BLOG

BENCHMARK FOR POVER-T TEST - PREDICTING POVERTY

TUE 19 DECEMBER 2017

Measuring poverty is hard(http://www.worldbank.org/en/topic/measuringpoverty), but The World Bank plans to end extreme poverty by 2030 – and they need your help! Poverty levels are often estimated at the country level by extrapolating the results of surveys taken on a subset of the population at the household or individual level.

The surveys are incredibly informative, but they are also incredibly long. A typical poverty survey has hundreds of questions, ranging from region-specific questions to questions about the last time a participant bought bread. In order to track progress towards its goal, The World Bank needs the most efficient survey possible. That's where you come in.

For more information, explore this recent World Bank report(http://www.worldbank.org/en/news/press-release/2017/10/17/world-bank-urges-action-to-break-the-cycle-of-poverty-from-generation-to-generation) on ending the cycle of poverty.

In our brand new competition(https://www.drivendata.org/competitions/50/worldbank-poverty-prediction/page/97/), we're asking you to predict poverty at the household level by building a great classification model. The strongest poverty predictors could be used by statisticians at The World Bank to design new, shorter, equally informative surveys. With these improvements, The World Bank can more easily track progress towards their ambitious and inspiring goal(http://www.worldbank.org/en/events/2017/03/21/ending-poverty-the-road-to-2030).

In this post, we'll walk through a very simple first pass model for poverty prediction from survey data, showing you how to load the data, make some predictions, and then submit those predictions to the competition.

To get started, we summon the tools of the trade.

In [1]:
```
%matplotlib inline

import os

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

# data directory
DATA_DIR = os.path.join('..', 'data', 'processed')
```

LOADING THE DATA

Check out this research focused on children(https://www.unicef.org/publications/index_92826.html) and the effort to end extreme poverty.

On the data download page(https://www.drivendata.org/competitions/50/worldbank-poverty-prediction/data/), we provide a couple of datasets to get started:

- **Household-level survey data:** This is obfuscated data from surveys conducted by The World Bank, focusing on household-level statistics. The data come from three different countries, and are separated into different files for convenience.
- **Individual-level survey data:** This is obfuscated data from related surveys conducted by The World Bank, only these focus on individual-level statistics. The set of interviewees and countries involved are the same as the household data, as indicated by shared `id` indices, but this data includes detailed (obfuscated) information about household members.
- **Submission format:** This gives us the filenames and columns of our submission prediction, filled with all `0.5` as a baseline.

In classic benchmark fashion, we're going to keep this analysis short, sweet, and simple. As such, **we're not going to use any of the individual-level data** that's included with the competition. Even without individual-level data, we have a lot of files to deal with. It's probably worth it to store our paths in an easily-accessible dictionary:

In [2]:
```
data_paths = {'A': {'train': os.path.join(DATA_DIR,
                    'test':  os.path.join(DATA_DIR,

              'B': {'train': os.path.join(DATA_DIR,
                    'test':  os.path.join(DATA_DIR,

              'C': {'train': os.path.join(DATA_DIR,
                    'test':  os.path.join(DATA_DIR,
```

In [3]:
```
# load training data
a_train = pd.read_csv(data_paths['A']['train'], in
b_train = pd.read_csv(data_paths['B']['train'], in
c_train = pd.read_csv(data_paths['C']['train'], in
```

As usual, let's take a quick look at the head.

In [4]:
```
a_train.head()
```

Out[4]:

	wBXbHZmp	SlDKnCuu	KAJOWiiw	DsKacCdL	rt
id					
46107	JhtDR	GUusz	TuovO	ZYabk	fe
82739	JhtDR	GUusz	TuovO	ZYabk	fe
9646	JhtDR	GUusz	BIZns	ZYabk	u
10975	JhtDR	GUusz	TuovO	ZYabk	fe
16463	JhtDR	alLXR	TuovO	ZYabk	fe

5 rows × 345 columns

In [5]:
```
b_train.head()
```

Out[5]:

	RzaXNcgd	LfWEhutl	jXOqJdNL	wJthinfa	PTLgv
id					
57071	zTghO	pYfmQ	INhMv	42	RQnVj
18973	zTghO	pYfmQ	INhMv	34	iuxWN
20151	zTghO	pYfmQ	INhMv	34	iuxWN
5730	zTghO	pYfmQ	INhMv	58	iuxWN
35033	zTghO	pYfmQ	INhMv	122	iuxWN

5 rows × 442 columns

In [6]:
```
c_train.head()
```

Out[6]:

	GRGAYimk	DNnBfiSl	cNDTCUPU	GvTJUYOo	vr
id					
57211	RslOh	SuNUt	gJLrc	EPKkJ	qK
62519	jPUAt	boDkl	gJLrc	EPKkJ	YX
11614	OpTiw	boDkl	vURog	EPKkJ	qK
6470	RslOh	VgxgY	gJLrc	EPKkJ	YX
33558	IXFlv	VgxgY	kPTaD	EPKkJ	YX

5 rows × 164 columns

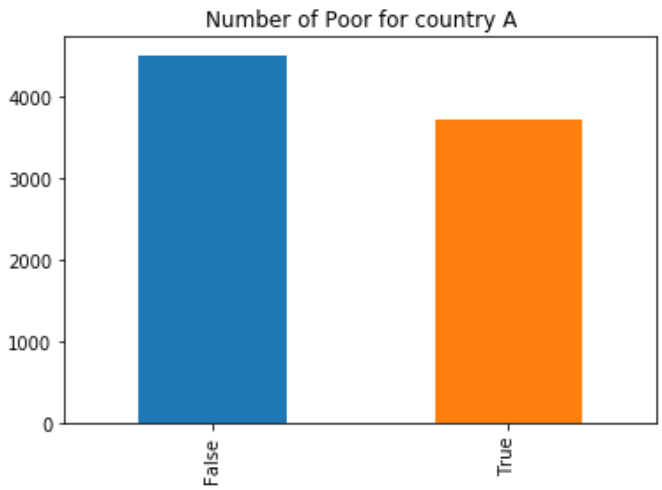
The first thing to notice is that each country's surveys have wildly different numbers of columns, so **we'll plan on training separate models for each country and combining our predictions for submission** at the end.

POVERTY DISTRIBUTIONS

Let's take a look at the class distributions for each country. In classification tasks, it's crucial to know the balance of class labels!

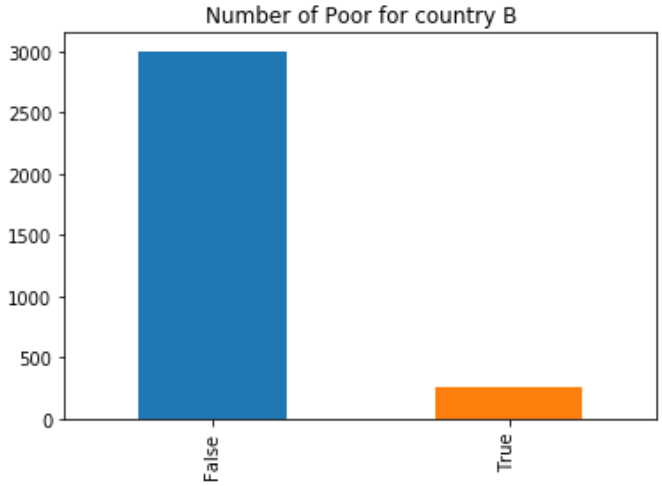
In [7]:
```
a_train.poor.value_counts().plot.bar(title='Number
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x116C
```



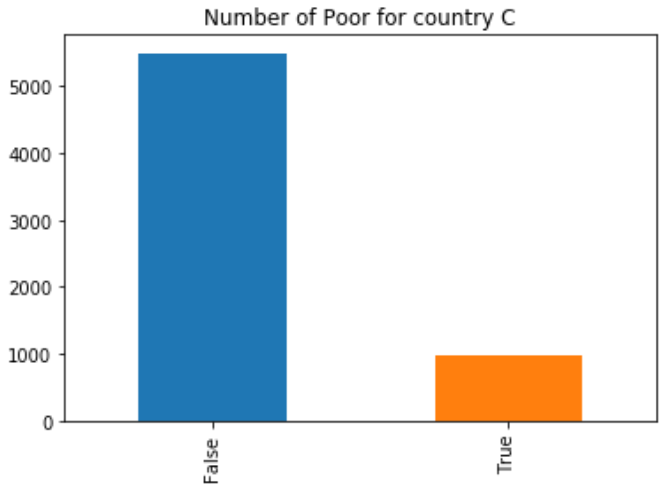
In [8]:
```
b_train.poor.value_counts().plot.bar(title='Number
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1175
```



In [9]:
```
c_train.poor.value_counts().plot.bar(title='Number
```

Out[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1183
```



Country A is well-balanced, but countries B and C are quite unbalanced. This could definitely impact the confidence of our predictor. But solving that problem is up to you – it's outside the scope of this humble benchmark.

We expect most of the data types here to be the dreaded `object` type, but let's make sure.

In [10]:
```
a_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8203 entries, 46107 to 39832
Columns: 345 entries, wBXbHZmp to country
dtypes: bool(1), float64(2), int64(2), object(34
memory usage: 21.6+ MB
```

In [11]:
```
b_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3255 entries, 57071 to 4923
Columns: 442 entries, RzaXNcgd to country
dtypes: bool(1), float64(9), int64(14), object(4
memory usage: 11.0+ MB
```

In [12]:
```
c_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6469 entries, 57211 to 7646
Columns: 164 entries, GRGAYimk to country
dtypes: bool(1), float64(1), int64(29), object(1
memory usage: 8.1+ MB
```

Sure enough, the `bool` types are our labels--the `poor` column--then there are a few numeric types with the rest being object. We'll need to convert the `object` columns to categorical variables before training anything.

Pre-process the Data

We're going to do some simple pre-processing here. Standardizing the data and converting the object types to categoricals should get us pretty far. Let's write a couple of simple functions to help this effort.

In [13]:

```
# Standardize features
def standardize(df, numeric_only=True):
    numeric = df.select_dtypes(include=['int64', '

    # subtracy mean and divide by std
    df[numeric.columns] = (numeric - numeric.mean(

    return df


def pre_process_data(df, enforce_cols=None):
    print("Input shape:\t{}".format(df.shape))


    df = standardize(df)
    print("After standardization {}".format(df.sha

    # create dummy variables for categoricals
    df = pd.get_dummies(df)
    print("After converting categoricals:\t{}".for


    # match test set and training set columns
    if enforce_cols is not None:
        to_drop = np.setdiff1d(df.columns, enforce_
        to_add = np.setdiff1d(enforce_cols, df.colu

        df.drop(to_drop, axis=1, inplace=True)
        df = df.assign(**{c: 0 for c in to_add})

    df.fillna(0, inplace=True)

    return df
```

Time to convert these surveys!

In [14]:
```
print("Country A")
aX_train = pre_process_data(a_train.drop('poor', a
ay_train = np.ravel(a_train.poor)

print("\nCountry B")
bX_train = pre_process_data(b_train.drop('poor', a
by_train = np.ravel(b_train.poor)

print("\nCountry C")
cX_train = pre_process_data(c_train.drop('poor', a
cy_train = np.ravel(c_train.poor)
```

```
Country A
Input shape:    (8203, 344)
After standardization (8203, 344)
After converting categoricals:  (8203, 859)

Country B
Input shape:    (3255, 441)
After standardization (3255, 441)
After converting categoricals:  (3255, 1432)

Country C
Input shape:    (6469, 163)
After standardization (6469, 163)
After converting categoricals:  (6469, 795)
```

The data is probably looking pretty different now. Let's take a peek at country A.

In [15]:
```
aX_train.head()
```

Out[15]:

	nEsgxvAq	OMtioXZZ	YFMZwKrU	TiwRsIOh	wBX
id					
46107	-1.447160	0.325746	1.099716	-0.628045	0
82739	-0.414625	-0.503468	-0.016050	0.713467	0
9646	0.617910	-0.503468	-0.016050	-0.628045	0
10975	0.617910	-1.332682	-1.131816	0.713467	0
16463	0.617910	0.325746	-1.131816	-0.180874	0

5 rows × 859 columns

Oh yeah, now *that* looks like the kind of matrix `scikit-learn` wants to process!

THE ERROR METRIC -

MEANLOGLOSS

The error metric for this competition is our old friend, log loss ... with a twist. Since we're predicting for three countries, our overall score is going to be the *mean* of the log losses for each country. However, the countries labels are conditionally independent, so in practice we should be able to train three independent models and combine their predictions for submission.

See the competition submission page for more info on the metric!

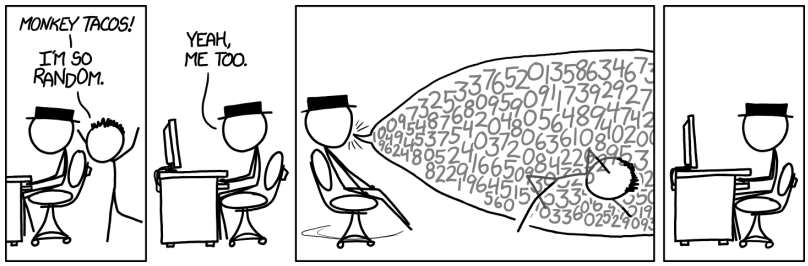
BUILD THE MODEL

As mentioned above, we're keeping this benchmark short, sweet, and simple. So where do we turn when looking for a great out-of-the-box model? If you answered "Random Forests!" then we may just be two trees of the same ensemble. No? Then perhaps we're... splitting on the same node? At any rate, random forests are often a good model to try first(https://medium.com/rants-on-machine-learning/the-unreasonable-effectiveness-of-random-forests-f33c3ce28883), especially when we have numeric and categorical variables in our feature space.

Random Forest

In `scikit-learn`, it almost couldn't be easier to grow a random forest with a few lines of code.

In [16]:
```
from sklearn.ensemble import RandomForestClassifi

def train_model(features, labels, **kwargs):

    # instantiate model
    model = RandomForestClassifier(n_estimators=50,

    # train model
    model.fit(features, labels)

    # get a (not-very-useful) sense of performance
    accuracy = model.score(features, labels)
    print(f"In-sample accuracy: {accuracy:0.2%}")

    return model
```



Another classic from xkcd(https://xkcd.com/1210/).

That's it as far model building is concerned. Let's grow some trees!

In [17]:
```
model_a = train_model(aX_train, ay_train)
```

```
In-sample accuracy: 100.00%
```

In [18]:
```
model_b = train_model(bX_train, by_train)
```

```
In-sample accuracy: 99.94%
```

In [19]:
```
model_c = train_model(cX_train, cy_train)
```

```
In-sample accuracy: 100.00%
```

TIME TO PREDICT AND SUBMIT

Remember, accuracy is not a very informative metric, especially when dealing with imbalanced classes(https://blogs.msdn.microsoft.com/data_insights_global_practice/2016/02/29/evaluatin machine-learning-models-when-dealing-with-imbalanced-classes/). Furthermore, accuracy is not the metric for this competition!

The above scores suggest little more than an overfit training set. But it's confidence that counts – we'll need to use the `.predict_proba()` method to generate our submissions. Let's load up the test data, process it, and see what we get.

In [20]:
```
# load test data
a_test = pd.read_csv(data_paths['A']['test'], inde
b_test = pd.read_csv(data_paths['B']['test'], inde
c_test = pd.read_csv(data_paths['C']['test'], inde
```

In [21]:
```
# process the test data
a_test = pre_process_data(a_test, enforce_cols=aX_
b_test = pre_process_data(b_test, enforce_cols=bX_
c_test = pre_process_data(c_test, enforce_cols=cX_
```

```
Input shape:    (4041, 344)
After standardization (4041, 344)
After converting categoricals:  (4041, 851)
Input shape:    (1604, 441)
After standardization (1604, 441)
After converting categoricals:  (1604, 1419)
Input shape:    (3187, 163)
After standardization (3187, 163)
After converting categoricals:  (3187, 773)
```

Note that we're taking a very simple approach to filling missing values, as well as enforcing column consistency after converting to categoricals. (See the preprocessing function again to see what `enforce_cols` actually does.)

Make Predictions

To return the confidence probabilities that the submission format requires, we need to call the `predict_proba()` method on our models.

In [22]:
```
a_preds = model_a.predict_proba(a_test)
b_preds = model_b.predict_proba(b_test)
c_preds = model_c.predict_proba(c_test)
```

That was easy enough. Time to format the predictions and send them on their way.

Save Submission

We'll write a simple function that converts the predictions a `DataFrame` and adds a column for the correct country code.

In [23]:
```
def make_country_sub(preds, test_feat, country):
    # make sure we code the country correctly
    country_codes = ['A', 'B', 'C']

    # get just the poor probabilities
    country_sub = pd.DataFrame(data=preds[:, 1],
                               columns=['poor'],
                               index=test_feat.ind

    # add the country code for joining later
    country_sub["country"] = country
    return country_sub[["country", "poor"]]
```

In [24]:
```
# convert preds to data frames
a_sub = make_country_sub(a_preds, a_test, 'A')
b_sub = make_country_sub(b_preds, b_test, 'B')
c_sub = make_country_sub(c_preds, c_test, 'C')
```

Finally, it's time to combine our predictions and save for submission!

In [25]:
```
submission = pd.concat([a_sub, b_sub, c_sub])
```

How about one last look at the fruits of or hard work...

In [26]:
```
submission.head()
```

Out[26]:

	country	poor
id		
418	A	0.32
41249	A	0.28
16205	A	0.26
97501	A	0.36
67756	A	0.26

In [27]:
```
submission.tail()
```

Out[27]:

	country	poor
id		
6775	C	0.30
88300	C	0.20
35424	C	0.20
81668	C	0.28
98377	C	0.18

Looks good, let's save and send'er off!

In [28]:
```
submission.to_csv('submission.csv')
```

Submit to Leaderboard

Woohoo! We processed your submission!

Your score for this submission is:

0.5739

Woohoo! It's a start! And that's exactly what we intend with these benchmarks. We're sure you'll be able to top this model in no time, and we can't wait to see what you come up with(https://www.drivendata.org/competitions/50/worldbank-poverty-prediction/).

Visit The World Bank's site(http://www.worldbank.org/en/topic/measuringpoverty) to learn more about how poverty is measured.

(HTTP://DRIVENDATA.CO/FEEDS/ALL.ATOM.XML)

COMPETITION SITE(HTTP://WWW.DRIVENDATA.ORG)

PROJECTS(HTTP://DRIVENDATA.CO#PROJECTS)

SERVICES(HTTP://DRIVENDATA.CO#SERVICES)

TEAM(HTTP://DRIVENDATA.CO#TEAM)

LABS (/)

BLOG(HTTP://DRIVENDATA.CO/BLOG.HTML)

CONTACT(HTTP://DRIVENDATA.CO#CONTACT)

(//TWITTER.COM/DRIVENDATAORG)

(//WWW.LINKEDIN.COM/COMPANY/9202422/)

(//GITHUB.COM/DRIVENDATAORG)