters_ham: ', self.parameters_spam)
```

```python
def fit(self, X, y):
    "*** YOUR CODE HERE ***"
    rows = []
    vocabulary = set()
    self.word_counts_spam = {}
    self.word_counts_ham = {}
    self.total_words_spam = 0
    self.total_words_ham = 0

    for message, label in zip(X, y):
        word_counts = {}
        for word in message:
            vocabulary.add(word)
            word_counts[word] = word_counts.get(word, 0) + 1

            if label == 'spam':
                self.word_counts_spam[word] = self.word_counts_spam.get(word, 0) + 1
                self.total_words_spam += 1
            else:
                self.word_counts_ham[word] = self.word_counts_ham.get(word, 0) + 1
                self.total_words_ham += 1

        word_counts['Label'] = label
        rows.append(word_counts)

    self.vocabulary = vocabulary
    self.data = pd.DataFrame(rows).fillna(0)

    total = len(self.data)
    self.p_spam = (self.data['Label'] == 'spam').sum() / total
    self.p_ham = (self.data['Label'] == 'ham').sum() / total

    return self.data
```

Problem 2. Implement the `predict_proba()` method in your naïve Bayes classifier. This should take as an argument X, the data that needs to be classified. For each message x in X compute $P(S|\mathbf{x})$ and $P(H|\mathbf{x})$ using Equations 1.3, 1.4, and 1.5.

The method should return a (Nx2) list, where N is the length of X. The first column corresponds to $P(C = H|\mathbf{x})$, and the second to $P(C = S|\mathbf{x})$.

```python
def predict_proba(self, X):
    proba = []
    "*** YOUR CODE HERE ***"
    alpha = 1
    V = len(self.vocabulary)

    for message in X:
        log_p_spam = np.log(self.p_spam)
        log_p_ham = np.log(self.p_ham)

        for word in message:
            if word in self.vocabulary:
                count_spam = self.word_counts_spam.get(word, 0)
                prob_word_spam = (count_spam + alpha) / (self.total_words_spam + alpha * V)
                log_p_spam += np.log(prob_word_spam)

                count_ham = self.word_counts_ham.get(word, 0)
                prob_word_ham = (count_ham + alpha) / (self.total_words_ham + alpha * V)
                log_p_ham += np.log(prob_word_ham)

        max_log = max(log_p_spam, log_p_ham)
        p_spam = np.exp(log_p_spam - max_log)
        p_ham = np.exp(log_p_ham - max_log)
        total = p_spam + p_ham

        norm_p_spam = p_spam / total
        norm_p_ham = p_ham / total

        proba.append([norm_p_ham, norm_p_spam])

    return proba
```

Problem 3. Implement the `predict()` method in your naïve Bayes classifier. This should take as an argument X, the data that needs to be classified. Implement equation 1.2 and return a list of labels that predicts each message in X.

For example**:**

```
#create the filter
NB = NaiveBayesFilter()
#fit the filter with train data
NB.fit(X_train, y_train)
#test the predict function with five data points in test data
NB.predict(X_test[500:505])
```

Output: ['ham', 'ham', 'ham', 'ham', 'spam']

```python
def predict(self, X):
    prob = []
    "*** YOUR CODE HERE ***"
    predictProba = self.predict_proba(X)
    for p_ham, p_spam in predictProba:
        if p_spam > p_ham:
            prob.append('spam')
        else:
            prob.append('ham')
    return prob
```

Problem 4. Implement the `score()` method in your naïve Bayes classifier. This should take two arguments as a list of predicted labels (returns from `predict()` method) and a list of true labels in X and return the matched labels between them which refer as recall metric.

For example:

```
#test the predict function with five data points in test data
predict_labels = NB.predict(X_test[500:505])
#calculate the score
recall = NB.score(y_test[500:505], predict_labels)
print("recall of NB: ", recall)
```

Output: recall of NB: 0.8

```python
def score(self, true_labels, predict_labels):
    recall = 0
    "*** YOUR CODE HERE ***"
    true_positives = sum((t == "spam") and (p == "spam") for t, p in zip(true_labels, predict_labels))
    possible_positives = sum(t == "spam" for t in true_labels)
    if possible_positives == 0:
        return 0.0
    recall = true_positives / possible_positives
    return recall
```

```
------------------BEFORE CLEANING DATA------------------------
  Label                                                SMS
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
Total message:  5572
training_test_index:  4458
Length of test data:  1114
----AFTER CLEANING AND SPLIT TRAIN AND TEST DATA----
0                  [yep,, by, the, pretty, sculpture]
1    [yes,, princess., are, you, going, to, make, m...
2                     [welp, apparently, he, retired]
3                                             [havent.]
4    [i, forgot, 2, ask, ü, all, smth.., there's, a...
Name: SMS, dtype: object
parameters_spam:  {}
parameters_ham:  {}
recall train of NB:  0.9633333333333334
recall test of NB:  0.8979591836734694
```

Problem 5: practice again with Sklearn MultinomialNB

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split

# Đọc dữ liệu
df = pd.read_csv("SMSSpamCollection.csv", sep='\t', header=None, names=['label', 'message'])
df.columns = ['label', 'message']
df['label'] = df['label'].map({'ham': 0, 'spam': 1})

# Chia dữ liệu
X_train, X_test, y_train, y_test = train_test_split(df['message'], df['label'], test_size=0.2, random_state=42)

# Tiền xử lý: chuyển văn bản thành ma trận số lượng từ
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)

# Khởi tạo và huấn luyện mô hình MultinomialNB
clf = MultinomialNB()
clf.fit(X_train_counts, y_train)

# Dự đoán
y_pred = clf.predict(X_test_counts)

# Đánh giá recall
recall = recall_score(y_test, y_pred)
print("Recall of MultinomialNB:", recall)

```

```
PS E:\Homework\AI\Lab67> & C:/Python312/python.exe e:/Homework/AI/Lab67/nb_sklearn_comparison.py
Recall of MultinomialNB: 0.9395973154362416
```

------END----