Finding Donors for CharityML

Definition

Project Overview :

In this project, I will employ several supervised algorithms of my choice to accurately model individuals' income using data collected from the 1994 U.S. Census. I will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. my goal with this implementation is to construct a model that accurately predicts whether an individual makes more than $50,000. This sort of task can arise in a non-profit setting, where organizations survive on donations. Understanding an individual's income can help a non-profit better understand how large of a donation to request, or whether or not they should reach out to begin with. While it can be difficult to determine an individual's general income bracket directly from public sources, I can (as we will see) infer this value from other publically available features. The dataset for this project originates from the UCI Machine Learning Repository. The datset was donated by Ron Kohavi and Barry Becker, after being published in the article "Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid".I can find the article by Ron Kohavi online. The data I investigate here consists of small changes to the original dataset, such as removing the 'fnlwgt' feature and records with missing or ill-formatted entries.

Implementation: Data Exploration

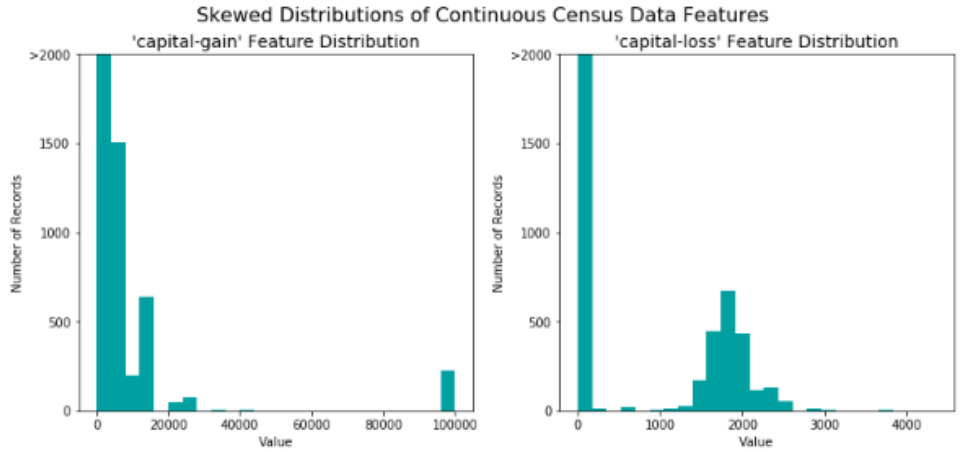
A cursory investigation of the dataset will determine how many individuals fit into either group, and will tell me about the percentage of these individuals making more than $50,000. In the code cell below, I will need to compute the following:

- The total number of records, 'n_records'
- The number of individuals making more than $50,000 annually, 'n_greater_50k'.
- The number of individuals making at most $50,000 annually, 'n_at_most_50k'.
- The percentage of individuals making more than $50,000 annually, 'greater_percent'.

```
n_records = data.shape[0]

# TODO: Number of records where individual's income is more than $50,000
n_greater_50k = np.sum(data['income']==">50K")
# TODO: Number of records where individual's income is at most $50,000
n_at_most_50k = n_at_most_50k = np.sum(data['income'] == '<=50K')


# TODO: Percentage of individuals whose income is more than $50,000
greater_percent = float(n_greater_50k) /float(n_records)

# Print the results
print("Total number of records: {}".format(n_records))
print("Individuals making more than $50,000: {}".format(n_greater_50k))
print("Individuals making at most $50,000: {}".format(n_at_most_50k))
print("Percentage of individuals making more than $50,000: {}%".format(greater_percent))

vs.distribution(data)
```

```
Total number of records: 45222
Individuals making more than $50,000: 11208
Individuals making at most $50,000: 34014
Percentage of individuals making more than $50,000: 0.2478439697492371%
```

```
C:\Users\lenovo\Untitled Folder 4\visulaised.py:105: UserWarning: Matplotlib is currently using module://ipykernel.pylab.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```



Featureset Exploration

- age: continuous.
- workclass:Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: Black, White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other.
- sex: Female, Male.
- capital-gain: continuous.
- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands

Normalizing Numerical Features

In addition to performing transformations on features that are highly skeId, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as 'capital-gain' or 'capital-loss' above); hoIver, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is

applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

Run the code cell below to normalize each numerical feature. I will use sklearn.preprocessing.MinMaxScaler for this.

Evaluating Model Performance

can use F-beta score as a metric that considers both precision and recall:

In particular, when , more emphasis is placed on precision. This is called the F score (or F-score for simplicity).

Algorithm and Techniques

Supervised Learning Models

The following are some of the supervised learning models that are currently available in scikit-learn that you may choose from:

- Gaussian Naive Bayes (GaussianNB)

- Decision Trees

- Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting)

- K-Nearest Neighbors (KNeighbors)

- Stochastic Gradient Descent Classifier (SGDC)

- Support Vector Machines (SVM)

- Logistic Regression

Random Forest

Random forest model can be applied in medical domain to identify a disease based on symptoms.

Example: detection of Alzheimer's disease.

Strengths - very good for large datasets, gives estimates of feature's importance, can be run in parallel to speed up training, reduces variance caused by decision trees by combining multiple decision trees.

Weaknesses - relatively high prediction time

Candidacy - random forest gives good performance when there are categorical variables. As there are around 45000 entries, random forest can train much better.

Gradient Boosting

Gradient Boosting can be applied in ranking algorithms, like ranking of searches by search engines.

Example: McRank: Learning to Rank Using Multiple Classification and Gradient Boosting.

Strengths - very good for large datasets, reduces bias and variance, combines multiple weak

predictors to a build strong predictor.

Weaknesses - relatively high training time, over-fitting if the data sample is too small.

Candidacy - the data we have is sufficiently large and clean so gradient boosting is suitable in this case.

Bootstrap Aggregation (or Bagging for short)

is a simple and very powerful ensemble method. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Example :Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset

Strengths - Bootstrap involves random sampling of small subset of data from the dataset. This subset can be replace. The selection of all the example in the dataset has equal probability. This method can help to better understand the mean and standand deviation from the dataset.

predictors to a build strong predictor.

Weaknesses - more complex the model gets

Candidacy- This algorithm train and selects the strong classifiers on subsets of data to reduce variance error

Implementing:

Creating a Training and Predicting Pipeline¶

To properly evaluate the performance of each model you've chosen, it's important that I create a training and predicting pipeline that allows me to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. my implementation here will be used in the following section. In the code block below, I will need to implement the following:

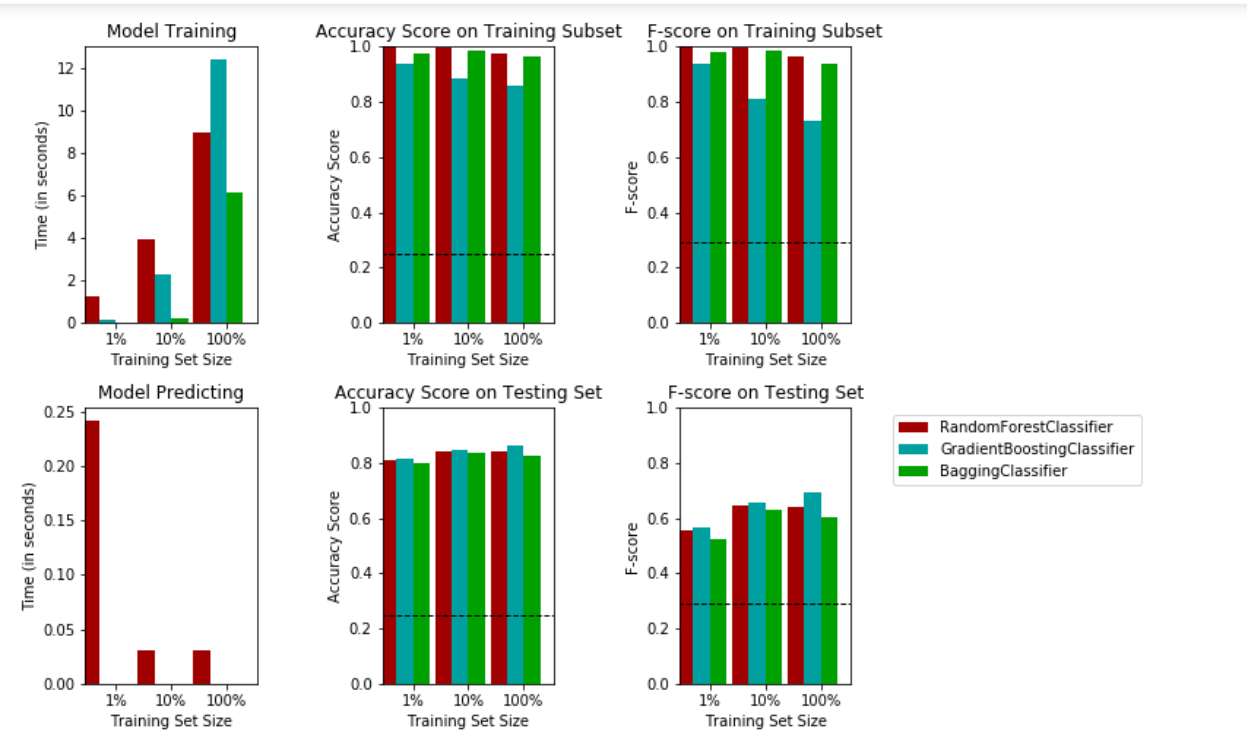
- Import fbeta_score and accuracy_score from sklearn.metrics.
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data X_test, and also on the training points
- Record the total prediction time.
- Calculate the accuracy score for both the training subset and testing set.
- Calculate the F-score for both the training subset and testing set.

Implementation: Initial Model Evaluation¶

In the code cell, I will need to implement the following: Import the three supervised learning models I've discussed in the previous section. Initialize the three models and store them in 'clf_A', 'clf_B', and 'clf_C'.Use a 'random_state' for each model I use, if provided.

Note: Use the default settings for each model — I will tune one specific model in a later section. Calculate the number of records equal to 1%, 10%, and 100% of the training data. Store those values in 'samples_1', 'samples_10', and 'samples_100' respectively. Note: Depending on which algorithms I chose, the following implementation may take some time to run!

Result



Improving Results¶

In this final section, I will choose from the three supervised learning models the best model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (X_train and y_train) by tuning at least one parameter to improve upon the untuned model's F-score.

Choosing the Best Model¶

Based on the evaluation I performed earlier, in one to two paragraphs, explain to CharityML which of the three models I believe to be most appropriate for the task of identifying individuals that make more than $50,000. HINT: Look at the graph at the bottom left from the cell above(the visualization created by vs.evaluate(results, accuracy, fscore)) and check the F score for the testing set when 100% of the training set is used. Which model has the highest score? Your answer should include discussion of the: metrics - F score on the testing when 100% of the training data is used, prediction/training time the algorithm's suitability for the data.

I choose to go with GradientBoostingClassifier() . This classifier can compensate for this after tuning as it will be able to discover more complex dependencies, this will however be done at the cost of more time for fitting.

Describing the Model in Layman's Terms

In one to two paragraphs, explain to CharityML, in layman's terms, how the final model chosen is supposed to work. Be sure that you are describing the major qualities of the model, such as how the model is trained and how the model makes a prediction. Avoid using advanced mathematical jargon, such as describing equations.

```
Unoptimized model
------
Accuracy score on testing data: 0.8630
F-score on testing data: 0.7395

Optimized Model
------
Final accuracy score on the testing data: 0.8712
Final F-score on the testing data: 0.7516
```

```
print(best_clf)
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=1000,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```