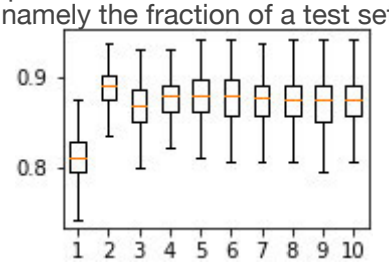
Assignment 1

This assignment involves trying out a variety of classification algorithms and at least one clustering algorithm. It requires use of `python`, `numpy`, `matplotlib`, and `sklearn`, so it also serves as an introduction to all those tools. You can run python any way you like - one option is to do the development in a `jupyter notebook` but there are others.
You should produce and submit **EITHER**
a text report as a `.pdf` file, including figures and tables as needed, PLUS code as a separate file, **OR**
a single jupyter notebook, as a `.ipynb` file
Whichever one you choose, submit it using the usual submission system. If you choose the notebook, make sure it has already run (the marker won't want to run it and wait to see what it produces!), and remember to use markdown cells to write text, as per Tutorial 1, not comments in code cells.
And if your code ends up very verbose, consider the separate text report option above, to make it more readable.

Part one: classifiers

This part is inspired by plot_classifier_comparison which uses `sklearn` to build a 3-by-10 table of Classifiers *versus* Data. In that case, the end result of each test was a single 2-dimensional plot, but in this assignment, the goal for each such combination is instead a box plot. The x-axis gives options for a parameter of the model, while the y-axis indicates the spread for a simple performance measure of the algorithm, namely the fraction of a test set that it gets correct.



Set up

Start by downloading the following datasets (e.g. as csv files)
steel-plates-fault
ionosphere
banknotes
you will also be testing out models using "fake" data you make yourself, for which you will need to import `make_classification` from `sklearn.datasets` . The following code makes appropriate "fake data":

```
# making a fake data set here.
X, y = make_classification(n_features=20, n_redundant=0, n_informative=5,
n_clusters_per_class=1)
X += 4.0 * numpy.random.uniform(size=X.shape)
myfakedataset = (X,y)
```

Use that to make your fake data. Notice how it assigns labels and then moves the `X` around randomly, in order to add "noise" to the data.
Preprocess the X data first, using `X = StandardScaler().fit_transform(X)` .
You will be trying out the following models, in `sklearn` :
`KNeighborsClassifier` (k nearest neighbours)
`GaussianNB` (the Gaussian form of Naive Bayes)
`DecisionTreeClassifier` (a decision tree (DT))
`LogisticRegression` (essentially, a perceptron)
`GradientBoostingClassifier` (Gradient Boosted DTs)
`RandomForestClassifier` (random forest, ensemble of DTs)
`MLPClassifier` (neural net)
You are to try out these classifiers on the above datasets, using their default settings unless otherwise specified. The assignment is to investigate one complexity control parameter for each classifier, by setting it to a range of plausible values and seeing how well it does on "held out" data. To do this you will need `train_test_split` from `sklearn` .
If we were to do this (split the dataset into train and test) only once, we would only get a single estimate of performance. To get better estimates, simply repeat at least 250 times with different random splits. For simplicity, use a 50:50 train:test split in all cases. For each setting of the control parameter, we then have a distribution over 250 or so values to convey. A nice way to do this is to produce what is called a box plot (or box and whiskers plot): in `sklearn` it is `boxplot`.

Produce the plots

Although it won't fit into one summary figure, you're essentially building a big table like the plot_classifier_comparison one. However you should structure it as separate main plots (one per model/algorithm), each consisting of several subplots (the different datasets).
In each case, the aim is adjust a parameter that has some effect on the "complexity" or modelling power of the Classifier. In some cases there are multiple options here, so for consistency across the class the table below gives the one you should try, and a sensible range of values for it.

Classifier	controller	values
KNeighborsClassifier	"n_neighbors"	1 to 5
GaussianNB	"var_smoothing"	[1e-9,1e-5,1e-1])
LogisticRegression	"C"	[.1,.5,1.0,2.0, 5.0])
DecisionTreeClassifier	"max_depth"	1 to 10
GradientBoostingClassifier	"max_depth"	1 to 10
RandomForestClassifier	"max_depth"	1 to 10
MLPClassifier	"alpha"	[1e-5, 1e-3,0.1, 10.0])

note: the `GaussianNB` case is a bit weird: `var_smoothing` is not a very obvious control parameter. But for completeness, use it anyway.

Additional commentary following the plots

Include two summary tables, with rows being models, and columns being datasets:
Table I is to contain the best average value (of validation error).
Table II is to contain the associated (ie. best) value for the control parameter.
Write a paragraph summarising the overall results, as captured in these two tables.
If you notice something unexpected, point it out, explaining why you think it is worth mentioning.
Part one is worth 70% of the total marks for this assignment. The breakdown is 50 for the main plots, 10 for the tables, and 10 for the discussion of them.

Part two: clustering, for semi-supervised learning

Worth 30% of the total marks for this assignment.
The real world often throws up situations in whic we have a mountain of *unlabelled* data but only relatively few examples with their labels. Ultimately this is because acquiring accurate labels is expensive, since it often requires human expertise, whereas acquiring unlabelled vectors is often automatic and cheap.
This part asks you to investigate whether it is possible to use features derived from unsupervised learning to enhance the predictions made by a supervised learner (in this case, a Classifier).
The idea of semi-supervised learning is that, by using the results of unsupervised learning, we might be able to improve those of supervised learning. Suppose we perform clustering on the total data available (including the "test" - this is okay provided we *don't use the labels* when clustering). We could include this cluster label (e.g. as an integer) in an additional column of `X`. Might a classifier trained on this augmented X actually do better (on the test data) than it would if trained only on the original `X`?
We might expect that semi-supervised learning should only improve performance in cases where there is some legitimate causal connection between the process that generates the X patterns and the process that assigns the Y classes. Your task is to see whether this is so, for a particular dataset.
To do the test, act as if almost all of the data is missing its label: you will use the iris data and split it so that 95 percent of it goes into a test set, and thus **only 5 percent** is available as a training set.
use `KMeans(n_clusters=3)` (ie. k-means with k=3) as the cluster method, run on the whole dataset, and add the cluster label as a new column in X. Then,
for at least a hundred repeats:
randomly split the iris dataset (as above)
train `GaussianNB()` Classifiers on the training data (nb. you are welcome to optionally try other clustering methods as well, but begin with this one)
without the cluster column, and
with the cluster column
find and store the test-set scores of (i) and (ii)
plot a scatterplot of the (i)-scores on x-axis against the (ii)-scores on the y-axis. So if the cluster label tends to help, there will be more points above the diagonal than below it.
Finally, repeat the analysis for the `banknotes` data in place of `iris`, and write a paragraph explaining whether they are the same or different, **and why** .
NOTE: additionally, for Students doing AIML421 only: (extra 20%)
see whether you can demonstrate the same effect, but via a *dimensionality reduction algorithm* instead of *clustering* . To do so, it is likely you will need to use a dataset with more dimensions than 'iris'.