CS434 — Implementation Assignment 3 — Due 11:59PM May 09, 2020

General instructions.

1. Please add onto the Python starter code (version 3.6+). Note: you may need to install several packages (pip install –user), but try not to require anything else. I'll run it with the packages already there, and you should have everything you need. (Things like 'os' are fine though if they're packaged with Python 3.6).

2. You can work in a team of up to 3 people. Each team will only need to submit one copy of the source code and report. You need to explicitly state each member's contribution in percentages (a rough estimate).

3. Your source code and report will be submitted through Canvas

4. You need to submit a readme file that contains the commands to run your code for requested experiments (e.g. python main.py {args}).

5. Please make sure that you can run code remotely on the class server (vm-cs434-1).

6. Be sure to answer all the questions in your report. You will be graded based on your code as well as the report. In particular, **the clarity and quality of the report will be worth 10 pts**. So please write your report in clear and concise manner. Clearly label your figures, legends, and tables.

7. In your report, **the results should always be accompanied by discussions of the results**. Do the results follow your expectation? Any surprises? What kind of explanation can you provide?

Decision Tree Ensemble for Predicting Election Results by US County Statistics
(total points: 90 pts + 10 report pts)

In this assignment we will work on a task to classify United States counties between Democratic and Republican majority vote in the 2016 US Election. We do this based on a set of discrete features related to each county. These features are a categorical representation of continuous features, which are provided in the data description (county dictionary) file. The goal in this assignment is to develop variations of the **decision tree** algorithm and ensemble methods.

Data. The data for this assignment is taken from a collection of US election data in the primary elections from 2016. The data has already been preprocessed for you. Here is a short description of each train and validation split:

(a) **Train Set Features/Labels (x_train.csv, y_train.csv):** Includes 2098 rows (samples, one for each county) in each file. Each sample in X contains 51 categorical features related to bins of continuous data encompassing various statistics of a given county. If you're curious, I can provide what the bins are, but it's not necessary for the assignment. Just think "quantiles" of some sort and you'll be fine understanding it.

(b) **Test Set Features/Labels (x_test.csv, y_test.csv):** Includes 700 rows. Each row obeys the same format given for the train set. This set will be used to see the performance of the models.

Important Guidelines. For all parts of this assignment:

(a) We assigned labels already for the class **1 to Republican and 0 to Democrat**. Be sure they aren't modified when making predictions, as the starter code assumes 0/1 labels, not -1/1 labels.

(b) Please do not add bias to the features.

(c) Please use the base classes provided for your implementations and modifications. You may add more functions if needed, but ensure the class structure preserves the format in the starter code.

(d) NOTE: This data is **class imbalanced**. That is, we have about a 2:1 ratio of positive to negative class. This will affect how we quantify predictions, as we may need to use more than just raw accuracy for model selection. You will be asked to explain your intuition behind why this is the case in the assignment.

Definitions

(a) Precision: $\frac{\mathbf{TP}}{\mathbf{TP+FP}}$, where TP = true positive predictions, and FP = false positive predictions. Interpretation: how well did we do recovering true positive predictions? That is, a low false positive rate (not many predictions for positive class that were incorrect predictions).

(b) Recall: $\frac{\mathbf{TP}}{\mathbf{TP+FN}}$, where TP = true positive predictions, and FN = false negative predictions. Interpretation: out of all the positive examples, how many did we find? That is, a low false negative rate (not many true positive classes got predicted as negative).

(c) F1 Score: $\mathbf{2} * \frac{\mathbf{precision*recall}}{\mathbf{precision+recall}}$, weighted average of precision and recall. Essentially, the balance between the two.

Part 1 (20 pts) : Decision Tree (DT). For this part we are interested in using a decision tree with below configuration:

- The DT uses Gini impurity to measure the uncertainty, rather than the entropy/information gain as presented on the slides. We do this to avoid taking logarithms, but the two serve a similar purpose. Specifically, if we have a node split from training list A to two left and right list AL and AR as depicted in figure 1 then

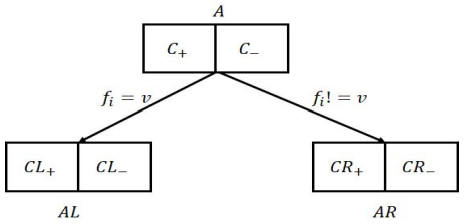

Figure 1: Split according to feature f_i testing against value v

the benefit of split for feature f_i against value v is computed as:

$$B = U(A) - p_l U(AL) - p_r U(AR) \tag{1}$$

Where U is the uncertainty measure. For this assignment, we will use gini-index, which is computed for a list such as AL as follows:

$$U(AL) = 1 - p_+^2 - p_-^2 = 1 - (\frac{CL_+}{CL_+ + CL_-})^2 - (\frac{CL_-}{CL_+ + CL_-})^2 \tag{2}$$

and p_l and p_r are the probabilities for each split which is given by

$$p_l = \frac{CL_+ + CL_-}{C_+ + C_-} \tag{3}$$

- The features are categorical with more than 2 possible values in most cases. So a feature might be tested multiple times against different values in the tree. Most of the logic for this is done for you, but there are a few things you will need to do to get the basic decision tree working.

Please implement the following steps:

(a) Please read through the starter code provided and work to understand how the recursive process is implemented for building a decision tree. I have chosen to use a fairly common, minimal approach for creating the trees. You will need to know this for modifying the classes for both Random Forest and AdaBoost.

(b) Add your code into the specified sections of the DecisionTreeClassifier class. You should only need to handle the Gini Impurity calculation (as shown above). Your function should calculate all the impurities and return the gain.

(c) The base function included in the **main.py** can be used to test your implementation. (Note: there is one included for Random Forest as well, feel free to make a similar one for AdaBoost when you implement it).

(d) Now, add a function in main for creating trees with depths ranging from 1 to 25 (inclusive). Plot and explain the behavior of train/testing performance against the depth. That is, plot accuracy versus number of trees, as well as F1 score vs number of trees. (F1 score is a weighted average of precision and recall, I believe you've covered precision/recall already).

(e) Report the depth that gives the best validation accuracy? You will probably hit a point where it will not need to be deeper, and it's best to use the simplest version.

(f) What is the most important feature for making a prediction? How can we find it? Report the name of it (see data dictionary) as well as the value it takes for the split.

Part 2 (30 pts) : Random Forest (Bagging). In this part we are interested in random forest which is a variation of bagging without some of its limitations. Please implement the following steps:

(a) Implement a random forest with parameters:
n_trees : The number of trees in the forest.
$max\_features$: The number of features for a tree.
max_depth : Maximum depth of the trees in the forest.

Here is how the forest is created: The random forest is a collection of n_trees trees. All the trees in the forest have maximum depth of max_depth. Each tree is built on a data set of size 2098 (size of your original training data) sampled (with replacement) from the training data. You sample both X and y together and need to preserve the indexes that correspond to (X, y) pairs. In the process of building a tree of the forest, each time we try to find the best feature f_i to split, we need to first sub-sample (without replacement) $max\_features$ number of features from the feature set and then pick f_i with highest benefit from $max\_features$ sampled features. Much of this is handled already in your DecisionTreeClassifier, but **you need to modify DecisionTreeClassifier class to handle feature bagging**. **You will also fill in missing RandomForestClassifier code in this class**.

(b) For $max\_depth = 7$, $max\_features = 11$ and $n\_trees \in [10, 20, ..., 200]$, plot the train and testing accuracy of the forest versus the number of trees in the forest n. Please also plot the train and testing F1 scores versus the number of trees.

(c) What effect does adding more trees into a forest have on the train/testing performance? Why?

(d) Repeat above experiments for $max\_depth = 7$, $n = 50$ trees, and $max\_features \in [1, 2, 5, 8, 10, 20, 25, 35, 50]$. How does $max\_features$ change the train/validation accuracy? Why?

(e) Optional: try to find the best combination of the three parameters using a similar idea.

(f) With your best result, run 10 trials with different random seeds (for the data/feature sampling you can use the same seed) and report the individual train/testing accuracy's and F1 scores, as well as the average train/testing accuracy and F1 scores across the 10 trials. Overall, how do you think randomness affects the performance?

Part 3 (30 pts) : AdaBoost (Boosting). For this part we are interested in applying AdaBoost to create yet another ensemble model with decision trees. Considering the AdaBoost algorithm described in the slides, please do following steps:

(a) Modify your **train_y and test_y** vectors (class labels) to **set class 0 to be class -1**. That is, change all the 0's in your labels to be -1 instead. AdaBoost assumed 1 and -1, rather than 1 and 0 for labels. Use:

```
y_train[y_train==0] = -1
y_test[y_test==0] = -1
```

(b) Implement your own AdaBoostClassifier class **using a modified DecisionTreeClassifier class for the weak learners** provided in the code. You should implement it in the same format as the other two classes (DecisionTreeClassifier and RandomForestClassifier). Once again, **you will need to slightly modify your DecisionTreeClassifier class**. Make a new class for DecisionTreeAdaBoost if you need to (since you'll have to add the weight vector as described below). However, **you need to preserve**

the original code as much as possible and simply modify the gain and prediction functions to handle the weights described. This helps make grading easier.

(c) Let the weak learner be a decision tree with **depth of only 1** (decision stump). The decision tree should get a weight parameter D which is a vector of size 2098 (training data size). Implement the decision tree with parameter D such that it considers D in its functionality.
(Hint: It changes the probability estimation used for computing gini-impurity and also for the predictions at leaves, use majority "weights" instead of majority counts. Basically, your weight vector D dictates your predictions, rather than "majority class").

(d) Using the decision tree with parameter D implemented above, develop the AdaBoost algorithm as described in the slide (and part a) with parameter L for the class (number of base classifiers/number of trees).

(e) Report the train and validation accuracy for $L \in [10, 20, ..., 200]$.

(f) Plot the train/test accuracy and train/test F1 scores of AdaBoost against the parameter L.

Part 4 (10 pts): Class Imbalance Questions Here, I hope you've noticed that with a 2:1 ratio of positive to negative class (approximately 66% of the data is class 1), accuracy might not be the best thing to report. Basically, we're used to seeing data with 50% for each class, and we know we're doing well if we get better than "random guessing", which we can see immediately with accuracy. In this class imbalanced case, we could predict the positive class every time and have a base accuracy of approximately 66%, which might trick us into thinking our model is doing okay. We've done nothing to handle this imbalance while training, but there are many things that are often done, and there's not a "right" answer for every case. What you need to do here is the following:

(a) Read this article: `https://www.svds.com/learning-imbalanced-classes/`

(b) **Write a summary of ideas you have for handling the imbalanced classes in our data.** There's no "right answer" here, but you should think about things we could use. I've added F1 score as a metric, which is coarse, but better than just accuracy. What else could I have done to see how good the classifier is on the testing data? I think this is important because sometimes (very, very often) in the real world we see much worse class imbalance than this, and we don't want to blindly report metrics like accuracy if we're reporting the wrong ones. Feel free to report ideas from the article, and if you're not familiar with precision/recall, refresh on it via Wikipedia or slides.

Submission. Your submission should include the following:
1) The modified source code with a short instruction on how to run the code for your experiments in a readme.txt.
2) Your report (preferably in **PDF format**), which begins with a general introduction section, followed by one section for each part of the assignment.
3) Please note that all the files should be in one folder and compressed only by **.zip.**