SEng 474 / CSc 578D
Data Mining – Fall 2016
Assignment 1

Due: September 27th

Submit a pdf for Q 1-3 and code for Q 4 through ConneX before 11:55pm.

Different marking schemes will be used for undergrad (SEng 474) and grad (CSc 578D) students.
Undergrad students do not have to answer the grad questions.

All code questions use the python and/or the sckit-learn library. You may install it, along with the NumPy and SciPy libraries on your own computer. Alternatively, you can work in the lab.
http://scikit-learn.org/stable/install.html
http://www.numpy.org/

1.

a) (SEng 474: 20 points; CSc 578D: 10 points)
By hand, construct the root and the first level of a decision tree for the contact lenses data (attached to this assignment on connex) using the ID3 algorithm. Show the details of your construction and all your calculations; no points will be given for solutions only.

b) (SEng 474: 5 points; CSc 578D: 5 points)
Using the tree.DecisionTreeClassifier module from python's sckit-learn, fit a tree using the contact-lenses data using *criterion='entropy'*.

Compare the entropy values obtain in part a) with the ones calculated by the sklearn.tree module. Explain in detail why the trees are not the same. You may find the documentation for decision trees helpful:
http://scikit-learn.org/stable/modules/tree.html

Note: You can import the data directly from the 'contact-lenses.arff' file using the *Arff2Skl()* converter from *util2.py* provided with this assignment, using these lines of code:

```
from util2 import Arff2Skl

cvt = Arff2Skl('contact-lenses.arff')
label = cvt.meta.names()[-1]
X, y = cvt.transform(label)
```

2. (SEng 474: 20 points; CSc 578D: 15 points)
Calculate the probabilities needed for Naïve Bayes using the contact lens dataset. Classify: "*prepresbyopic, hypermetrope, yes, reduced, ?"* using your calculated probabilities. Use additive smoothing, as described in class.

3. (Total: SEng 474: 55 points; CSc 578D: 70 points)
a) (SEng 474: 40 points; CSc 578D: 30 points)
Implement Naïve Bayes, assuming that each feature is binary (can only take on values 0 or 1). This is called Bernoulli Naïve Bayes, because each feature is a Bernoulli random variable. Use the skeleton code provided, as we will be using the class for testing. Recall that for Naïve Bayes, the classification decision is:

$$\hat{y} = argmax_y(P(y)\prod_{i=1}^{p} P(x_i|y))$$

where y is the class, p is the total number of attributes (features) in x, and x_i is the ith feature of the vector x.

Using the training data, you must calculate P(y) for each of the classes (y), as well as $P(x_i=0|y)$ and for each y and every feature x_i. Note that $P(x_i=1|y)=1- P(x_i=0|y)$, so you don't need to explicitly calculate that value. Consult the class notes for information on how to calculate these values.

Implement additive smoothing with $\alpha = 1$.

We will be testing your code by creating an instance of MyBayesClassifier and passing it a new binary dataset, so *don't assume* that you know the number of features or the number of classes.

b) **(SEng 474: 15 points; CSc 578D: 10 points)**
Implement additive smoothing https://en.wikipedia.org/wiki/Additive_smoothing so that you can vary α in your code. Create (and hand in) a plot of the accuracy on the test set as a function of α, varying from 0.01 to 1 (use steps of 0.01). Based on your experiment, what is the optimal value for alpha?

c) Graduate students only. **(SEng 474: 0 points; CSc 578D: 30 points)**
Implement a multinomial version of the Naïve Bayes classifier that can accept non-binary input vectors (see skeleton in code). We will rewrite the probability of a feature given the class as:

$$P(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha p}$$

Where N_{yi} is the total number of times feature i appeared for *any* instance with label y, and $N_y = \sum_{i=1}^{p} N_{yi}$ is the total number of times any feature appeared in an instance label y, p is the total number of features, and α is the additive smoothing parameter. Note that

in this implementation, if a feature doesn't appear in a test instance, we ignore it (that is, we don't model $P(x_i|y) = 0$, we just model the existence of features).

Now you should be able to change the CountVectorizer to have parameter binary=false. Which performs better on the test data, the (binary + Bernoulli) setup or (non-binary + multinomial)? Report your accuracies with each setup.