School of Computing and Information Systems
The University of Melbourne
COMP30027 MACHINE LEARNING (Semester 1, 2019)

Practical exercises: Week 7

Today, we're going to do a "bake–off[1]" between **Logisitic Regression** and the classifiers which are its most obvious competitors: **Naive Bayes** (a simpler probabilistic approach) and **Support Vector Machines** (using a linear kernel).

Don't forget that you should refer back to earlier weeks or the `scikit-learn` API where necessary.

1. Let's begin with the simple *Iris* dataset. Recall that this dataset has four numerical attributes, three classes, and only a small number of instances.
   Using Zero-R as a **baseline**, and the following[2] classifiers in their default configurations (i.e. no **parameter tuning**):

   (i) `naive_bayes.MultinomialNB`

   (ii) `naive_bayes.GaussianNB`

   (iii) `svm.LinearSVC`

   (iv) `svm.SVC` (this should the only time you try this model today!)

   (v) `linear_model.LogisticRegression`

   Which has the best **accuracy**[3] when **cross–validating** on this dataset? (Note that the whole bake–off should only take at most a couple of seconds to run.) Why?

2. Download the *Abalone* dataset from the UCI ML repository (`https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data`). The class here is numeric (number of rings, which defines the age of the mollusc), but we can set up a two-class problem:

```
def convert_class(raw):
        if int(raw)<=10: return 0
        else: return 1

for line in f:
        atts = line[:-1].split(",")
        X.append(atts[1:-1])
        y.append(convert_class(atts[-1]))
```

   Don't forget to make `X` as a `numpy` array using `.astype(np.float)` — again, using Zero-R as a baseline, and the four classifiers above (ignoring `SVC()`):

   (a) Who wins this bake–off? Why might that be? (It should again take at most a few seconds to run.)

   (b) Modify `convert_class()` so that it instead sets up a three-class problem: 8 rings or less; 9 or 10 rings; 11 rings or more — which classifier wins now?

   (c) Set up the 29-class problem (!) as follows: `y.append(int(atts[-1]))` — now which classifier is the best? Naive Bayes seems like it would be woefully inadequate for a problem like this: how is it going? This run will probably take noticeably longer than the others — which classifier is the main culprit?

   (d) The gender attribute (`atts[0]`) is mostly un-helpful for this problem. Nevertheless, let's incorporate it into the models as a one–hot attribute and see what happens:

[1] `https://en.wiktionary.org/wiki/bake-off`
[2] Recall why we aren't trying `BernoulliNB` on this dataset. Try it out for yourself, if you aren't convinced.
[3] You can average over the partitions using `np.average()`

```
def convert_gender(raw):
        if raw=="M": return 0
        elif raw=="I": return 1
        elif raw=="F": return 2
        else: return 3

for line in f:
        atts = line[:-1].split(",")
        atts[0]=convert_gender(atts[0])
        X.append(atts[0:-1])
        y.append(convert_class(atts[-1]))

ohe = OneHotEncoder(categorical_features=[0])
X = ohe.fit_transform(X).toarray()
```

Pay particularly close attention to the performance on *Abalone-29*: which model(s) are most resilient to the addition of these three attributes? Why do you suppose this would be?

3. Let's look at something a little bigger: the US census data *Adult* [4]. We've uploaded a slightly pre-processed version of this dataset (`adult.txt`) to the LMS.
   Note that this dataset is substantially larger; we recommend saving time by doing **hold–out** instead of cross–validation — but note that the accuracies you observe will be more subject to **variance**[5].
   Many of the attributes are nominal; for now, let's ignore them (or you can convert to one-hot if you're feeling brave!). An inelegant way of loading the data is to smush the numeric attributes into a list, for example:

   ```
        X.append([atts[0],atts[2],atts[4],atts[10],atts[11],atts[12]])
   ```

   Don't forget to convert the class to integers! Which of the classifiers wins this bake–off? What might be different about this problem, compared to the previous ones?

4. A better way of loading in a mixed data set like this one is the `DictVectorizer()`, which converts an array of dictionaries into a sparse representation[6], for example:

   ```
   for line in f:
           atts = line[:-1].split(",")
           this = {}
           this["age"]=int(atts[0])
           this["workclass"]=atts[1]
           ...
           X.append(this)
   ```

   (Note that the numeric attributes shouldn't be left as strings!) and:

   ```
   vec = DictVectorizer()
   X = vec.fit_transform(X).toarray()
   ```

   (a) Any changes to the winner of the bake–off? Was it worthwhile adding all of these various extra attributes? (There are a lot!)

   (b) As a final summary, compare the results of the four classifiers (and the baseline) when cross–validating over this large dataset. (It will probably take a few minutes to run.) Confirm that the averaged accuracy for `LinearSVC` is consistent with what you expect, based on your observations across the folds (or when trying different hold–out partitions).

[4] `https://archive.ics.uci.edu/ml/datasets/Adult`

[5] Pay extra–close attention to `LinearSVC` here; try a few train–test splits until something weird happens. Why is this?

[6] This is especially nice when we have lots of 0–valued features, like in a typical problem in **Natural Language Processing**. The `DictVectorizer` also automatically makes categorical attributes one–hot, which is especially useful when we have lots of categorical attributes with lots of different values, like in this problem.