CS158 - Assignment 4
k-NN and Feature Preprocessing

Due: Sunday, September 23 by 11:59pm

https://www.smbc-comics.com/comic/2008-06-08

For this assignment, we're going to add a third classifier to our arsenal (k-NN) and also implement a few feature preprocessing techniques so that we can start to play with some more interesting data sets, in particular, data sets that use real/continuous features!

As always, make sure to read through the entire handout before starting. You may (and I would strongly encourage you to) work with a partner on this assignment. If you do, you must both be there whenever you are working on the project. If you'd like a partner for the lab, e-mail me asap and I can try and pair people up.

High-level Requirements

For this assignment, you are going to be implementing three main things:

- the k-nearest neighbor (k-NN) classifier
- feature normalization (mean centering and variance scaling)

- example length normalization

Like the previous assignments, we're also going to then run a few experiments on these and, now that we can do statistical tests, we can be more confident with our comparisons :)

Starter

We'll be using Github Classroom to generate our repo. You can join by going to:

`https://classroom.github.com/a/ohcaxW8i`

Remember that if you want to work in a team (good idea!) the first person should create the team name and the second person should then select that team.

Code

The starter code includes some new classes/interfaces and some changes to some previous ones:

- Added `PerceptronClassifier` and `AveragePerceptronClassifier` classes. These will **NOT** be included in the starter code if you try and download it before Tuesday since people are still working on the last assignment. On Tuesday, I will update the starter code to include these.

  You are welcome to use your own if you'd like, but I'm providing these for those of you who weren't that comfortable with your versions :)

- I have created a new package `ml.data` and moved all of the data processing classes in there.

- Added `DataPreprocessor` interface. This defines the interface for any of our data preprocessing approaches.

- Added `CrossValidationSet` class. This class is generated from a `DataSet` by calling `getCrossValidationSet`. The class represents an n-fold splitting of the data, as talked about in class. By asking for each split, you can calculate cross validation scores.

- Added `DataSetSplit` to represent a train/test split of the data. This replaces the old (read, bad) way of doing it where we (I) represented a train/test split by an array of `DataSet`.

- `DataSet` has been updated to include this new split behavior, the new cross validation functionality, a new constructor, `addData` methods for adding additional data to a data set (useful with new constructor), and a few other minor changes.

- `Example` class has been updated to include a `setFeature` function that will allow you to update the value of a feature.

To make life a little easier (and to show you how useful docstrings are) I've generated the html documentation from the starter code and made it available at:

`http://www.cs.pomona.edu/classes/cs158/assignments/assign4/doc/`

Data

I have included one new data set: `titanic-train.real.csv`. This is almost the real data set made available on kaggle. This data set includes the same examples as our current dataset, but includes two additional features and there are now non-binary features.

k-NN Classifier

To round out our classification approaches, implement the k-NN classifier. In particular:

- Implement a class called `KNNClassifier` that implements the `Classifier` interface. The `train` method should be very easy. For the distance measure, use euclidean distance.
- The class should have a zero parameter constructor and by default k should be 3.
- The class should include a `setK` method that allows you to update the k value.

Data Pre-processing

So far, we've only been playing with boolean features. Generally, as long as there aren't too many of them, boolean features don't require a lot of pre-processing. Right now, the perceptron and k-NN classifiers will both work on real-valued features, however, if you run them on the real-valued data set, you'll notice that their performance is pretty bad. The problem is that we need to normalize the feature values.

Implement the following as new classes in the `ml.data` package:

- A class called `ExampleNormalizer` that implements the `DataPreprocessor` interface. This class should normalize each example so that its *length* is 1.
- A class called `FeatureNormalizer` that implements the `DataPreprocessor` interface. For each feature (e.g., age) this class should:
  - recenter all of the feature values for that one feature so that the mean is 0 and then
  - rescale all of the features values for that one feature so that the standard deviation is 1

  In the notes, this is referred to as "centering" and "variance scaling".

Experiments

Answer the following questions and put them in a file called `experiments` (pick some reasonable file type). Put your name(s) at the top of the file and explicitly label each answer with the question number.

1. Calculate the performance of the perceptron classifier on the 10-fold cross validation of the data (i.e. you should have 10 numbers) with the `AveragePerceptronClassifier` on the *old* binary data, i.e. "titanic-train.csv". Use a reasonable number of iterations based on your experience from last assignment or from a small experiment.

   Also include the average of the 10 folds.

   Because the perceptron algorithm involves randomness (i.e. because it shuffles the examples each round), to do this properly:

   - Generate a 10-fold cross validation. Only do this once for this experiment (i.e. don't keep repeatedly creating new 10-fold cross validations).
   - On each of the splits of the data, run the perceptron 100 times and average those results to get a single value for that split.
   - Repeat this for each of the 10 splits.

   For any of the experiments below for the perceptron classifiers, make sure to follow this procedure to get consistent results.

2. Calculate the accuracy on the 10 folds on the *new* non-binary data, i.e. "titanic-train.real.csv". You should notice a pretty big difference here. Why do you think there is such a big difference? (You don't have to write your answer.)

3. Repeat experiments 1 and 2 for your new k-NN classifier.

4. Now, generate a table of scores with the individual 10-fold scores and the 10-fold average on the following algorithm variants:

   - k-NN with length normalization
   - k-NN with feature normalization
   - k-NN with length and feature normalization
   - perceptron with length normalization
   - perceptron with feature normalization
   - perceptron with length and feature normalization

   This should be a table with 66 numbers!

5. Pick a few (say 4-5) of these results (including the earlier results) and calculate their t-test score to figure out if the differences are significant. Pick a couple of the experimental results that are close and a couple where they're further apart.

I'd suggest just using Excel/Google Docs/Open Office to calculate these, though you can use whatever you'd like. If you use these the `ttest` function is what you want to use. The first two parameters are the two data sets, the third parameter (tails) should be 2 , and the fourth parameter (type) should be 1 (paired t-test).

List the comparisons that you made and their t-test p values.

6. Write a short (3-4 sentence) paragraph summarizing your results.

Hints/Advice

http://www.xkcd.com/1263/

- For the k-NN classifier the hardest part is not calculating the distance, but finding the k closest. The challenge is keeping track of both the distance to an example AND the example itself so that when you sort the data based on distance, you can recover the examples too. My suggestion is to make a helper class that stores an `Example` along with the distance from that example to the example you're trying to classify. This class should implement the `Comparable` interface. The comparison can then just compare the distances. This will allow you to use built-in sorting or even built in priority queues to pick the k closest. There are other ways of doing this, of course.

- For the two feature pre-processing techniques think about what information, if any, you need to store from train preprocessing to use during test preprocessing. For feature normalization, all examples (train and test) should be normalized *using the same mean and standard deviation.*

- For the feature normalization, make sure to do recentering before you do variance scaling.

- For feature normalization you don't need to create a new data set or new examples. Mutate (i.e. change) the existing examples using the `setFeature` method.

- The `CrossValidationSet.getValidationSet` method has a second parameter, `copyData`, which indicates whether the `DataSetSplit` returned shares the underlying `Example`s with the original data or is a new copy. If you are not mutating the data during evaluation, then

you can just set `copyData` to `false`. However, if you are mutating the `Examples` (e.g. feature preprocessing) then my advice is to set `copyData` to `true` and get a fresh copy each time. For these feature preprocessing approaches it generally won't change the results either way, but can be a bit confusing if you're trying to debug.

Extra Credit

For those who would like to experiment (and push themselves) a bit more (and of course, get a bit of extra credit) you can try out some of these extra credit options. If you try out these options, include an extra file called `extra.txt` that describes what extra credit you did.

- The easiest way to implement k-NN is to calculate all of the distances, sort them and then take the k smallest. A better way is to only keep track of the k smallest as you go with something like a priority queue. This saves on memory and in many situations also the runtime. Implement your k-NN classifier this way. What is the big-O runtime for sorting vs. priority queue as implemented above?
- Add another feature normalizer. One option would be to do max value scaling, rather than variance scaling. If you go this route, also include some experimentation.

When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.). Your k-NN classe should follow the package structure and be inside the `ml.classifiers` package/directory.

Your `experiments` file with the answers/graphs from Section should also be added to your repo.

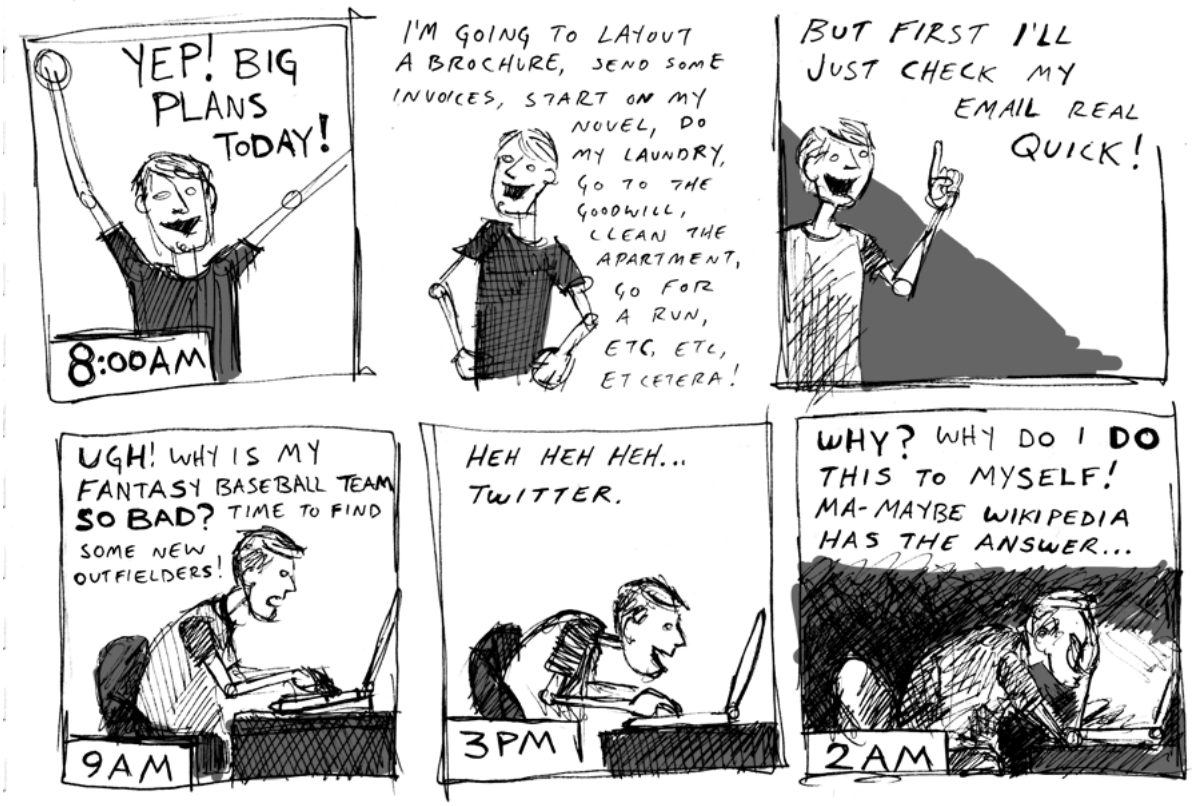
Submit your assignment on gradescope by providing the url for your github repo. **Make sure that you have pushed the latest version of your code, etc. to the repo before you submit.** Note that you may submit as many times as you want up until the assignment is due.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate doctsring
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.



http://www.entrepreneursolo.com/entrepreneur-thoughts/more-on-procrastination-is-there-an-app-for-that