DontOverfit

January 4, 2018

0.1 Implementation of Salisman's Don't Overfit submission

From Kaggle >In order to achieve this we have created a simulated data set with 200 variables and 20,000 cases. An 'equation' based on this data was created in order to generate a Target to be predicted. Given the all 20,000 cases, the problem is very easy to solve – but you only get given the Target value of 250 cases – the task is to build a model that gives the best predictions on the remaining 19,750 cases.

```
In [30]: import gzip
         import requests
         import zipfile

         url = "https://dl.dropbox.com/s/lnly9gw8pb1xhir/overfitting.zip"


         results = requests.get(url)

In [31]: import StringIO
         z = zipfile.ZipFile(StringIO.StringIO(results.content))
         # z.extractall()

In [32]: z.extractall()

In [38]: z.namelist()

Out[38]: ['overfitting.csv']

In [69]: d = z.open('overfitting.csv')
         d.readline()

Out[69]: 'case_id,train,Target_Practice,Target_Leaderboard,Target_Evaluate,var_1,var_2,var_3,va

In [70]: import numpy as np

In [63]: M = np.fromstring(d.read(), sep=",")

In [71]: len(d.read())

Out[71]: 23919756
```

```
In [75]: np.fromstring?

In [ ]:

In [4]: data = np.loadtxt("overfitting.csv", delimiter=",", skiprows=1)

In [5]: print """
        There are also 5 other fields,

        case_id - 1 to 20,000, a unique identifier for each row

        train - 1/0, this is a flag for the first 250 rows which are the training dataset

        Target_Practice - we have provided all 20,000 Targets for this model, so you can develo

        Target_Leaderboard - only 250 Targets are provided. You submit your predictions for the

        Target_Evaluate - again only 250 Targets are provided. Those competitors who beat the

        """

        data.shape

There are also 5 other fields,

case_id - 1 to 20,000, a unique identifier for each row

train - 1/0, this is a flag for the first 250 rows which are the training dataset

Target_Practice - we have provided all 20,000 Targets for this model, so you can develop your m

Target_Leaderboard - only 250 Targets are provided. You submit your predictions for the remaini

Target_Evaluate - again only 250 Targets are provided. Those competitors who beat the 'benchmar



Out[5]: (20000L, 205L)

In [6]: ix_training = data[:, 1] == 1
        ix_testing = data[:, 1] == 0

        training_data = data[ix_training, 5:]
        testing_data = data[ix_testing, 5:]

        training_labels = data[ix_training, 2]
        testing_labels = data[ix_testing, 2]
```

```
        print "training:", training_data.shape, training_labels.shape
        print "testing: ", testing_data.shape, testing_labels.shape
```

```
training: (250L, 200L) (250L,)
testing:  (19750L, 200L) (19750L,)
```

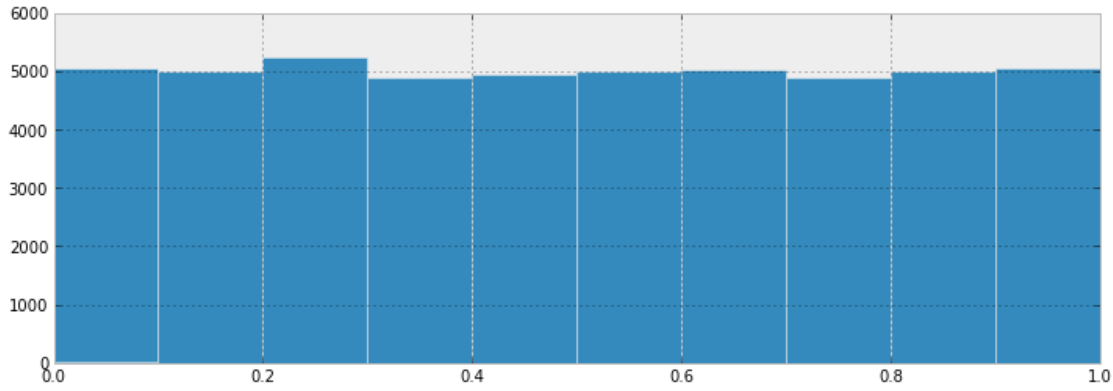
0.2 Develop Tim's model

He mentions that the X variables are from a Uniform distribution. Let's investigate this:

```
In [7]: figsize(12, 4)
```

```
In [8]: hist(training_data.flatten())
        print training_data.shape[0] * training_data.shape[1]
```

```
50000
```



looks pretty right

```
In [127]: import pymc as pm

          to_include = pm.Bernoulli("to_include", 0.5, size=200)
```

```
In [128]: coef = pm.Uniform("coefs", 0, 1, size=200)
```

```
In [129]: @pm.deterministic
          def Z(coef=coef, to_include=to_include, data=training_data):
              ym = np.dot(to_include * training_data, coef)
              return ym - ym.mean()
```

```
In [130]: @pm.deterministic
          def T(z=Z):
              return 0.45 * (np.sign(z) + 1.1)
```

```
In [132]: obs = pm.Bernoulli("obs", T, value=training_labels, observed=True)

          model = pm.Model([to_include, coef, Z, T, obs])
          map_ = pm.MAP(model)
          map_.fit()
```

```
Warning: Stochastic to_include's value is neither numerical nor array with floating-point dtype
```

```
In [133]: mcmc = pm.MCMC(model)
```

```
In [176]: mcmc.sample(100000, 90000, 1)
```

```
[****************100%******************]  100000 of 100000 complete
```

```
In [177]: (np.round(T.value) == training_labels).mean()
```

```
Out[177]: 0.73999999999999999
```

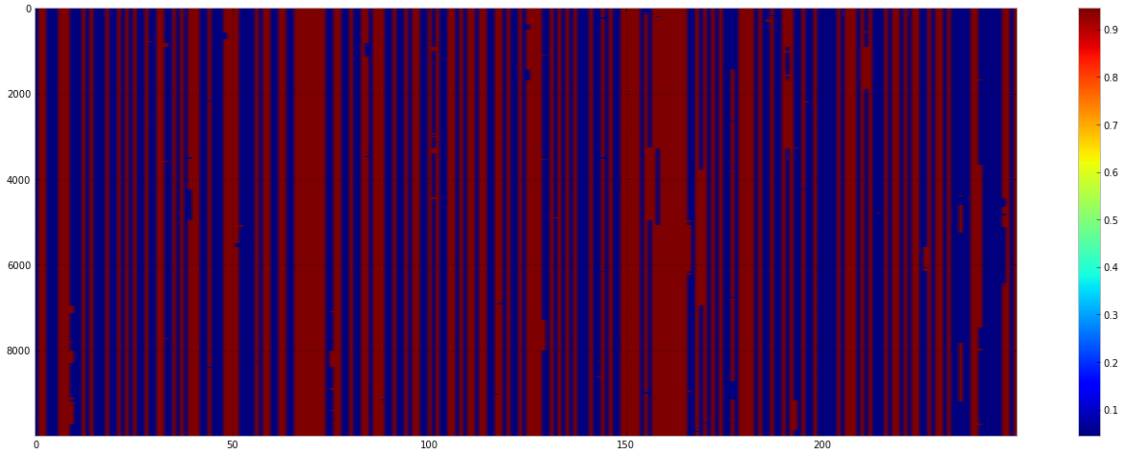
```
In [178]: t_trace = mcmc.trace("T")[:]
          (np.round(t_trace[-500:-400, :]).mean(axis=0) == training_labels).mean()
```

```
Out[178]: 0.72399999999999998
```

```
In [179]: t_mean = np.round(t_trace).mean(axis=1)
```

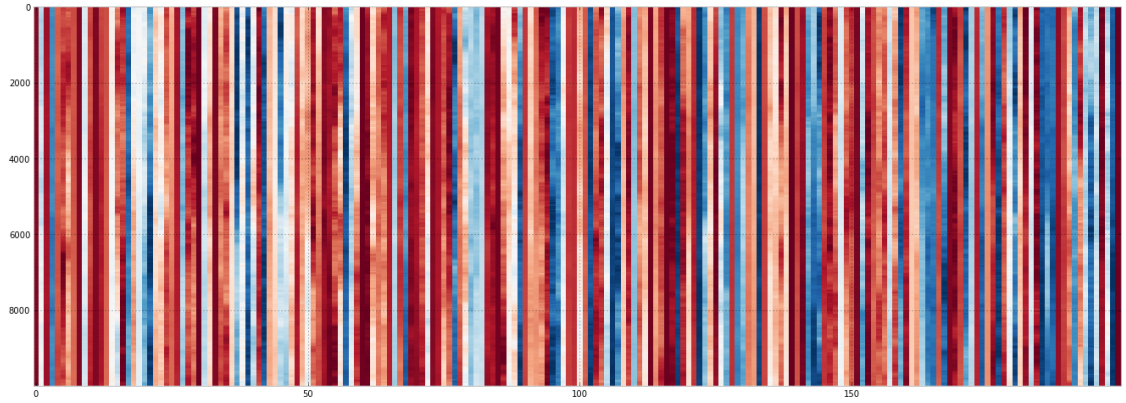
```
In [180]: imshow(t_trace[-10000:, :], aspect="auto")
          colorbar()
```

```
Out[180]: <matplotlib.colorbar.Colorbar instance at 0x0000000013270208>
```




```
In [181]: figsize(23, 8)
          coef_trace = mcmc.trace("coefs")[:]
          imshow(coef_trace[-10000:, :], aspect="auto", cmap=pyplot.cm.RdBu, interpolation="nor
```

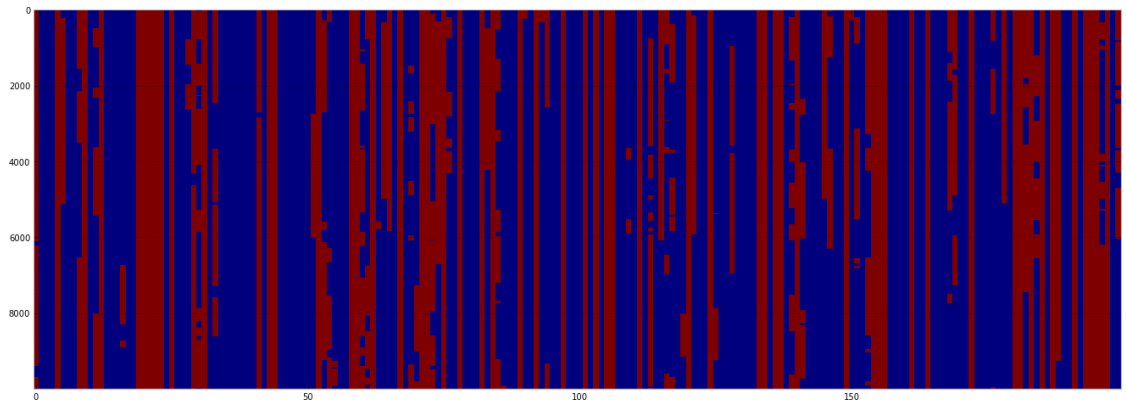
Out[181]: <matplotlib.image.AxesImage at 0x19ce2780>



In [183]:
```
include_trace = mcmc.trace("to_include")[:]
```

In [184]:
```
figsize(23, 8)
imshow(include_trace[-10000:, :], aspect="auto", interpolation="none")
```

Out[184]: <matplotlib.image.AxesImage at 0x18d8ef60>



In [ ]: