FastAI-Notes

10 October 2018 10:20

python partial dependence plot toolbox

From <https://github.com/SauceCat/PDPbox>

When using black box machine learning algorithms like random forest and boosting, it is hard to understand the relations betw een predictors and model outcome.

For example, in terms of random forest, all we get is the feature importance.

Although we can know which feature is significantly influencing the outcome based on the importance calculation, it really su cks that we don't know in which direction it is influencing. And in most of the real cases, the effect is non-monotonic.

We need some powerful tools to help understanding the complex relations between predictors and model prediction.

1. Helper functions for visualizing target distribution as well as prediction distribution.
2. Proper way to handle one-hot encoding features.
3. Solution for handling complex mutual dependency among features.
4. Support multi-class classifier.
5. Support two variable interaction partial dependence plot.

Lesson1Notes:

Fastai import bunch of useful tools.

- Random forest is a kind of universal machine learning technique
- It can be used for both regression (target is a continuous variable) or classification (target is a categorical variable) pro blems
- It also works with columns of any kinds, like pixel values, zip codes, revenue, etc.
- In general, random forest does not overfit (it's very easy to stop it from overfitting)
- You do not need a separate validation set in general. It can tell you how well it generalizes even if you only have one datas et
- It has few (if any) statistical assumptions (it doesn't assume that data is normally distributed, data is linear, or that you need to specify the interactions)
- Requires very few feature engineering tactics, so it's a great place to start. For many different types of situations, you do not have to take the log of the data or multiply interactions together

Curse of dimensionality

The curse of dimensionality is the idea that the more dimensions we have, the more points sit on the edge of that space. So i f the number of columns is more, it creates more and more empty space.

What that means, in theory, is that the distance between points is much less meaningful. This should not be true because the points still are different distances away from each other. Even though they are on the edges, we can still determine how far away they are from each other.

Continuous, categorical, ordinal variables

- Continuous variables are variables with integer or float values. For example – Age, distance, weight, income etc
- Categorical variables are usually strings or values representing names or labels. For example – gender, state, zip code, rank etc
- Ordinal variables are those categorical variables which have an order among them.
  For example rank (I, II, III) or remark (poor, good, excellent) have an order.

- Underfitting: A model that performs poorly on the train data and also on the test data. The model does not generalize well.
- Overfitting: A model that performs extremely well on the train data but does not show a similar high performance on the test data.

Data Pre-Processing:

From the date column, we can extract numerical values such as – year, month, day of month, day of the week, holiday or not, weekend or weekday, was it raining?, etc.

add_datepart function from the fastai library to create these features for us. The function creates the following features:

'Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear', 'Is_month_end', 'Is_month_start',
'Is_quarter_end', 'Is_quarter_start', 'Is_year_end', 'Is_year_start'

Syntax:
add_datepart(df, fldname, drop=True, time=False)

Docstring: add_datepart converts a column of df from a datetime64 to many columns containing the information from the date. This applies changes inplace.
Parameters:
df: A pandas data frame. df gain several new columns.
fldname: A string that is the name of the date column you wish to expand.
If it is not a datetime64 series, it will be converted to one with pd.to_datetime.
drop: If true then the original date column will be removed.
time: If true time features: Hour, Minute, Second will be added.

The categorical variables are currently stored as strings, which is inefficient, and doesn't provide the numeric coding requi red for a random forest.
Therefore we call train_cats function from fastai to convert strings to pandas categories.

train_cats(df_raw)

Signature: train_cats(df)
Docstring:
Change any columns of strings in a panda's dataframe to a column of categorical values. This applies the changes inplace.
Parameters:
df: A pandas dataframe. Any columns of strings will be changed to categorical values.

While converting categorical to numeric columns, we have to take the following two issues into consideration:

1. Some categorical variables can have an order among them (for example – High>Medium>Low). We can use *set_categories* to set the order.

   df_raw.UsageBand.cat.set_categories(['High', 'Medium', 'Low'], ordered=True, inplace=True)

2. If a category gets a particular number in the train data, it should have the same value in the test data.
   For instance, if the train data has 3 for high and test data has 2, then it will have two different meanings. We can use *apply_cats* for validation and test sets to make sure that the mappings are the same throughout the different sets

Signature: apply_cats(df, trn)
Docstring: Changes any columns of strings in df into categorical variables using trn as a template for the category codes.

Parameters:

df: A pandas dataframe. Any columns of strings will be changed to categorical values. The category codes are determined by tr n.
trn: A pandas dataframe. When creating a category for df, it looks up the what the category's code were in trn and makes thos e the category codes for df.

Missing Value Treatment:

display_all(df_raw.isnull().sum().sort_index()/len(df_raw))
We use *.isnull().sum()* to get the total number of missing values. This is divided by the length of the dataset to determine the ratio of missing val ues.

We have to impute the missing values and store the data as dependent and independent part. This is done by using the fastai f unction *proc_df*.

The function performs the following tasks:

- For continuous variables, it checks whether a column has missing values or not
- If the column has missing values, it creates another column called *columnname_na*, which has 1 for missing and 0 for not missing
- Simultaneously, the missing values are replaced with the median of the column
- For categorical variables, pandas replaces missing values with -1. So *proc_df* adds 1 to all the values for categorical variables. Thus, we have 0 for missing while all other values are incremented by 1.

```
df, y, nas = proc_df(df_raw, 'SalePrice')
```

```
m = RandomForestRegressor(n_jobs=-1)
m.fit(df, y)
m.score(df,y)
```

The *n_jobs* is set to -1 to use all the available cores on the machine. This gives us a score (r^2) of 0.98, which is excellent.

The caveat here is that we have trained the model on the training set, and checked the result on the same. There's a high cha nce that this model might not perform as well on unseen data (test set, in our case).The only way to find out is to create a validation set and check the performance of the model on it.

Lesson 2 Notes:

Hyperparameter are the tuning parameters that change how your model behaves.

Remove the

Creating a Validation set

Creating a good validation set that closely resembles the test set is one of the most important tasks in machine learning. Th e validation score is representative of how our model performs on real -world data, or on the test data.

Keep in mind that if there's a time component involved, then the most recent rows should be included in the validation set

Building a single tree:

Random Forest is a group of trees which are called estimators. The number of trees in a random forest model is defined by the parameter *n_estimator*.

In the bagging technique, we create multiple models, each giving predictions which are not correlated to the other. Then we a verage the predictions from these models.
Random Forest is a bagging technique.
If all the trees created are similar to each other and give similar predictions, then averaging these predictions will not im prove the model performance.
Instead, we can create multiple trees on a different subset of data, so that even if these trees overfit, they will do so on a different set of points.
These samples are taken with replacement.

Each models are build on the subset of the dataset so the errors are not co-related with each other. to construct multiple models so the errors are not co-related with each other.
sampling with replacement(n rows with replacement)

In simple words, we create multiple poor performing models and average them to create one good model. The individual models must be as predictive as possible, but together should be uncorrelated. We will now increase the number of estimators in our random forest and see the results.

to find a variable which is important. ML model is effective if it generalizes well in test data.
uncorrelated trees - correlation between the trees should be very less. extratressregressor - randomly try few split with few variables

Out of Bag error:

Is our validation set worse than our training set because we're over-fitting, or because the validation set is for a different time period, or a bit of both? With the existing information we've shown, we can't tell. However, random forests have a very clever trick called *out-of-bag (OOB) error* which can handle this (and more!)
The idea is to calculate error on the training set, but only include the trees in the calculation of a row's error where that row was *not* included in training that tree.
This allows us to see whether the model is over-fitting, without needing a separate validation set.
This also has the benefit of allowing us to see whether our model generalizes, even if we only have a small amount of data so want to avoid separating some out to create a validation set.

Hyperparameter set_rf_samples()

- pick up a subset of rows
- summarize relationship between hyperparameters and its effects on overfitting, collinearity
  set_rf_samples(20000) determines how many rows of data in each tree
  - Step 1: we have a big dataset, grab a subset of data and build a tree
  - we either bootstrap a sample (sample with replacement) or subset a small number of rows

- When you decrease the sample size, it means that there are less final decisions that can be made; tree will be less rich; it also is making less binary choices to get to those decision
- setting set_rf_samples lower means you overfit less, but you'll have a less accurate tree model
- each individual tree (aka "estimator") is as accurate as possible on the training set
- across the estimators, the correlation between them is as low as possible, so when you average them out together, you end up with something that generalizes
- by decreasing the set_rf_samples() number, we are actually decreasing the power of the estimator and increasing the correlation. it may result in a better or wo rse validation set result; this is the compromise you have to figure out when you do ML models.
- oob=True whatever your subsample is, take all the remaining rows, and put them into a dataset and calculate the error on those (it doe sn't impact the training set); it's a quasi-validation set

Hyperparameter Min sample leaf

This can be treated as the stopping criteria for the tree. The tree stops growing (or splitting) when the number of samples i n the leaf node is less than specified.

```
m = RandomForestRegressor(n_estimators=40, min_samples_leaf=3,n_jobs= -1, oob_score=True)
```
Here we have specified the *min_sample_leaf* as 3. This means that the minimum number of samples in the node should be 3 for each split.

Hyperparameter Max feature

Another important parameter in random forest is *max_features*. We have discussed previously that the individual trees must be as uncorrelated as possible. For the same,

random forest uses a subset of rows to train each tree.

Additionally, we can also use a subset of columns (features) instead of using all the features. This is achieved by tweaking the *max_features* parameter.

m = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features=0.5, n_jobs= -1, oob_score=True)

max_features is set to 0.5 which means using 50% of the features for each split.

We can also increase the amount of variation amongst the trees by not only use a sample of rows for each tree, but to also us ing a sample of *columns* for each *split*. We do this by specifying max_features, which is the proportion of features to randomly select from at each split.

- None
- 0.5
- 'sqrt'

max_features=1,0.5 ,sqrt,log2

Lesson3: Notes

Median of the numeric may be different in test set

df,y,nas= proc_df()

Value of the dictionary are medians

But if we are set_rf_samples less then we are decreasing the power of the estimator and increasing the correlation between t hem

Each individual estimator is as accurate(strong) as possible on training set but across the estimators(tree) the correlation between them is as low as possible. On averaging them together it will genearlizes well

Feature importance:

It's not normally enough to just to know that a model can make accurate predictions - we also want to know *how* it's making predictions. The most important way to see this is with *feature importance*.

Bias- average