CSCI 390 Machine Learning – HW1

DUE: Friday, September 13 - 11:59PM

In this assignment you will be exploring the application of decision tree models to a data set of census data that I have provided. The data is in an Excel spreadsheet called "Census_Supplement.xlsx". The sheet has 100000 rows; the first row contains the variable names. Each row in the file corresponds to an individual person (there is no identifying information, of course). The relevant fields are:

AGI	Adjusted gross income
A_Age	Age
A_Sex	Sex; 1=Male, 2=Female
WKSWORK	Weeks worked
HAS_Div	Received dividend income; 1=Y, 0=N

The other fields can be ignored or dropped (for now).

The overall task is to develop a model to predict HAS_DIV from the other fields; we will do this in several ways. First, we will develop decision tree models that use only binary versions of the predictors and evaluate their performance. Next, we will use the continuous variables, and finally, we will experiment with a random forest of decision trees using the continuous variables.

Part 1 (40 points)

In part 1, you will gain experience with the details of the tree learning algorithm.

First, create three more variables: AGI_BIN, A_AGE_BIN and WKSWORK_BIN. Each of these is a binary that is 1 if the value is greater than the mean value of that variable, i.e.:

$$VAR_{BIN} = \begin{cases} 1, if\ AGI > \frac{1}{N}\sum_{N} AGI_N \\ 0, else \end{cases}$$

You can do this with Python, Excel (as I did) or however you wish.

Next, use the entropy-based information gain method to determine which of the binary predictors (AGI_BIN, A_AGE_BIN, WKSWORK_BIN and A_SEX) is the best to use for the top node in the decision tree to predict HAS_DIV. For each of these variables, compute BY HAND the information gain as described in the lecture and choose the variable with the largest information gain. Clearly label the overall entropy, the individual remainders and information gains, and the best feature to use first.

Part 2 (40 points)

In part 2, you will write code to develop a simple decision tree classifier using the four binary variables to predict HAS_DIV. You should use Python and scikit-learn – specifically the DecisionTreeClassifier in the "tree" module of sklearn. Be sure to follow the recommended steps

for implementing a model (no need for computed fields or variable selection this time). Use "criterion=entropy" for your decision tree. Use a 70/30 split of the data set into training and test portions. Train the model and test it in separate statements. Evaluate the performance of the model in two ways: print out the confusion matrix (see sklearn.metrics.confusion_matrix()) and the cross-validation scores (see sklearn.model_selection.cross_val_score()) on the test set.

Once you have your model trained, experiment with different maximum depths of the tree (the "max_depth" parameter to DecisionTreeClassifier()). How does the performance change for maximum depths of 3, 4, 5 and 10?

For max_depth=3, use the Python function supplied below to write your resulting classifier tree to a PNG image file, and include the tree in your report. Does the model choose the same top-level decision that you chose in part 1? Why or why not?

Finally, try the model with "criterion=gini", for the various model depths. Are there any changes? Are they significant?

Part 3 (40 points)

Now, create a decision tree classifier with the original, continuous versions of the variables that you binarized earlier (instead of the binary versions): AGI, A_AGE and WKSWORK. All other things being equal, how does the performance compare (choose a suitable value for the tree depth)? Is the same variable chosen as the top decision?

Once you have your model trained, experiment with different maximum depths of the tree (the "max_depth" parameter to DecisionTreeClassifier()). How does the performance change for maximum depths of 3, 4, 5 and 10?

For max_depth=3, use the Python function supplied below to write your resulting classifier tree to a PNG image file, and include the tree in your report.

Change the criterion to "gini" and compare the performance to the "gini" model using the binary variables.

Part 4 (30 points)

Finally, using the continuous variables, determine the best accuracy (on the test set) obtainable using a random forest classifier (sklearn.ensemble.RandomForestClassifier()). A random forest is a collection of decision trees trained on subsets of the training data, with the results aggregated. They are often highly accurate machine learning models for classification.

Experiment with parameters such as n_estimators, criterion, min_samples_leaf and bootstrap to see what gives the best performance. Document your efforts and select the best model.

Tips

- In sklearn, most classifiers and many other functions (train_test_split()) accept a random seed. If you pass a random seed, then any random behavior in the functions will behave the same each time; this is handy for debugging.

- The sklearn API documentation is at https://scikit-learn.org/stable/modules/classes.html.
- In the code I showed you in class, I used pandas to read from the Excel file; you should do the same. Pandas also has lots of other capabilities (writing to an Excel file, processing data in a DataFrame, etc.). You are welcome to use it. Documentation is at https://pandas.pydata.org/docs/.
- There is sample code everywhere on the Web for many of these functions; you are welcome to use it if you clearly note which lines you borrowed and clearly list the source URL.
- When working with larger datasets, I often find it helpful to create a small version of the dataset for testing my code (it loads and runs much faster). The modeling results are NOT useful, because I am not randomly sampling from the larger file, but I can use it to debug my program. In this case, I copied the first 1000 rows of the spreadsheet into a file called "Census_Supplement_1000.xlsx" and used that for code development.
- Here is code to write a decision tree classifier (clf) out to a file (at pathname); pass in the names of the features and classes as Python lists of strings:

```
import pydotplus
import collections

# call this function like this:
    writegraphtofile(clf, dataset.features, ("0", "1"), dataset.basedir+"conttree.png")

def writegraphtofile(clf, featurenames, classnames, pathname):
    dot_data = tree.export_graphviz(clf, out_file=None,
                                    feature_names=featurenames, impurity=True,
                                    class_names=classnames, filled=True,
                                    rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data)
    colors = ('lightblue', 'lightgreen')
    edges = collections.defaultdict(list)
    for edge in graph.get_edge_list():
        edges[edge.get_source()].append(int(edge.get_destination()))
    for edge in edges:
        edges[edge].sort()
        for i in range(2):
            dest = graph.get_node(str(edges[edge][i]))[0]
            dest.set_fillcolor(colors[i])
    graph.write_png(pathname)
```

Submission

Your submission will consist of a Word or pdf file, containing the items below, and your source code for all problems. Your Word file should contain:

- Your work for part 1 (if you want to write it out and scan it, insert the pics, but pay attention to file size!)
- The two .png image files for your trees for parts 2 and 3
- All output from your program for the tasks specified
- In the places where I have asked you to compare alternatives or to select the best, a brief explanation of what you did