CPSC 340 Assignment 2 (due Friday February 3 at 11:55pm)

The assignment instructions are the same as Assignment 1, except you have the option to work in a group of 2. It is recommended that you work in groups as the assignment is quite long, but please only submit one assignment for the group.

Name(s) and Student ID(s):

1 Naive Bayes

In this section we'll implement naive Bayes, a very fast classification method that is often surprisingly accurate for text data with simple representations like bag of words.

1.1 Naive Bayes by Hand

Consider the dataset below, which has 12 training examples and 3 features:

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \\ \text{not spam} \end{bmatrix}.$$

The feature in the first column is <your name> (whether the e-mail contained your name), in the second column is "pharmaceutical" (whether the e-mail contained this word), and the third column is "PayPal" (whether the e-mail contained this word). Suppose you believe that a naive Bayes model would be appropriate for this dataset, and you want to classify the following test example:

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$

1.1.1 Prior probabilities

Compute the estimates of the class prior probabilities (you don't need to show any work):

- $p(\text{spam})$.
- $p(\text{not spam})$.

1.1.2 Conditional probabilities

Compute the estimates of the 6 conditional probabilities required by naive Bayes for this example (you don't need to show any work):

- $p(\text{<your name>} = 1 \mid \text{spam})$.
- $p(\text{pharmaceutical} = 0 \mid \text{spam})$.
- $p(\text{PayPal} = 1 \mid \text{spam})$.
- $p(\text{<your name>} = 1 \mid \text{not spam})$.
- $p(\text{pharmaceutical} = 0 \mid \text{not spam})$.
- $p(\text{PayPal} = 1 \mid \text{not spam})$.

1.1.3 Prediction

Under the naive Bayes model and your estimates of the above probabilities, what is the most likely label for the test example? (Show your work.)

1.2 Bag of Words

If you run `python main.py 1.2`, it will load the following dataset:

1. `X`: A binary matrix. Each row corresponds to a newsgroup post, and each column corresponds to whether a particular word was used in the post. A value of 1 means that the word occured in the post.
2. `wordlist`: The set of words that correspond to each column.
3. `y`: A vector with values 0 through 3, with the value corresponding to the newsgroup that the post came from.
4. `groupnames`: The names of the four newsgroups.
5. `Xvalidate` and `yvalidate`: the word lists and newsgroup labels for additional newsgroup posts.

Answer the following:

1. Which word corresponds to column 73 of X? (This is index 72 in Python.)
2. Which words are present in training example 803 (Python index 802)?
3. Which newsgroup name does training example 803 come from?

1.3 Naive Bayes Implementation

If you run `python main.py 1.3` it will load the newsgroups dataset, fit a basic naive Bayes model and report the validation error.

The `predict()` function of the naive Bayes classifier is already implemented. However, in `fit()` the calculation of the variable `p_xy` is incorrect (right now, it just sets all values to 1/2). Modify this function so that `p_xy` correctly computes the conditional probabilities of these values based on the frequencies in the data set. Submit your code. Report the training and validation errors that you obtain.

1.4 Runtime of Naive Bayes for Discrete Data

Assume you have the following setup:

- The training set has n objects each with d features.
- The test set has t objects with d features.
- Each feature can have up to c discrete values (you can assume $c \leq n$).
- There are k class labels (you can assume $k \leq n$)

You can implement the training phase of a naive Bayes classifier in this setup in $O(nd)$, since you only need to do a constant amount of work for each $X[i, j]$ value. (You do not have to actually implement it in this way for the previous question, but you should think about how this could be done). What is the cost of classifying t test examples with the model?

2 K-Nearest Neighbours

In *citiesSmall* dataset, nearby points tend to receive the same class label because they are part of the same state. This indicates that a $k-$nearest neighbours classifier might be a better choice than a decision tree (while naive Bayes would probably work poorly on this dataset). The file *knn.py* has implemented the training function for a $k-$nearest neighbour classifier (which is to just memorize the data) but the predict function always just predicts 1.

2.1 KNN Prediction

Fill in the *predict* function in *knn.py* so that the model file implements the k-nearest neighbour prediction rule. You should use Euclidean distance.

Hint: although it is not necessary, you may find it useful to pre-compute all the distances (using the `utils.euclidean_dist_squared`) and to use the numpy's `sort` and/or `argsort`.

1. Hand in the predict function.
2. Report the training and test error obtained on the *citiesSmall* dataset for $k = 1$, $k = 2$, and $k = 3$.
3. Generate a plot with `utils.plot_classifier` on the *citiesSmall* dataset (plotting the training points) for $k = 1$, using your implementation of kNN. Include the plot here. To see if your implementation makes sense, you might want to check against the plot using `sklearn.neighbors.KNeighborsClassifier`. Remember that the assignment 1 code had examples of plotting with this function and saving the result, if that would be helpful.
4. If we entered the coordinates of Vancouver into the predict function, would it be predicted to be in a blue state or a red state (for $k = 1$)?
5. Why is the training error 0 for $k = 1$?
6. If you didn't have an explicit test set, how would you choose k?

Hint: when writing a function, it is typically a good practice to write one step of the code at a time and check if the code matches the output you expect. You can then proceed to the next step and at each step test is if the function behaves as you expect. You can also use a set of inputs where you know what the output should be in order to help you find any bugs. These are standard programming practices: it is not the job of the TAs or the instructor to find the location of a bug in a long program you've written without verifying the individual parts on their own.

2.2 Picking k in KNN

The file `data/ccdebt.pkl` contains a subset of Statistics Canada's 2019 Survey of Financial Security; we're predicting whether a family regularly carries credit card debt, based on a bunch of demographic and financial information about them. (You might imagine social science researchers wanting to do something like this if they don't have debt information available – or various companies wanting to do it for less altruistic reasons.) If you're curious what the features are, you can look at the `'feat_descs'` entry in the dataset dictionary.

Anyway, now that we have our kNN algorithm working,[1] let's try choosing k on this data!

1. Remember the bedrock principle: we don't want to look at the test data when we're picking k. Inside the `q2_2()` function of `main.py`, implement 10-fold cross-validation, evaluating on the `ks` set there ($k = \{1, 5, 9, \ldots, 29\}$), and store the *mean* accuracy across folds for each k into a variable named `cv_accs`.

   Specifically, make sure you test on the first 10% of the data after training on the remaining 90%, then test on 10% to 20% and train on the remainder, etc – don't shuffle (so your results are consistent with ours; the data is already in random order). Implement this yourself, don't use scikit-learn or any other existing implementation of splitting. There are lots of ways you could do this, but one reasonably convenient way is to create a numpy "mask" array, maybe using `np.ones(n, dtype=bool)` for an all-`True` array of length `n`, and then setting the relevant entries to `False`. It also might be helpful to know that `~ary` flips a boolean array (`True` to `False` and vice-versa).

   Submit this code, following the general submission instructions to include your code in your results file.

2. The point of cross-validation is to get a sense of what the test error for a particular value of k would be. Implement a loop to compute the test accuracy for each value of k above. Submit a plot of the cross-validation and test accuracies as a function of k. Make sure your plot has axis labels and a legend.

3. Which k would cross-validation choose in this case? Which k has the best test accuracy? Would the cross-validation k do okay (qualitatively) in terms of test accuracy?

4. Separately, submit a plot of the training error as a function of k. How would the k with best training error do in terms of test error, qualitatively?

[1] If you haven't finished the code for question 2.1, or if you'd just prefer a slightly faster implementation, you can use scikit-learn's `KNeighborsClassifier` instead. The `fit` and `predict` methods are the same; the only difference for our purposes is that `KNN(k=3)` becomes `KNeighborsClassifier(n_neighbors=3)`.

3 Random Forests

3.1 Implementation

The file `vowels.pkl` contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a `RandomStump` class that differs from `DecisionStumpInfoGain` in that it only considers $\lfloor \sqrt{d} \rfloor$ randomly-chosen features.[2] You are also provided with a `RandomTree` class that is exactly the same as `DecisionTree` except that it uses `RandomStump` instead of `DecisionStump` and it takes a bootstrap sample of the data before fitting. In other words, `RandomTree` is the entity we discussed in class, which makes up a random forest.

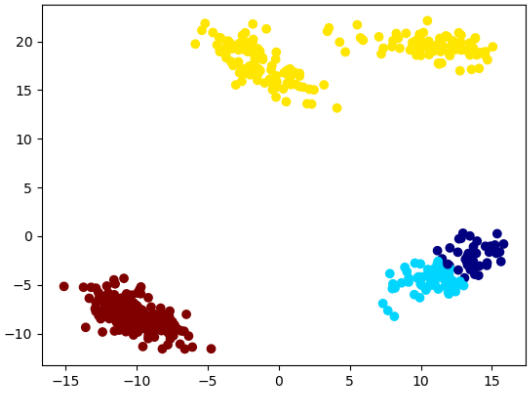
If you run `python main.py 3` it will fit a deep `DecisionTree` using the information gain splitting criterion. You will notice that the model overfits badly.

1. Using the provided code, evaluate the `RandomTree` model of unlimited depth. Why doesn't the random tree model have a training error of 0?
2. For `RandomTree`, if you set the `max_depth` value to `np.inf`, why do the training functions terminate instead of making an infinite number of splitting rules?
3. Complete the `RandomForest` class in `random_tree.py`. This class takes in hyperparameters `num_trees` and `max_depth` and fits `num_trees` random trees each with maximum depth `max_depth`. For prediction, have all trees predict and then take the mode. Submit this code.
4. Using 50 trees, and a max depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss.
5. Why does a random forest typically have a training error of 0, even though random trees typically have a training error greater than 0?

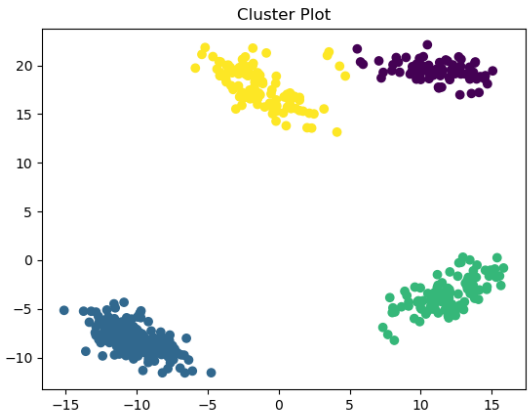
[2] The notation $\lfloor x \rfloor$ means the "floor" of x, or "x rounded down". You can compute this with `np.floor(x)` or `math.floor(x)`.

4 K-Means Clustering

If you run `python main.py 4`, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:



(Note that the colours are arbitrary due to the label switching problem.) But the 'correct' clustering (that was used to make the data) is something more like this:



4.1 Selecting among k-means Initializations

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples x_i and their means w_{y_i},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} (x_{ij} - w_{y_i j})^2.$$

where y_i is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the w_c and the y_i values.

1. In the `kmeans.py` file, complete the `error()` method. `error()` takes as input the data used in fit (`X`), the indices of each examples' nearest mean (`y`), and the current value of means (`means`). It returns the value of this above objective function. Hand in your code.
2. Instead of printing the number of labels that change on each iteration, what trend do you observe if you print the value of this error after each iteration of the k-means algorithm?
3. Run k-means 50 times (with $k = 4$) and take the one with the lowest error. Report the lowest error obtained. Visualize the clustering obtained by this model, and submit your plot.

4.2 Selecting k in k-means

We now turn to the task of choosing the number of clusters k.

1. Explain why the error function should not be used to choose k.
2. Explain why even evaluating the error function on test data still wouldn't be a suitable approach to choosing k.
3. Hand in a plot of the minimum error found across 50 random initializations, as you vary k from 1 to 10.
4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers.

4.3 k-Medians

The data in *clusterData2.pkl* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

1. Visualize the clustering obtained by running k-means 50 times (with $k = 4$) on *clusterData2.pkl* and taking the one with the lowest error. Are you satisfied with the result?
2. Hand in the elbow plot for this data. What values of k might be chosen by the elbow method for this dataset?
3. Instead of the squared distances between the examples x_i and their cluster centers, consider measuring the distance to the cluster centers in the L1-norm,

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij} - w_{y_i j}|.$$

   Hand in the elbow plot for k-means when we measure the error of the final model with the L1-norm. What value of k would be chosen by the elbow method?
4. The k-means algorithm tries to minimize the squared error and not the L1-norm error, so in the last question there is a mis-match between what the learning algorithm tries to minimize and how we measure the final error. We can try to directly minimize the L1-norm with the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and to update the w_c by setting them to the "median" of the points assigned to the cluster (we define the d-dimensional median as the concatenation of the median of the points along each dimension). Implement the k-medians algorithm, and hand in your code and the the plot obtained by minimizing the L1-norm error across 50 random initializations of k-medians with $k = 4$.

5 Very-Short Answer Questions

Write a short one or two sentence answer to each of the questions below. Make sure your answer is clear and concise.

1. What is a feature transformation that you might do to address a "coupon collecting" problem in your data?
2. What is one reason we would want to look at scatterplots of the data before doing supervised learning?
3. When we fit decision stumps, why do we only consider $>$ (for example) and not consider $<$ or $\geq$?
4. What is a reason that the data may not be IID in the email spam filtering example from lecture?
5. What is the difference between a validation set and a test set?
6. Why can't we (typically) use the training error to select a hyper-parameter?
7. If you can fit one model in 10 sec., how long (in days) does it take to find the best among a set of 16 hyperparameter values using leave-one-out cross-validation on a dataset containing $n = 4320$ examples?
8. Naïve Bayes makes the assumption that all features are conditionally independent given the class label. Why is this assumption necessary and what would happen without it?
9. Why is KNN considered a non-parametric method and what are two undesirable consequences of KNNs non-parametric design?
10. For any parametric model, how does increasing number of training examples n affect the two parts of the fundamental trade-off.
11. Suppose we want to classify whether segments of raw audio represent words or not. What is an easy way to make our classifier invariant to small translations of the raw audio?
12. Both supervised learning and clustering models take in an input x_i and produce a label y_i. What is the key difference?
13. In k-means clustering the clusters are guaranteed to be convex regions. Are the areas that are given the same label by KNN also convex?