UDACITY

‹ Return to "Machine Learning Engineer Nanodegree" in the classroom

Finding Donors for CharityML

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Congratulations on completing the project 🏆

All your implementation in this submission is great and meets the rubric expectation 👏

You have done an awesome job on the discussion sections too 😄

Keep up the Impressive work; All the best for the rest of the nano-degree and for all your ML endeavors 👍

Exploring the Data

Student's implementation correctly calculates the following:

- **Number of records**
- **Number of individuals with income >$50,000**
- **Number of individuals with income <=$50,000**
- **Percentage of individuals with income > $50,000**

Good Job getting the dataset stats ! 😎

You have correctly calculated:

- Number of records : 45222 : ✔️
- Number of individuals with income >$50,000 : 11208 : ✔️

- Number of individuals with income <=$50,000 : 34014 : ✔
- Percentage of individuals with income > $50,000 : 24.78% : ✔

Below are a few other methods you can use to get your label counts.

- Value Counts :

```
n_at_most_50k, n_greater_50k = data.income.value_counts()
```

- Shape : `Shape` relates to the size of the dimensions of an N-dimensional array. Because generally we want multi-dimensional arrays, `array.reshape` allows us to get from 1D to an nD array. So, when we call `shape`, we get the N dimension shape of the array, so we can see exactly how the array looks like.

- `shape[0]` gives no of rows.
- `shape[1]` gives no of columns.

```
n_greater_50k = data[data['income']=='>50K'].shape[0]
```

- Size: Size regarding arrays, relates to the amount (or count) of elements that are contained in the array (or sometimes, at the top dimension of the array - when used as length).

In essence, `size` is equal to the product of the elements of `shape`. `size = shape[0] x shape[1]`

For example, let `A` be a matrix:

```
1  2  3  4
5  6  7  8
9 10 11 12
```

The `shape` of `A` is (3, 4), `shape[0]` is 3, `shape[1]` is 4 and the size of `A` is `12`.

Note: In case when the Target Class is imbalanced (same as in our current scenario)

From the dataset stats we can see that the dataset we have is an Imbalanced dataset! Which means that the ratio of individuals making more than $50k vs those making less are not in proportion, and in such cases we should make sure that the metric we will be using for model evaluation can capture how well the model is actually doing. (Additional info here)

Here is another nice article which discusses handling of imbalanced dataset which should be resourceful to you!

As you can see, In this project we are using the precision, recall, and F-beta scores(`beta` being 0.5). **F-beta score** is a metric that considers both precision and recall, In particular, when `beta = 0.5`, more emphasis

is placed on precision than recall.
Alternatively we can also consider F1 score (which is equivalent to using F-beta with `beta = 1`).

F1 Score:

Often referred to as the traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Preparing the Data

Student correctly implements one-hot encoding for the feature and income data.

Great Job on encoding the features and target labels using `get_dummies`.
Your code outputs total 103 features after One Hot Encoding, which is the expected value.

Something to note here is that we can also use Label Encoder as an alternative in case where we have a huge number of output classes Multi Class predictions. Label Encoder can be implemented as below:

```
encoder = LabelEncoder()
income = encoder.fit_transform(income_raw)
```

Also please go through the below article which explains when and why we use OHE
Link

For your reference, check out this post that demonstrates various ways in which we can handle categorical variables. 😎

Evaluating Model Performance

Student correctly calculates the benchmark score of the naive predictor for both accuracy and F1 scores.

Great work on correctly expressing Accuracy, Precision and Recall! Also, well done getting the right expression for the F-score. 👍

- Both accuracy and F-score are correctly calculated: `Accuracy score: 0.2478` , `F-Score: 0.2917`

💡 Note:

- As `TN = 0` and `FN = 0`; both the Accuracy and Precision in our case is the same!
- Also chcek out this nice article which explains perormance metrics for Classification Problems!

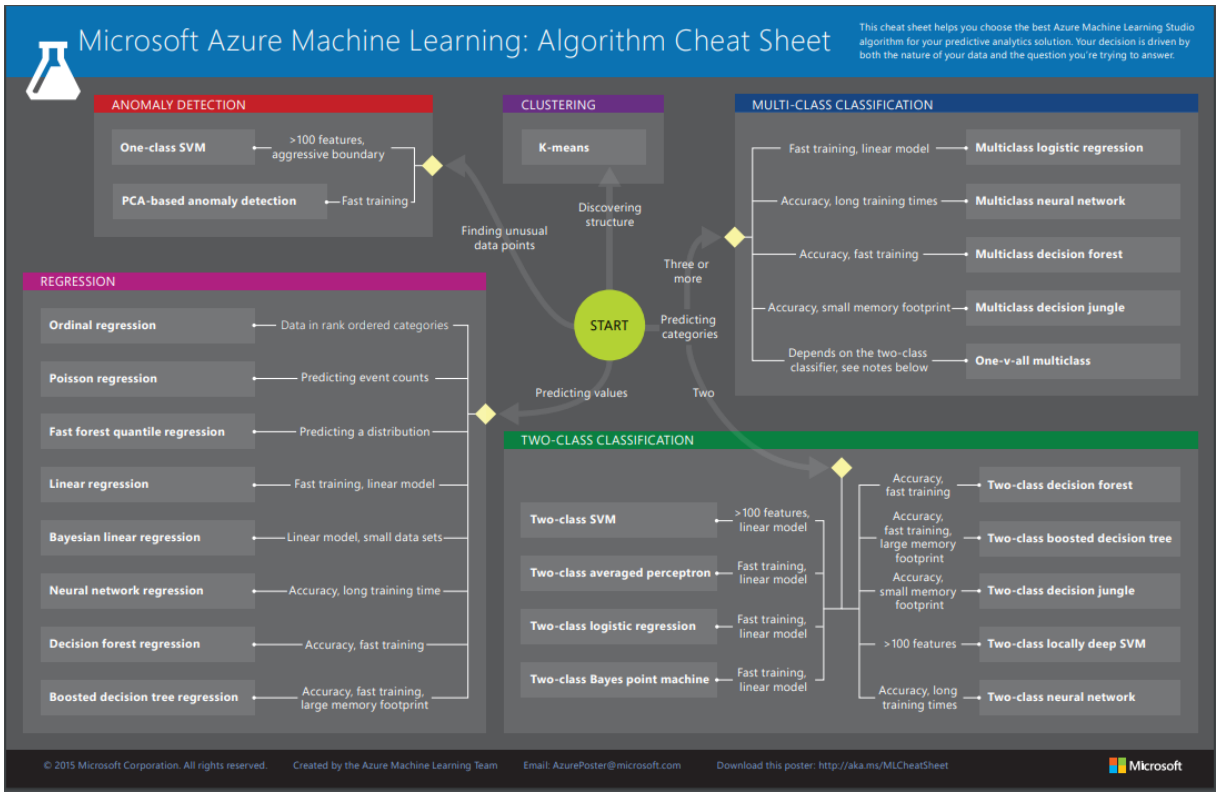
The pros and cons or application for each model is provided with reasonable justification why each model was chosen to be explored.

Please list all the references you use while listing out your pros and cons.

Nice discussion. Seems like you really understood the models.

To know the pros and cons of a model is really important as this will help you select the best model based on the data-set properties. If the data-set contains a property which a model generally does well on, selecting that model is always a good choice.

As a side-note, please check out this Quora post and the below image which will help while selecting a model from a bunch of options and which model is suitable for what kind of a problem.



Student successfully implements a pipeline in code that will train and predict on the supervised learning algorithm given.

Nice Job on implementing the pipeline correctly 👍

ML pipelines are awesome tools. A Pipeline can specified as a sequence of stages which runs these sequence of stages one after the other. What this achieves is that, imagine you are working on large ML projects and you need to find out which model performs best. You'll then often experiment with the models to see which one works better and also what combination of parameters work best. A pipeline helps us with

repeatable task without the same implementation steps for each algorithm every time. It basically helps you with the hard work.

There are standard workflows in applied machine learning. Standard because they overcome common problems like data leakage in your test harness. Scikit-learn provides a Pipeline utility to help automate machine learning workflows. Pipelines work by allowing for a linear sequence of data transforms to be chained together culminating in a modeling process that can be evaluated.
The goal is to ensure that all of the steps in the pipeline are constrained to the data available for the evaluation, such as the training dataset or each fold of the cross-validation procedure.
You can learn more about Pipelines in scikit-learn from the API documentation Here

Student correctly implements three supervised learning models and produces a performance visualization.

👍

Improving Results

Justification is provided for which model appears to be the best to use given computational cost, model performance, and the characteristics of the data.

Student is able to clearly and concisely describe how the optimal model works in layman's terms to someone who is not familiar with machine learning nor has a technical background.

Great discussion here! 👍

You have put in a lot of effort to explain the model as simply as possible. 👏

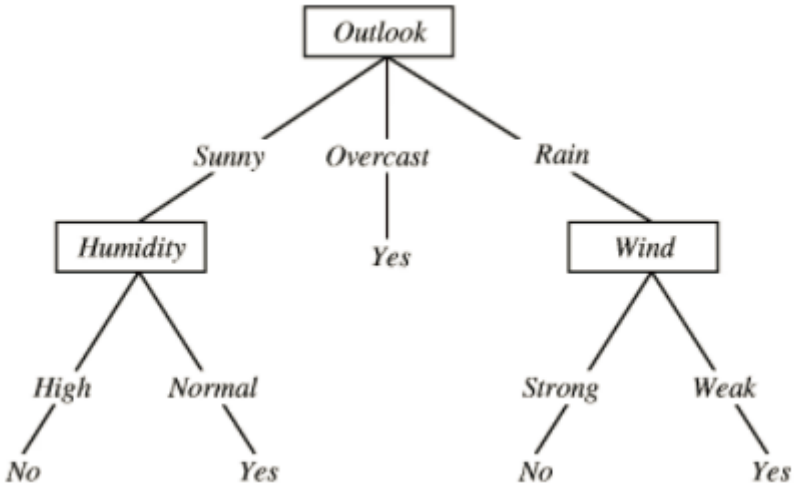
💡 **Note:** For your reference please go through the Layman's explanation of a **Random Forrest Classifier** and a **Logistic Regression Model** which should be resourceful:

The Random Forest Classifier

Random Forest Classifier is considered to be an ensemble method type of model. Ensemble methods basically gather together the results of multiple trained models to create a much better model. In the case of the Random Forest Classifier, it takes the results of multiple decision tree models, uses their indivudial predictions to "vote" for one, much more informed prediction. For example, if there are five descision trees in our Random Forest model, and three of them predict a label of 1 and two of the predict a label of 0, our Random Forest model will return a prediction of 1.

To better understand the underlying nature of the Random Forest Classifier, we must also understand the Decision Tree Classifier. A decision tree can be thought of like a flow chart built of nodes, branches, and leafs. At each node, we ask a question of the data. Ex. What is the outlook?. The branch represents the possible answers, in this case, sunny, overcast, or rain. At the end of the branch is a leaf which represents

the classification label. In this example, if it is ovecast, then yes, the event will happen. Decision Trees can be made of multiple levels, a variable we call depth. Below is an example of one such decision tree graph.



In this example, based on features about the weather, we are trying to predict whether and event will happen. Now, imagine building this model multiple times with random samples from our dataset to give us a more informed prediction.

Training and predicting:
So how does a decision tree learn? How does the model know where to split the data at a node into sepearate branches? Well, remember that the goal of a decision tree is to classify a record with a certain label. And the goal of each split in the tree is to separate the data as much as possible into the possible labels. The ideal split for example would be for the left branch to only contain Class A labels and the right branch to only contain Class B labels. The tree measures the effectivness of the split by using something called a loss function, which could either be the gini index or entropy. The model will go through the whole dataset and calculate the loss function for all possible splits, to find the one that describes the data the best. It would then use that particular split and increase the depth one level. Then it would calculate the loss function again for each node at that level and make the appropriate splits until the max depth of the tree is reached.

When making a prediction, the data is fed into the tree and runs through the trained model using the previously determined split thresholds until it ends at a leaf, resulting in the predicted label.

And remember, with Random Forest models, this happens with many trees and each tree results in a "vote" for the label. Majority vote wins!

Logistic Regression:

Considering Logistic Regression to be the best fit for our problem, let us try to understand what regression is and then let us try to explain what Logistic Regression is (In terms of Statistics/Machine Learning).
Regression is, in the simplest terms, understanding the relationship between the input and the output variables. This linear regression is something that gives a linear relationship between the input and output variables.
Logistic regression is basically a linear regression model which is explains the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables with only difference being, the output for linear regression is a number that has its real meaning(a label) while the output of a logistic regression is a number that represents the probability of the event happening (i.e. the probability of people clicking an ad online, the probability of death in titanic disaster, etc.). In our case predicting the probability of an individual making more money than $50,000. The intuition behind

logistic regression is to transform the output of a linear regression to a range of (0,1) in which the probability lies. So, if for a person we are trying to find if he likes apple or mango, Liner regression will give an output like A or B (A: person likes Apple, B: person likes Mango) while logistic regression will output probabilities for both the A and B. So, if (Probability)A > Probability(B), the person likes Apple.
Now as we understood what Logistic regression is and what it does, let us try to understand how our logistic Regression Model works. Our purpose is to classify a person as earning more than 50k or earning less than 50k. For our task, we have some data which has certain features like Age, Education Level, Hours per Week. So, from basic understanding we would say that a person having higher education who works more hours per week would earn more. And someone with same education working same hours, but younger will earn less because he has less work experience. Our model also works the same way, but with huge no of features and huge population of data. It analyzes all the features and figures our relationship between the features. It finds out which features greatly impacts a person's earning and which does not. So, after the model is done with figuring out the training data, it defines a boundary. This process is called training the Model. After training we are ready to predict if a person earns > 50 k or not. In this process, the model will input the data features to the already trained model and output probabilities belonging to each group. The person will belong to the class whose probability is greater.

Also I have added a few links which explains different ML models in simple terms. Go through each and it will help you improve your understanding on them.

https://www.quora.com/What-is-logistic-regression
https://rayli.net/blog/data/top-10-data-mining-algorithms-in-plain-english/
http://blog.echen.me/2011/03/14/laymans-introduction-to-random-forests/
https://prateekvjoshi.com/2014/05/05/what-is-adaboost/

The final model chosen is correctly tuned using grid search with at least one parameter using at least three settings. If the model does not need any parameter tuning it is explicitly stated with reasonable justification.

Awesome! You have correctly implemented **GridSearchCV** ! 👍

Now that you have got the correct Implementation of GrisSearchCV, lets take a moment to think why are we using GridSearchCV in the first place. Can we not search for the optimum parameters manually ? The answer is: we can, we don't need to use GridSearchCV and can write all the required code manually. Without GridSearchCV we would need to loop over the parameters and then run all the combinations of parameters. If we were then to use cross-validated result, we would also need to add the code to find the best average CV results across all the combinations of aparameters. Rather than doing all this coding and hard work, we have used GridSearchCV to find the optimum parameters from a given set of values.

GridSearch is not the only technique available to us though! Another similar technique worth looking is RandomizedSearchCV

While `GridSearchCV` works with the explicitly declared values, `RandomizedSearchCV` doesn't search every value. Instead, it samples random subsets and then uses them to find the optimum parameter value.

Now, first thing that might be coming to you mind is: "Do I use `RandomizedSearchCV` over our beloved `GridSearchCV` ! It just searches through random values right ?! How would that ever be helpful !"

But in applied ML scenarios, we have seen that that `RandomizedSearchCV` performs better than its counterpart, by often finding the optimal value much faster. So it turns out `RandomizedSearchCV` is great. Another great thing about this method is that it samples from value distribution rather than the pre-determined values.

Now when do we use it, or more precisely in which situations will it work best. `RandomizedSearchCV` is great when we have a handful no of hyperparameters .

Here is a post regarding `RandomizedSearchCV` which i highly recommend for you to check out if you are curious !

Student reports the accuracy and F1 score of the optimized, unoptimized, models correctly in the table provided. Student compares the final model results to previous results obtained.

Feature Importance

Student ranks five features which they believe to be the most relevant for predicting an individual's' income. Discussion is provided for why these features were chosen.

Logical Selection of features and the justification intuitively makes sense ! 👍🏽

Student correctly implements a supervised learning model that makes use of the `feature_importances_` attribute. Additionally, student discusses the differences or similarities between the features they considered relevant and the reported relevant features.

Great work here!

Just for your curious mind! What is Feature Selection and why is it important? 😉

As data collection has become easy and efficient, datasets now a day have become very rich in information. So understanding the dataset is always a good choice as this always saves a lot of troubles later. The data are often high dimensional and it is quite common to see datasets with hundreds of features and is not unusual to see it go to tens of thousands.

So, first thing that comes to our mind is: More features mean better for the model right ? Well not always ! When a model is presented data with very high dimensionality which means a huge number of features, models usually choke. This is because:

1. As no of features increases, training time increases exponentially.
2. Models often suffer from overfitting with more no of features.
   As this nice article points out, *"Sometimes Less is better!"*

Feature Selection is a very critical step in any Machine Learning algorithm's workflow. Machine learning follows a simple rule : if you put garbage in, you will only get garbage as your output. In our case garbage will be noisy data.
What feature selection does is that it prioritizes features so that you can decide how many of them you'd like to choose.
Top reasons to use feature selection are:
• It enables the algorithm to train faster.
• It reduces the complexity of a model and makes it easier to interpret.
• It improves the accuracy of a model if the right subset is chosen.
• It reduces overfitting.

Something to note here is that there are a lot of methods for implementing feature selection. Below are some of the most common one:
• SelectBest
• SelectPercentile
A more advanced method is Recursive Feature Elimination. Recursive feature elimination is based on the idea to repeatedly construct a model (for example an SVM or a regression model) and choose either the best or worst performing feature (for example based on coefficients), setting the feature aside and then repeating the process with the rest of the features. This process is applied until all features in the dataset are exhausted. Features are then ranked according to when they were eliminated. As such, it is a greedy optimization for finding the best performing subset of features.

Tips:
Below are a few posts for your reference which you should check out:
https://machinelearningmastery.com/an-introduction-to-feature-selection/
http://blog.datadive.net/selecting-good-features-part-iv-stability-selection-rfe-and-everything-side-by-side/
https://topepo.github.io/caret/recursive-feature-elimination.html

Student analyzes the final model's performance when only the top 5 features are used and compares this performance to the optimized model from Question 5.

Great work!

Please note that often more than not, in instances where we really value training time or training time is a crucial deciding factor for model selection, we can also opt for a simpler model.
Because cutting down on `features` on complex models often result in worse performance and worse training time.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review