Machine Learning Engineer Nanodegree

Capstone Proposal

Abhishek Mishra
January 10th, 2019

Proposal

Domain Background

Many social programs have a hard time making sure the right people are given enough aid. It's especially tricky when a program focuses on the poorest segment of the population. The world's poorest typically can't provide the necessary income and expense records to prove that they qualify.

In Latin America, one popular method uses an algorithm to verify income qualification. It's called the Proxy Means Test (or PMT). With PMT, agencies use a model that considers a family's observable household attributes like the material of their walls and ceiling, or the assets found in the home to classify them and predict their level of need.

To improve on PMT, the IDB (the largest source of development financing for Latin America and the Caribbean) has turned to the Kaggle community. They believe that new methods beyond traditional econometrics, based on a dataset of Costa Rican household characteristics, might help improve PMT's performance.

Beyond Costa Rica, many countries face this same problem of inaccurately assessing social need. If we as Data scientists can generate an improvement, the new algorithm could be implemented in other countries around the world for social welfare.

Multiclass classification algorithms have been quite popular in academics and industry to solve real world problems and there are several papers published on this. We can refer below paper which depicts classification algorithms and results obtained on various data sets.

https://arxiv.org/ftp/arxiv/papers/1609/1609.00843.pdf

Problem Statement

The Inter-American Development Bank is asking for algorithmic solutions, based on a dataset of Costa Rican household characteristics, with income qualification for some of the world's poorest families. It hosted dataset and relevant details on Kaggle platform to get the best solutions for classification of incomes. Below is the link of Kaggle competition hosted by IDB.

https://www.kaggle.com/c/costa-rican-household-poverty-prediction

Here we have to classify income of families in 4 groups: 1(extreme poverty) ,2 (moderate poverty) ,3(vulnerable households) ,4(non-vulnerable households) based on household characteristics (dependent variables) hence this problem can be solved using supervised multi-class classifications algorithms.

Datasets and Inputs

Dataset for this problem can be accessed thorough Kaggle link:

https://www.kaggle.com/c/costa-rican-household-poverty-prediction/data

{train|test}.csv - the training set

- This is the main table, broken into two files for Train (with a Target column) and Test (without the Target column).
- One row represents one person in our data sample.
- Multiple people can be part of a single household. Only predictions for heads of household are scored.
- Total no of training records=9558
- Total no of features=142
- Total no of test records=23.9k
- Class distribution: Classes are not well balanced as 62.7% examples belong to category:4 and rest 31.3% examples belong to other 3 classes.

Target	No. of examples
1	755
2	1597
3	1209
4	5996

Core Data fields

- Id - a unique identifier for each row.
- Target - the target is an ordinal variable indicating groups of income levels.
  1 = extreme poverty
  2 = moderate poverty
  3 = vulnerable households
  4 = non-vulnerable households
- idhogar - this is a unique identifier for each household. This can be used to create household-wide features, etc. All rows in a given household will have a matching value for this identifier.
- parentesco1 - indicates if this person is the head of the household.
- This data contains 142 total columns.

As we don't have explicit test data with labels to validate and test our models' performance, we will split train set into train/validation and test sets so that we can choose optimal model based performance on validation and test sets.

Now to maintain similar class balance in train/validation/test set after split as in original trainset, we should use StratifiedShuffleSplit or StratifiedKFold.

StratifiedShuffleSplit: Provides train/test indices to split data in train test sets. This means that it splits your data into a train and test set. The stratified part means that **percentages will be maintained in this split**. So if 10% of your data is in class 1 and 90% is in class 2, this will ensure that 10% of your train set will be in class 1 and 90% will be in class 2. Same for the test set.

StratifiedKFold: This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.
KFoldCrossValidation without stratification just splits data into k folds. Then, each fold 1 <= i <= k is used once as the test set, while the others are used for training. The results are averaged in the end. It's similar to running the ShuffleSplit k times.
Stratification will ensure that the percentages of each class in your entire data will be the same (or very close to) within each individual fold.

Reference used: https://stackoverflow.com/questions/32615429/k-fold-stratified-cross-validation-with-imbalanced-classes

Solution Statement

In this project, we will employ several supervised algorithms to accurately model household's income levels using data of Costa Rican household characteristics. We will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Our goal with this implementation is to construct a model that accurately classifies income levels of families in given 4 categories.

Since its supervised learning problem, we have to find relationship of target variable with 142 features (dependent variables) using training set in such a way that model could be generalized to unseen data. As no of features are slightly high, we will try to eliminate unimportant features using feature selection methods and train model with reduced feature space. It will help reduce overfitting and improve model training-duration and performance.

To properly evaluate the performance of each model we will choose, we will create a training and predicting pipeline that allows us to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. We will evaluate performance using evaluation metrics for classification algorithms: f-score, accuracy. Following execution of pipeline for different models, we will pick the best-performing model and tune its hyper-parameters to achieve best results.

Before we pass our training data to machine learning algorithm, we must pre-process our data which typically includes steps:

- Handling missing values
- Treating outliers
- Transforming skewed continuous features
- Normalizing numerical features
- Encoding categorical features (Label encoding or one-hot encoding)

Below is the summary of supervised learning algorithms we will use to implement our solution:

DecisionTree: Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and

therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Strengths of the model: Simple to understand and interpret. It can be combined with other decision techniques.Decision trees implicitly perform variable screening or feature selection.Decision trees require relatively little effort from users for data preparation(Normalization,scaling etc.).Decision trees do not require any assumptions of linearity in the data,therefore nonlinear relationships between parameters do not affect tree performance.

Ada Boost: is short for Adaptive Boosting. It is basically a machine learning algorithm that is used as a classifier. Whenever you have a large amount of data and you want divide it into different categories, we need a good classification algorithm to do it. We usually use AdaBoost in conjunction with other learning algorithms to improve their performance. Hence the word 'boosting', as in it boosts other algorithms! Boosting is a general method for improving the accuracy of any given learning algorithm. So obviously, adaptive boosting refers to a boosting algorithm that can adjust itself to changing scenarios.

AdaBoost was formulated by Yoav Freund and Robert Schapire. Problems in machine learning tend to suffer from the curse of dimensionality. What it means is that a single feature point ends up having a huge dimensionality. As in, the feature vector used to describe a particular thing can be very huge. So if each sample consists of a huge number of potential features, the overall system becomes very slow. This is the reason we cannot use a powerful model with a full feature set because it cannot run in real time. We can only afford to have simple machine learning algorithms. But if the algorithms are too simple, they tend to be less accurate. They are called 'weak learners'. So we cascade them together to create a strong classifier. The output of 'weak learners' is combined into a weighted sum that represents the final output of the boosted classifier.

How is it "adaptive"?

AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. When we collect the set of weak learners, we give equal weights to all of them. But on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. If you consider our earlier example of classifying people in San Francisco, it's like saying that you trust all three people equally. But with each passing day, you realize that the second person is more accurate than the other two. So you give more weightage to his guesses and less weightage to the other two. This way, the overall accuracy will increase. Basically, the weak learner's job is the find a weak hypothesis that is appropriate for the current distribution of the data.

References: http://mccormickml.com/2013/12/13/adaboost-tutorial/
https://prateekvjoshi.com/2014/05/05/what-is-adaboost/

Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' or by taking the majority vote in the case of classification.

Reference: https://en.wikipedia.org/wiki/Random_forest

Benchmark Model

First, I will train random forest model and obtain F1 score, which is going to be threshold score and model would be benchmark model.

After getting benchmark model, I will train other models based on various classification algorithms as specified in solution section with inclusion of techniques: feature selection, hyperparameter tuning etc. and target to obtain better F1 score than benchmark model's F1 value.

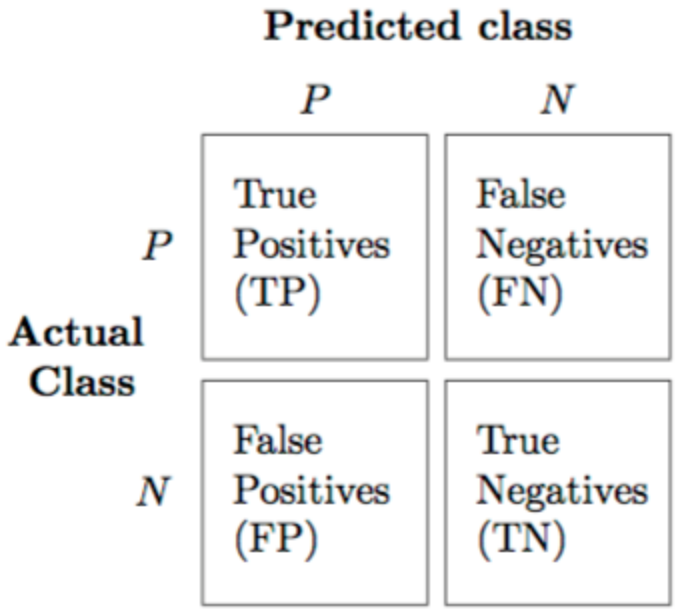
Evaluation Metrics

Submission will be evaluated based on macro F1 score.

References used: https://sebastianraschka.com/faq/docs/multiclass-metric.html
https://en.wikipedia.org/wiki/F1_score

In statistical analysis of binary classification, the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

We can generalize all the binary performance metrics such as precision, recall, and F1-score etc. to multi-class settings. In the binary case, we have



$$PRE = \frac{TP}{TP + FP}$$

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

(PRE=precision, REC=recall, F1=F1-Score) And to generalize this to multi-class, assuming we have a One-vs-All (OvA) classifier, we can either go with the “micro” average or the “macro” average. In “micro averaging,” we’d calculate the performance, e.g., precision, from the individual true positives, true negatives, false positives, and false negatives of the the k-class model:

$$PRE_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k}$$

And in macro-averaging, we average the performances of each individual class:

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k}$$

Project Design

We will start with data exploration and pre-processing. Data exploration includes descriptive analysis of dataset and identifying missing values/outliers etc. Pre-processing should include below steps:

Transforming Skewed Continuous Features: For highly-skewed feature distributions, it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers.

Normalizing Numerical Features: In addition to performing transformations on features that are highly skewed, 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 however, normalization ensures that each feature is treated equally when applying supervised learners.

In addition, we would check for categorical variables, if any, and convert it to numerical features using label encoder/one hot encoder.

After above pre-processing steps, we will shuffle and split data into training and test sets. 80% of the data will be used for training and 20% for testing.

The following are some of the supervised learning models that we will use to model our dataset and pick specific learner which achieves best score.

- 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

To properly evaluate the performance of each model, we will create a training and predicting pipeline that would help us quickly and effectively train models using various sizes of training data and perform predictions on the testing data. Finally, we will choose from the above-mentioned supervised learning models the best model to use on the costa rica data. We will then perform a grid search optimization for the selected model over the entire training set by tuning at parameters to improve upon the untuned model's F-score.