CS289a - HW5

John Semerdjian

November 10, 2015

Decision Trees for Classification

1. You must implement Decision Trees and Random Forests, then train each on the spam data, and use each to separately classify the spam data. Repeat for the census income classification. See the report below.

2. You are not allowed to use any off the shelf decision tree/random forest implementation for the homework. You can use external libraries for data preprocessing (in fact, we recommend it). You can use any programming language you wish to as long as we can read and run your code with minimal effort.

3. Code and report requirements:

> **3.a) For spam, if you use any other features or feature transformations, explain what you did in your report. You may choose to use something like bag-of- words. You should implement any custom feature extraction code in `featurize.py`, which will save your features to a `.mat` file.**
>
> **Answer:**
>
> No other additional features/transformations were used in my code.
>
> **3.b) A report of the results you obtained. List the performance of your decision tree and random forest on your validation set. Also list your single best Kaggle score with any method (state which method you used).**
>
> **Answer:**
>
> My best score on Kaggle was 75.7% accuracy using Decision Tree with a maximum tree depth of 10 and at least 5 observations per node. However, while tuning my models and submitting results that demonstrated improved cross-validation accuracy, I kept getting the same score on Kaggle. I only realized after I had exceeded the maximum number of allowable submissions that I was actually submitting an earlier version of my predictions rather than the newer, better versions. This was due to my a global variable that contained older results that was being converted into a CSV file for submission. Alas...
>
> My performance against a small hold out set using Decision Trees was 80.2%. I used a maximum tree depth of 12, and at least 5 observations per node.
>
> My performance against a small hold out set using Random Forests was 81.5%. My hyperparameters included a total of 20, maximum depth of 10, at least 5 observations per node, and 50 trees.
>
> For both Decision Trees and Random Forests, increasing the tree depth from 10 to 20 did not significantly improve results.

```
In [1]: import numpy as np
        import pandas as pd
        from DecisionTree import *
        from RandomForest import *
        from scipy.io import loadmat
        from sklearn.feature_extraction import DictVectorizer
```

```
In [2]: spam = loadmat('./spam-dataset/spam_data.mat')
        spam_test = spam['test_data']
        spam_train = spam['training_data']
        spam_train_n = spam_train.shape[0]
        spam_train_label = spam['training_labels'][0]
        first_obs = spam_train[0,:].reshape(1,32)

        # # shuffle data
        np.random.seed(289)
        spam_train_combined = np.hstack((spam_train, spam_train_label.reshape(spam_train_n,1)))
        np.random.shuffle(spam_train_combined)

        spam_train_label = spam_train_combined[:,32]
        spam_train = spam_train_combined[:,:32]
```

Spam Decision Tree

```
In [3]: dt_spam = DecisionTree(max_depth=12, min_obs=5)
        dt_spam.fit(spam_train[:4500,:], spam_train_label[:4500])
        dt_spam_pred = dt_spam.predict(spam_train[4500:,:])
        sum(dt_spam_pred == spam_train_label[4500:])/len(spam_train_label[4500:])
```

```
Out[3]: 0.80208333333333337
```

Spam Random Forests

```
In [4]: rf_spam = RandomForest(num_features=20, max_depth=10, min_obs=5, num_trees=50)
        rf_spam.fit(spam_train[:4500,:], spam_train_label[:4500])
        rf_spam_pred = rf_spam.predict(spam_train[4500:,:])
        sum(rf_spam_pred == spam_train_label[4500:])/len(spam_train_label[4500:])
```

```
Out[4]: 0.81547619047619047
```

```
In [ ]: # kaggle_spam = np.vstack((np.arange(1,len(rf_spam_pred)+1), rf_spam_pred)).T
        # np.savetxt(fname="./spam_prediction_rf_f20_md10_mo5.csv", X=kaggle_spam, fmt='%d',
        #            delimiter=',', header='Id,Category', comments='')
```

3.c) For your decision tree, and for a data point of your choosing, state what splits (i.e. which feature and which value of that feature to split on) that your decision tree made to classify it. An example of what this might look like:

i. ("viagra") >= 2

ii. ("thanks") < 1

iii. ("nigeria") >= 3

Answer:

For the first observation in the training dataset:

```
In [8]: first_obs

Out[8]: array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 2.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  1.,  0.,  0.,  0.]])
```

the splits were as follows:

```
! (feature 28) > 0.0
pain (feature 0) <= 0.0
& (feature 31) <= 0.0
money (feature 3) <= 0.0
$ (feature 26) <= 1.0
( (feature 29) <= 0.0
message (feature 15) <= 0.0
# (feature 27) <= 1.0
business (feature 14) <= 1.0
other (feature 12) <= 1.0
! (feature 28) <= 2.0
$ (feature 26) <= 0.0
```

3.d) For random forests, find and state the most common splits made at the root node of the trees. For example:

i. ("viagra") >= 3 (20 trees)

ii. ("thanks") < 4 (15 trees)

iii. ("nigeria") >= 1 (5 trees)

Answer:

For the first data point in the training set, the following splits were made:

```
pain (feature 0) <= 0.0 (11 trees)
featured (feature 9) > 0.0 (3 trees)
featured (feature 9) <= 0.0 (2 trees)
spam (feature 5) <= 0.0 (2 trees)
bank (feature 2) > 0.0 (2 trees)
bank (feature 2) <= 0.0 (2 trees)
path (feature 18) <= 0.0 (2 trees)
revision (feature 17) > 0.0 (2 trees)
volumes (feature 16) > 0.0 (2 trees)
volumes (feature 16) <= 0.0 (2 trees)
business (feature 14) > 0.0 (2 trees)
energy (feature 13) > 0.0 (2 trees)
private (feature 1) > 0.0 (2 trees)
height (feature 8) > 0.0 (1 tree)
creative (feature 7) <= 0.0 (1 tree)
spam (feature 5) > 0.0 (1 tree)
drug (feature 4) <= 0.0 (1 tree)
money (feature 3) <= 0.0 (1 tree)
meter (feature 19) > 0.0 (1 tree)
```

```
meter (feature 19) <= 0.0 (1 tree)
path (feature 18) > 0.0 (1 tree)
revision (feature 17) <= 0.0 (1 tree)
width (feature 11) > 0.0 (1 tree)
width (feature 11) <= 0.0 (1 tree)
differ (feature 10) > 0.0 (1 tree)
differ (feature 10) <= 0.0 (1 tree)
pain (feature 0) > 0.0 (1 tree)
```

4. (a), (b), (c), (d). Repeat the same parts for the census dataset. You will need to do your own feature processing. In part (a), you must report what you did to handle categorical variables and missing features. See the Appendix for detailed instructions.

My best score on Kaggle was 85.4% accuracy. I used a Decision Tree with maximum depth of 15 and at least 20 observations per node.

My performance against a small hold out set using Decision Trees (with the same hyperparameters as my Kaggle submisison) was 85.37%.

My performance against a small hold out set using Random Forests was 85.1%. My hyperparameters included a total of 50 features, maximum depth of 10, at least 15 observations per node, and 20 trees. I had previously used a 100 trees in my Random Forest but training time took too long.

I used the DictVectorizer function to perform binary one-hot coding of categorical data and treated the missing data as a separate categorical rather than performing imputation. The `fnlwgt` variable was also converted into a categorical variable from a numeric variable. I binned these values into one of 10 categories based on the range of values.

For both Decision Trees and Random Forests, increasing the tree depth from 10 to 20 did not significantly improve results.

```
In [5]: census_train = pd.read_csv('./census-dataset/data.csv')
        census_train_label = census_train.label
        census_train = census_train.drop('label', 1)
        census_test = pd.read_csv('./census-dataset/test_data.csv')

        # feature engineering: convert numeric to categorical
        fnlwgt_bins = np.linspace(0, 1500000, 10)
        census_train.fnlwgt = pd.cut(census_train.fnlwgt, fnlwgt_bins)
        census_test.fnlwgt = pd.cut(census_test.fnlwgt, fnlwgt_bins)

        # feature engineering: perform hot-one coding
        dv = DictVectorizer(sparse=False)
        census_dict_train = census_train.T.to_dict().values()
        census_dict_test = census_test.T.to_dict().values()
        census_train = dv.fit_transform(census_dict_train)
        census_test = dv.transform(census_dict_test)

In [6]: dt = DecisionTree(max_depth=15, min_obs=20)
        dt.fit(census_train[:25000,:], census_train_label[:25000])
        dt_census_pred = dt.predict(census_train[25000:,:])
        sum(dt_census_pred == census_train_label[25000:])/len(census_train_label[25000:])

Out[6]: 0.85370274469186946
```

```
In [7]: rf = RandomForest(num_features=50, max_depth=10, min_obs=15, num_trees=20)
        rf.fit(census_train[:25000,:], census_train_label[:25000])
        rf_census_pred = rf.predict(census_train[25000:,:])
        sum(rf_census_pred == census_train_label[25000:])/len(census_train_label[25000:])

Out[7]: 0.85111341273951324

In [ ]: # kaggle_census = np.vstack((np.arange(1,len(rf_census_pred)+1), rf_census_pred)).T
        # np.savetxt(fname="./census_predictions_tree_md15_mo20.csv", X=kaggle_census, fmt='%d',
        #            delimiter=',', header='Id,Category', comments='')
```

5.a) An explanation of the decision tree techniques you implemented (stopping criteria, pruning, dealing with missing attributes, splitting criteria other than entropy, heuristics for faster training, complex decisions at nodes, cross-validation, Adaboost, bagging etc.).

Answer:

I used several techniques for my decision trees. I used a maximum depth and minimum number of observations per node as stopping criteria. I did not do any pruning. I resolved issues involving missing attributes by using the DictVectorizer to automatically convert them into categorical features rather than perform imputation. I also converted the numerical feature `fnlwght` into a categorical feature by binning the values into 10 categories.

5.b) An explanation of the random forest techniques you used. If the decision trees you used inside your random forest were dierent than your standalone decision tree implementation, explain how.

Answer:

In addition to the criteria I used for decision trees, I specified the number of features and trees to use for training. I bootstraped data and also let each tree see only a subset of the features. After building the individual trees, I averaged their results for classification.