CS5785 Homework 3

Due date: December 1

The homework is split into programming exercises and written exercises.You should turn in an electronic copy of your solutions to the homework. Please submit your homework to CMS. You are responsible for submitting clear, organized answers to the questions. Please include all relevant information for a question, including text response, equations, figures, graphs, etc. Please pay attention to the discussion board for relevant information regarding updates, tips, and policy changes. You are required to work in groups of 2 (unless given an exemption from the course staff). Version 1.

In this exercise, we're going to try and predict the decade of a movie given its plot synopsis from IMDB. The idea is that as our cultural interests shift over time, different decades have different iconic elements in the movies' plot that show up as different words. There are two main issues: how to represent the movie in feature space, and how to calculate the year's posterior probability.

1 Naive Bayes Classification Design

Our goal is to find the predicted decade Y given a movie's features X. We will treat this like a classification problem. Before we mess with the data, it's important to design a good probabilistic framework for how we might attack this problem using a Naive Bayesian classifier.

1a. Suppose we already know $P(X)$, $P(Y)$, and $P(X|Y)$ **Please derive** the value of $P(Y|X)$ given these values. Show your work.
Hint: Use Bayes' Rule to express the latter in terms of the former.
Hint 2: $P(A|B) = P(A \cap B)/P(B)$, which is always true for any A and B. You can use this to derive Bayes' Rule.

1b. **Short answer:** Is your equation always true, or does it depend on certain assumptions? Why or why not? If the equation is not always true, which step in your derivation discarded information?

It's time to pick a feature representation. We can get pretty far with a simple "Bag of Words" representation, so let's start there. Let W be the set of all possible words that appear in all movie plot summaries, but with some normalization: each word in the set is lowercase and all punctuation is stripped.
Given movie i, let feature X_j = (number of times word W_j appears in i's plot summary), so the entire feature vector for movie i is $X = \left[X_1, X_2, \ldots, X_{|W|}\right]^T$. Now each movie's year depends on several random variables, $X_1, X_2, \ldots, X_{|W|}$.

1c. This time, **derive** $P(Y|X_1, X_2, \ldots, X_{|W|})$ in terms of $P(Y)$ and $P(X_1, X_2, \ldots, X_{|W|}|Y)$. Hint: It should look almost the same as before.

1d. **Short answer:** In English, what does $P(Y)$ mean? What does $P(Y|X_i)$ mean, and what does $P(X_i|Y)$ mean? Give a small one-liner of code or an equation to compute each of the above quantities. It doesn't have to be efficient and it doesn't have to stand alone; we just want you to get the point across. You can assume you already loaded all of the data into variables of your choice.

1e. Derivation: The "Probabilistic Chain Rule" states that $P(A_1, A_2, \ldots|B) = P(A_1|B)P(A_2, \ldots|B, A_1)$. Use this fact repeatedly to rewrite your above equation by turning $P(X_1, X_2, X_3, \ldots, X_{|W|}|Y)$ into the product of several conditional probabilities. There should be $|W|$ of them in your result.

1f. Short answer: Now you have a (slightly more complicated) equation for $P(Y|X_1, \ldots, X_{|W|})$. Is this equation still always true, or does it depend on certain assumptions? Why or why not? If the equation is not always true, which step in your derivation discarded information?

"Naive" Bayes Assumption. One problem with your equation is that all of the conditional probabilities like $P(X_5|Y, X_1, X_2, X_3, X_4)$ are hard to compute at test time, especially since $|W|$ is so large. To simplify things, let's make the additional *independence* assumption that $P(X_n|Y, X_1, X_2, \ldots, X_{n-1}) = P(X_i|Y)$.

1g. Short answer: To convince you that this assumption actually does make things easier, write a snippet of code that *would* compute $P(X_n|Y, X_1, \ldots, X_{n-1})$. Why is this value hard to compute at test time?

1h. Short answer: What does the "Naive Bayes" assumption mean? When is it violated?

1i. Short answer: How does this change your equation for $P(Y|X_1, \ldots)$? Rewrite your final equation using this assumption.

2 TIME FOR SOME DATA

We'll use actual plot summaries from the Internet Movie Database. Visit `http://www.imdb.com/interfaces` and download `plot.list.gz` from the website. Here is some python code to parse the dataset: `https://gist.github.com/gcr/061fbf19b9f7e15a633c`

- **Optional Karma exercise**: Make the code work better. Right now, some movies are unnecessarily skipped because they fail to parse. If you get better results, tell us in Piazza!

2a. Plot the probability mass function (PMF) of $P(Y)$ across the entire dataset. You can use a histogram.

2b. Plot the PMF of $P(Y|X_{\texttt{"radio"}} > 0)$ across the entire dataset.

2c. Plot the PMF of $P(Y|X_{\texttt{"beaver"}} > 0)$ across the entire dataset.

2d. Plot the PMF of $P(Y|X_{\texttt{"the"}} > 0)$ across the entire dataset.

One issue is that the above dataset is *unbalanced* since there are more new movies than old movies. To account for this, uniformly sample 7,000 movies for each decade between 1930 and 2010. There should be 56,000 movies in total in your revised dataset.

2e. Plot the PMF of $P(Y|X_{\texttt{"radio"}} > 0)$ across the balanced dataset.

2f. Plot the PMF of $P(Y|X_{\texttt{"beaver"}} > 0)$ across the balanced dataset.

2g. Plot the PMF of $P(Y|X_{\texttt{"the"}} > 0)$ across the balanced dataset.

2h. **Short answer:** How do these plots compare? Is it easier to see which words are likely to be informative of decade?

- **Please only use the balanced dataset for the rest of the assignment.** It makes reporting and evaluation much simpler since we won't have to worry about the prior year distribution.

We need an experiment protocol, so split this dataset into training and test sets. You can do something simple like using even/odd movies as training/testing, or for extra karma, you can perform everything below using cross validation. Be sure that the distribution of years in your training and testing sets is balanced!

2i. **Implement** code that calculates $P(Y|X_1,\ldots,X_{|W|})$ using your equation for (**1i.**). Please don't use `SKLearn`'s implementation for this; write it yourself. Also, calculate training probabilities just from the training set.
Hint: It shouldn't be very complicated. My implementation has less lines than the parsing code. One simple data representation is to use a dictionary mapping decades to (dictionary mapping word to $P(X_i|Y_j)$). This is just a suggestion, of course – you are free to do this however you please.

To predict a movie's decade $\hat{y}$, take the most likely year as chosen by our model:

$$\hat{y} = \arg\max_y P(Y = y|X_1, X_2, \ldots) \tag{1}$$

There are three tricks that you should know about:

- **Ignoring the denominator.** Your equation for $P(Y|X_1,\ldots)$ is (*something*)$/Z$, but the denominator Z is independent of the year. This means

$$\arg\max_y P(Y = y|X_1, X_2, \ldots) \quad = \quad \arg\max_y Z \cdot P(Y = y|X_1, X_2, \ldots), \tag{2}$$

  so you can throw the denominator away.

- **Dirichlet prior.** If $P(X_i|Y) = 0$, then your final probability will become 0, which we don't want. In these cases, just pretend the probability is 0.0001, or whatever your favorite small number is.

- **Avoiding underflow.** Your probabilities will be too small to represent as a floating point number, even with all 64 bits of precision. This means all of your probabilities may underflow to 0 again. One trick is to perform all of your calculations in *log-space*:

$$\arg\max_y P(Y = y|X_1, X_2, \ldots) \quad = \quad \arg\max_y \log\big(P(Y = y|X_1, X_2, \ldots)\big). \tag{3}$$

  This is helpful because

$$\log\left(\prod_i P_i\right) = \sum_i \log P_i, \tag{4}$$

  and taking the sum of several small numbers is much more numerically stable than multiplying several small numbers.

2j. For each of the following movies, **plot** $P(\hat{Y} = y|\text{Movie} = m)$ as a histogram over y. Also, state the actual predicted value $\hat{y}$. Which movies can we correctly classify?

- Finding Nemo
- The Matrix
- Gone with the Wind
- Harry Potter and the Goblet of Fire
- Avatar

2k. **Short answer:** What is the accuracy of your classifier? What is the expected accuracy of random chance? What *would* be the expected accuracy of a uniformly random guess in the original *unbalanced* dataset?

2l. **Plot** a cumulative match curve (CMC). To do this, vary $k = (1,2,3,4,5,\ldots)$ along the X-axis. The Y axis should show the frequency that the correct label is within your classifier's top rank-k choices for the label. E.g. if the curve passes through $(3, 0.9)$, then for 90% of the test movies, the correct decade is one of your classifier's top three guesses.

2m. **Plot** a confusion matrix showing your classifier's results. Cell i, j of a confusion matrix denotes how often a movie with decade L_i was accidentally chosen as L_j. In a balanced test dataset, the rows should sum to a constant. Which decades are often confused with each other? In your opinion, why might that happen?

3 Finding Iconic Words

Now that we have a classifier that works rather well for guessing a movie's year, let's pull it apart and see what makes it work so well. What words are iconic for each decade? One crude measure of word w's importance in decade y is its probability within that decade divided by the minimum probability across all other decades:

$$f(w,y) = \frac{P(X_w > 0 | Y = y)}{\min_y P(X_w > 0 | Y = y)} \tag{5}$$

The idea behind this measure is simple: words that maximize $f(w,y)$ are common to that year and uncommon in other years, so they should influence $\hat{y}$ the most.

3a. What are the 10 most informative words for each decade?

3b. How much does performance degrade if you strip out each decade's 100 most informative words from each testing summary?

4 The Moment of Truth

4a. **Experiment**: Compare your classifier's performance to `sklearn`'s implementation in `sklearn.naive_bayes`. For feature extraction, use `sklearn.text.CountVectorizer`. Try to get within ±3% of the "gold standard" accuracy. (If you just used `sklearn`'s classifier earlier, this entire homework would take like two lines of code. That's why we had you write your own.)

4b. Long answer: Scikit-learn has three different Naive Bayesian classifiers: `GaussianNB`, `MultinomialNB`, and `BernoulliNB`. They're all slightly different. In your own words, explain the differences in the assumptions they each make. Which one is closest to our approach and why? Which assumption seems the most appropriate for our given feature representation? How does our classifier differ from each of these choices? Feel free to peruse the documentation and implementation of all three classifiers on github to answer this question.

5 MAXIMUM MARGIN CLASSIFICATION

Naive Bayes isn't the only classifier in town. Now it's time to break out the full arsenal provided by `sklearn` and see if other classifiers work well for this problem.

5a. Implement a feature extraction strategy to convert the movie into an actual feature vector. **Describe** the strategy you choose and what the dimensionality of your feature vectors are.

Careful – this is easier said than done! Before, with careful choice of data structures, we were able to use a *sparse* feature representation that only stored values for nonzero elements. Unfortunately, many of `sklearn`'s classifiers require a dense feature vector. Since each movie summary is unlikely to contain most of the 443,000 unique words in IMDB, most of these elements should be 0. If you simply use one dimension for each unique word, you'll wind up with a matrix of size $56,000 \times 443,000$, which takes 92GB to store. There are several options available to you. Pick one and describe it in your writeup:

- Only take k of the **most common** words in the dataset, for a k of your choice (justify it);
- Only take k **random** words;
- Use some form of **dimensionality reduction** after computing the feature vectors, like random projection. One simple dimensionality reduction technique is called **the hashing trick**, where each element of the feature vector is the sum of several (random) word counts rather than assigning each unique word count its own feature.

5b. Implement: Many maximum-margin classifiers expect to have *preprocessed* data. To do this, "zero" your data by subtracting off each feature's mean and "rescale" it by dividing each feature by its standard deviation. See `sklearn.preprocessing.StandardScaler`

5c. Experiment: Once you have the features and labels for your training/testing sets, trying a battery of `sklearn`'s classifiers is a piece of cake (ie. one-line change). Try each of the following classifiers on your data and compare its accuracy to your Naive Bayesian model earlier:

- `linear_model.SGDClassifier`, a maximum margin classifier;
- `svm.LinearSVC`, another max-margin classifier;
- `svm.SVC(kernel='rbf')`, which is still "technically" max-margin;
- `linear_model.Perceptron(penalty='l1')`
- `linear_model.Perceptron(penalty='l2',n_iter=25)`
- `svm.neighbors.NearestNeighbors`, our friendly neighborhood KNN algorithm;
- Another classifier of your choice.

Do any of them outperform your Naive Bayes classifier? If so, why?

5d. **Short answer:** For each of those classifiers, read the documentation (and implementation on `https://github.com/scikit-learn/scikit-learn/tree/master/sklearn`, if you like), and describe in one or two short sentences what that classifier does and why it might work. How does that classifier do its training? How does it do its testing? Do you think training/testing is computationally easy or difficult? If we ask you to use specific parameters, what do they mean?

5e. **Short answer:** Pick two of the above classifiers that do much better or much worse than the rest. Think about why these two models might do better/worse and explain your reasoning in your own words. For example, is there some property of the model/training/representation that might make it particularly well suited (or ill-suited) for this task?

5f. **Plot:** Pick your favourite model from the above list. How does its performance degrade with respect to the dimensionality of the input? To find out, vary k in your featurization strategy for (5a) (or perform some other dimensionality reduction) and repeat your experiment for low-dimensional, medium-dimensional, and high-dimensional features. Does this model get worse as dimensionality decreases, or is it reasonably robust?

5g. **Long answer:** In class, we discussed maximum margin *binary* classifiers that just separate positive-labeled instances from negative-labeled ones. However, if you weren't paying attention when you blindly tried `sklearn`'s classifiers, you might not have noticed that they work just fine on our data even though it has 9 labels. That's strange—a single hyperplane (like the kind we discussed in class) can't separate the space into 9 regions. *Something funny must be going on.*

For each of these classifiers, do some research on how and why they are able to predict a multi-class label even though most of those classifiers are based on binary hyperplanes.

You are free to rely on whatever sources you like: `sklearn`'s implementation on github, its documentation, any papers referenced therein, Wikipedia, etc. If you don't know how to begin, start by researching the difference between one-vs-one and one-vs-rest classification and see if you can't figure out which classifier is using which strategy.

- **Optional Karma exercise:** Try using some other feature representation. Do n-grams improve performance? What about TF/IDF features? If you try something different, explain how it works in your writeup.
- **Optional Karma exercise:** How does your Naive Bayes algorithm compare to your decision tree classifier in the previous homework?

A NOTE ABOUT CITATIONS

Some of these questions ask you to explore already-implemented systems, such as `sklearn`'s source code or documentation. Please quickly cite whatever Internet sources you use to answer these problems. If you used a tool or some other resource to help out with a derivation on a written question, please cite it as well. It is not acceptable to use sources that simply give you the answer ripe for copying.

A NOTE ABOUT KARMA

Assignments may include optional problems called karma problems. These problems exist to provide an extra challenge for those who are up to it. Karma problems will be graded, but will not affect your score — they mostly just give good karma. Students who do karma problems will be noticed, and doing them may be taken into account when assigning final grades. For example, karma might make a difference if you are right on the line between a B+ and an AâĂŞ. However, spending your time on regular problems is almost always a more effective way to improve your overall score. Tackle karma problems only after you are sure you have the rest of the assignment well in hand.