In [3]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler

from sklearn import linear_model
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler

import warnings
warnings.filterwarnings('ignore')

np.random.seed(1)
```

In [4]:

```
# setting up default plotting parameters
%matplotlib inline

plt.rcParams['figure.figsize'] = [20.0, 7.0]
plt.rcParams.update({'font.size': 22,})

sns.set_palette('viridis')
sns.set_style('white')
sns.set_context('talk', font_scale=0.8)
```

Datasets

train1.csv - column name 'index' is the name of the observation and columns 0-15 are the sixteen features associated with each entity

train2.csv - column name 'index' is the name of the observation and columns 0-19 are the nineteen features associated with each entity

train3.csv - column name 'index' is the name of the observation and column 'label' is the target (response/dependent) variable associated with each entity

Initialize datasets

In [5]:

```
train1 = pd.read_csv('https://raw.githubusercontent.com/pranavn91/blockchain/m
aster/2011gcn.csv')
train2 = pd.read_csv('https://raw.githubusercontent.com/pranavn91/blockchain/m
aster/tx2011partvertices_new.csv')
train3 = pd.read_csv('https://raw.githubusercontent.com/pranavn91/blockchain/m
aster/tx2011partvertices.csv')

print('Train 1 Shape: ', train1.shape)
print('Train 2 Shape: ', train2.shape)
print('Train 3 Shape: ', train3.shape)
```

```
Train 1 Shape:  (96498, 17)
Train 2 Shape:  (96498, 20)
Train 3 Shape:  (96498, 2)
```

In [6]:

```
train1.rename(columns={'Unnamed: 0':'index'}, inplace=True)
train1['index'] = train1['index'] + 1
train1.head()
```

Out[6]:

	index	0	1	2	3	4	5	6	7	8	
0	1	0.0	4.811244e+07	0.0	0.0	5.298934e+07	0.0	5.215435e+07	0.0	0.0	4.293262e
1	2	0.0	3.477977e+05	0.0	0.0	3.777575e+05	0.0	3.757520e+05	0.0	0.0	6.889133e
2	3	0.0	6.455196e+07	0.0	0.0	7.110790e+07	0.0	6.997804e+07	0.0	0.0	5.670157e
3	4	0.0	2.009876e+08	0.0	0.0	2.214679e+08	0.0	2.174101e+08	0.0	0.0	1.342720e
4	5	0.0	2.384675e+05	0.0	0.0	2.597246e+05	0.0	2.577884e+05	0.0	0.0	4.202993e

In [7]:

```
train2.rename(columns={'Unnamed: 0':'index'}, inplace=True)
train2.head()
```

Out[7]:

	index	txsize	txvirtualsize	txinputs_count	txoutputs_count	txinput_val	txoutput_val
0	1	7369	7369	5	190	215000000.0	214600000.0
1	2	293	293	1	3	4400000.0	4350000.0
2	3	11139	11139	1	322	125000000.0	124400000.0
3	4	495	495	1	9	27450000.0	27400000.0
4	5	462	462	1	8	3000000.0	2950000.0

In [8]:

```
train3.rename(columns={'Unnamed: 0':'index'}, inplace=True)
train3.head()
```

Out[8]:

	index	label
0	1	unclassified
1	2	donations
2	3	unclassified
3	4	donations
4	5	donations

Info

In [9]:

```
train1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96498 entries, 0 to 96497
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   index   96498 non-null  int64  
 1   0       96498 non-null  float64
 2   1       96498 non-null  float64
 3   2       96498 non-null  float64
 4   3       96498 non-null  float64
 5   4       96498 non-null  float64
 6   5       96498 non-null  float64
 7   6       96498 non-null  float64
 8   7       96498 non-null  float64
 9   8       96498 non-null  float64
 10  9       96498 non-null  float64
 11  10      96498 non-null  float64
 12  11      96498 non-null  float64
 13  12      96498 non-null  float64
 14  13      96498 non-null  float64
 15  14      96498 non-null  float64
 16  15      96498 non-null  float64
dtypes: float64(16), int64(1)
memory usage: 12.5 MB
```

In [10]:

```
train2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96498 entries, 0 to 96497
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             96498 non-null  int64  
 1   txsize            96498 non-null  int64  
 2   txvirtualsize     96498 non-null  int64  
 3   txinputs_count    96498 non-null  int64  
 4   txoutputs_count   96498 non-null  int64  
 5   txinput_val       96498 non-null  float64
 6   txoutput_val      96498 non-null  float64
 7   txfee             96498 non-null  int64  
 8   Min_received      96498 non-null  float64
 9   Max_received      96498 non-null  float64
 10  Avg_received      96498 non-null  float64
 11  Total_received    96498 non-null  float64
 12  Stdev_received    96498 non-null  float64
 13  Var_received      96498 non-null  float64
 14  Min_sent          96498 non-null  float64
 15  Max_sent          96498 non-null  float64
 16  Avg_sent          96498 non-null  float64
 17  Total_sent        96498 non-null  float64
 18  Stdev_sent        96498 non-null  float64
 19  Var_sent          96498 non-null  float64
dtypes: float64(14), int64(6)
memory usage: 14.7 MB
```

In [11]:

```
train3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96498 entries, 0 to 96497
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   index   96498 non-null  int64 
 1   label   96498 non-null  object
dtypes: int64(1), object(1)
memory usage: 1.5+ MB
```

Cleaning

In [12]:

```
results = pd.merge(train3, train1, on='index', how='inner')

# split data into X and y
X = results.iloc[:,2:18]
scaler = StandardScaler()
X = scaler.fit_transform(X)
Y1 = results['label']
```

In [13]:

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(results['label'].unique())
Y = pd.DataFrame(le.transform(Y1))
```

In [14]:

```
Y.nunique()
```

Out[14]:

```
0    6
dtype: int64
```

Train test split

In [15]:

```
seed = 1
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)
```

classification

In [16]:

```
# define models
logistic = linear_model.LogisticRegression(solver='liblinear')
sgd = linear_model.SGDClassifier()
```

In [17]:

```
models = [logistic, sgd]
```

In [18]:

```
# function to get cross validation scores
def get_cv_scores(model):
    scores = cross_val_score(model, X_train, y_train, cv=3, scoring='accuracy'
)
    print('CV Mean: ', np.mean(scores))
    print('STD: ', np.std(scores))
    print('\n')
```

In [19]:

```
# loop through list of models
for model in models:
    print(model)
    get_cv_scores(model)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10
0,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.00
01, verbose=0,
                   warm_start=False)
CV Mean:  0.8704241043757203
STD:  0.0008408605308690582


SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_int
ercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge'
,
              max_iter=1000, n_iter_no_change=5, n_jobs=None, pena
lty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.
001,
              validation_fraction=0.1, verbose=0, warm_start=False
)
CV Mean:  0.863429033523253
STD:  0.005747895882866403
```

Logistic reg and grid search

#class sklearn.model_selection.GridSearchCV(estimator, param_grid, *, scoring=None, n_jobs=None, #iid='deprecated', refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False) #n_jobs int, default=None #Number of jobs to run in parallel. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors. penalty = ['l1', 'l2'] C = [0.0001, 0.001, 0.01] class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}] solver = ['liblinear', 'saga'] param_grid = dict(penalty=penalty, C=C, class_weight=class_weight, solver=solver) grid = GridSearchCV(estimator=logistic, param_grid=param_grid, scoring='accuracy', verbose=1, n_jobs=-1) grid_result = grid.fit(X_train, y_train) print('Best Score: ', grid_result.best_score_) print('Best Params: ', grid_result.best_params_)logistic = linear_model.LogisticRegression(C=0.0001, class_weight={1:0.5, 0:0.5}, penalty='l2', solver='liblinear') get_cv_scores(logistic)from sklearn.metrics import accuracy_score logistic.fit(X_train, y_train) y_train_pred = logistic.predict(X_train) accuracy_train = accuracy_score(y_train, y_train_pred) print("Accuracy: %.2f%%" % (accuracy_train)) y_test_pred = logistic.predict(X_test) accuracy_test = accuracy_score(y_test, y_test_pred) print("Accuracy: %.2f%%" % (accuracy_test))Best Score: 0.7839596742612616 Best Params: {'C': 0.0001, 'class_weight': {1: 0.5, 0: 0.5}, 'penalty': 'l2', 'solver': 'liblinear'}

Gradient descent and random search

loss = ['hinge', 'log'] penalty = ['l1', 'l2'] alpha = [0.0001, 0.001] learning_rate = ['constant', 'optimal'] class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}] eta0 = [1, 10] param_distributions = dict(loss=loss, penalty=penalty, alpha=alpha, learning_rate=learning_rate, class_weight=class_weight, eta0=eta0) random = RandomizedSearchCV(estimator=sgd, param_distributions=param_distributions, scoring='accuracy', verbose=1, n_jobs=-1, n_iter=1000) random_result = random.fit(X_train, y_train) print('Best Score: ', random_result.best_score_) print('Best Params: ', random_result.best_params_)Best Score: 0.6772044947639583 Best Params: {'penalty': 'l1', 'loss': 'hinge', 'learning_rate': 'optimal', 'eta0': 1, 'class_weight': {1: 0.4, 0: 0.6}, 'alpha': 0.001}sgd = linear_model.SGDClassifier(alpha=0.001, class_weight={1:0.4, 0:0.6}, eta0=1, learning_rate='optimal', loss='hinge', penalty='l1') get_cv_scores(sgd) sgd.fit(X_train, y_train) y_train_pred = sgd.predict(X_train) accuracy_train = accuracy_score(y_train, y_train_pred) print("Accuracy: %.2f%%" % (accuracy_train)) y_test_pred = sgd.predict(X_test) accuracy_test = accuracy_score(y_test, y_test_pred) print("Accuracy: %.2f%%" % (accuracy_test))Accuracy: 0.83% Accuracy: 0.83%

Neural Networks On the Dataset

In [20]:

```
from keras import models
from keras import layers
from keras.utils import to_categorical
```

In [21]:

```
Y = to_categorical(Y)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.4)
x_dev, x_test, y_dev, y_test = train_test_split(x_test, y_test, test_size = 0.
5)
# x_train = x_train.applymap(lambda x: np.log(x+1))
# x_dev = x_dev.applymap(lambda x: np.log(x+1))
# x_test = x_test.applymap(lambda x: np.log(x+1))

print(x_train.shape)
print(x_dev.shape)
print(x_test.shape)
print(y_train.shape)
print(y_dev.shape)
print(y_test.shape)
```

```
(57898, 16)
(19300, 16)
(19300, 16)
(57898, 6)
(19300, 6)
(19300, 6)
```

In [22]:

```
nn = models.Sequential()
nn.add(layers.Dense(128, activation = 'tanh', input_shape = (16, )))
nn.add(layers.Dense(64, activation = 'relu'))
nn.add(layers.Dense(36))
nn.add(layers.LeakyReLU(alpha = 0.01))
nn.add(layers.Dense(6, activation = 'softmax'))
nn.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy',
           metrics = ['accuracy'])
```

In [23]:

```
history = nn.fit(x_train, y_train, epochs = 30, validation_data=(x_dev, y_dev)
, batch_size = 10000)
nn.evaluate(x_dev, y_dev)[1]
```

```
Epoch 1/30
6/6 [==============================] - 0s 43ms/step - loss: 1.6612
- accuracy: 0.7982 - val_loss: 1.4852 - val_accuracy: 0.8365
Epoch 2/30
6/6 [==============================] - 0s 28ms/step - loss: 1.3500
- accuracy: 0.8395 - val_loss: 1.1459 - val_accuracy: 0.8365
Epoch 3/30
6/6 [==============================] - 0s 23ms/step - loss: 1.0038
- accuracy: 0.8398 - val_loss: 0.8200 - val_accuracy: 0.8366
Epoch 4/30
6/6 [==============================] - 0s 18ms/step - loss: 0.7456
- accuracy: 0.8398 - val_loss: 0.6878 - val_accuracy: 0.8366
```

```
Epoch 5/30
6/6 [==============================] - 0s 18ms/step - loss: 0.6645
- accuracy: 0.8398 - val_loss: 0.6561 - val_accuracy: 0.8367
Epoch 6/30
6/6 [==============================] - 0s 26ms/step - loss: 0.6393
- accuracy: 0.8404 - val_loss: 0.6377 - val_accuracy: 0.8388
Epoch 7/30
6/6 [==============================] - 0s 21ms/step - loss: 0.6220
- accuracy: 0.8422 - val_loss: 0.6231 - val_accuracy: 0.8389
Epoch 8/30
6/6 [==============================] - 0s 23ms/step - loss: 0.6073
- accuracy: 0.8427 - val_loss: 0.6103 - val_accuracy: 0.8395
Epoch 9/30
6/6 [==============================] - 0s 21ms/step - loss: 0.5947
- accuracy: 0.8453 - val_loss: 0.5996 - val_accuracy: 0.8439
Epoch 10/30
6/6 [==============================] - 0s 24ms/step - loss: 0.5843
- accuracy: 0.8498 - val_loss: 0.5895 - val_accuracy: 0.8524
Epoch 11/30
6/6 [==============================] - 0s 22ms/step - loss: 0.5735
- accuracy: 0.8580 - val_loss: 0.5801 - val_accuracy: 0.8574
Epoch 12/30
6/6 [==============================] - 0s 24ms/step - loss: 0.5656
- accuracy: 0.8638 - val_loss: 0.5737 - val_accuracy: 0.8652
Epoch 13/30
6/6 [==============================] - 0s 22ms/step - loss: 0.5592
- accuracy: 0.8682 - val_loss: 0.5667 - val_accuracy: 0.8657
Epoch 14/30
6/6 [==============================] - 0s 24ms/step - loss: 0.5527
- accuracy: 0.8698 - val_loss: 0.5618 - val_accuracy: 0.8660
Epoch 15/30
6/6 [==============================] - 0s 21ms/step - loss: 0.5499
- accuracy: 0.8699 - val_loss: 0.5586 - val_accuracy: 0.8663
Epoch 16/30
6/6 [==============================] - 0s 20ms/step - loss: 0.5447
- accuracy: 0.8704 - val_loss: 0.5556 - val_accuracy: 0.8664
Epoch 17/30
6/6 [==============================] - 0s 19ms/step - loss: 0.5411
- accuracy: 0.8708 - val_loss: 0.5543 - val_accuracy: 0.8666
Epoch 18/30
6/6 [==============================] - 0s 19ms/step - loss: 0.5410
- accuracy: 0.8708 - val_loss: 0.5508 - val_accuracy: 0.8666
Epoch 19/30
6/6 [==============================] - 0s 18ms/step - loss: 0.5386
- accuracy: 0.8705 - val_loss: 0.5490 - val_accuracy: 0.8666
Epoch 20/30
6/6 [==============================] - 0s 19ms/step - loss: 0.5363
- accuracy: 0.8709 - val_loss: 0.5544 - val_accuracy: 0.8658
Epoch 21/30
6/6 [==============================] - 0s 23ms/step - loss: 0.5372
- accuracy: 0.8706 - val_loss: 0.5485 - val_accuracy: 0.8666
Epoch 22/30
6/6 [==============================] - 0s 21ms/step - loss: 0.5344
```

```
- accuracy: 0.8709 - val_loss: 0.5481 - val_accuracy: 0.8673
Epoch 23/30
6/6 [==============================] - 0s 30ms/step - loss: 0.5359
- accuracy: 0.8707 - val_loss: 0.5462 - val_accuracy: 0.8667
Epoch 24/30
6/6 [==============================] - 0s 22ms/step - loss: 0.5329
- accuracy: 0.8709 - val_loss: 0.5452 - val_accuracy: 0.8667
Epoch 25/30
6/6 [==============================] - 0s 21ms/step - loss: 0.5342
- accuracy: 0.8710 - val_loss: 0.5441 - val_accuracy: 0.8668
Epoch 26/30
6/6 [==============================] - 0s 29ms/step - loss: 0.5326
- accuracy: 0.8709 - val_loss: 0.5451 - val_accuracy: 0.8668
Epoch 27/30
6/6 [==============================] - 0s 24ms/step - loss: 0.5343
- accuracy: 0.8709 - val_loss: 0.5447 - val_accuracy: 0.8669
Epoch 28/30
6/6 [==============================] - 0s 25ms/step - loss: 0.5312
- accuracy: 0.8711 - val_loss: 0.5427 - val_accuracy: 0.8675
Epoch 29/30
6/6 [==============================] - 0s 21ms/step - loss: 0.5313
- accuracy: 0.8710 - val_loss: 0.5417 - val_accuracy: 0.8673
Epoch 30/30
6/6 [==============================] - 0s 22ms/step - loss: 0.5321
- accuracy: 0.8711 - val_loss: 0.5442 - val_accuracy: 0.8676
604/604 [==============================] - 0s 667us/step - loss: 0
.5442 - accuracy: 0.8676
```

Out[23]:

```
0.8675647377967834
```

In [24]:

```
nn.evaluate(x_test, y_test)
```

```
604/604 [==============================] - 0s 717us/step - loss: 0
.5387 - accuracy: 0.8690
```

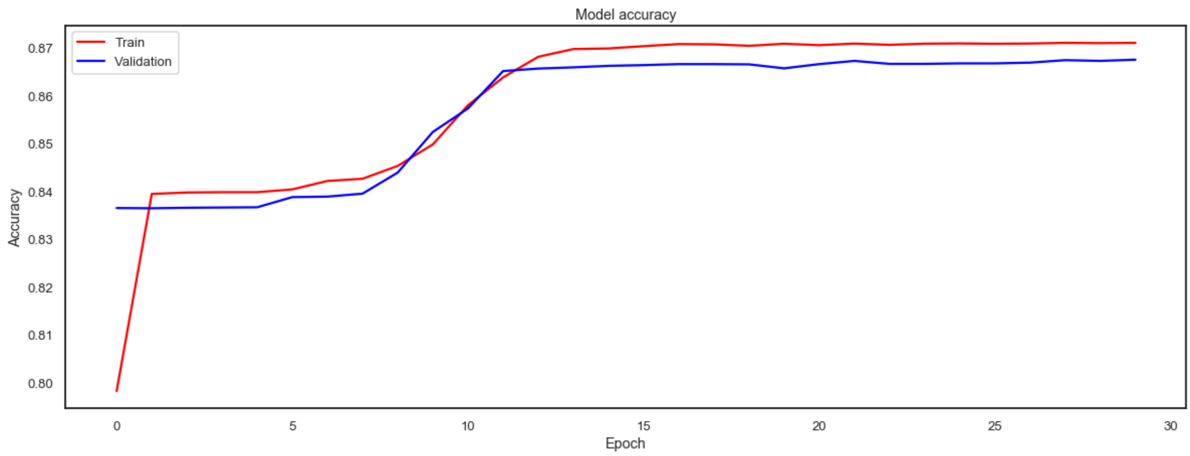
Out[24]:

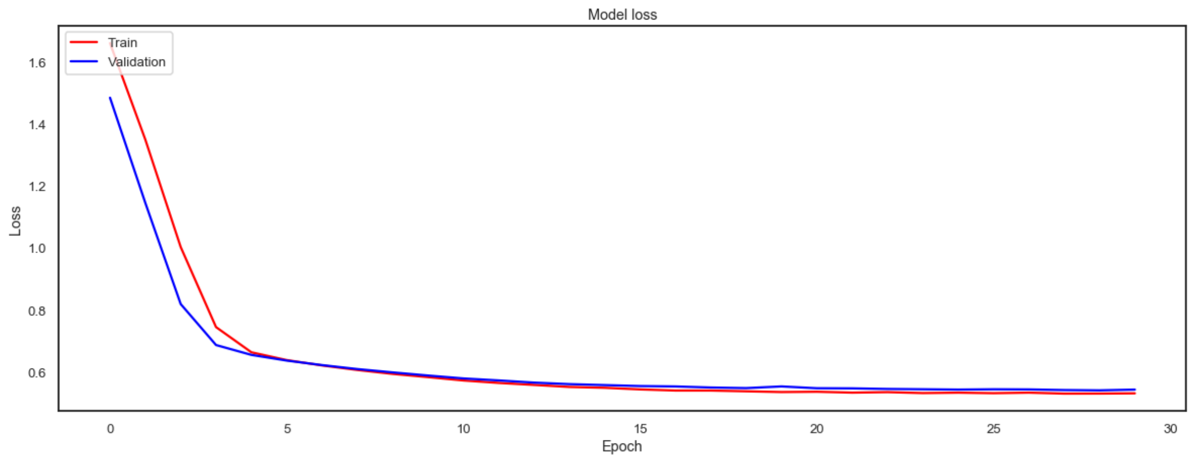
```
[0.5387383699417114, 0.8689637184143066]
```

In [25]:

```
import matplotlib.pyplot as plt
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'], color = 'red')
plt.plot(history.history['val_accuracy'], color = 'blue')
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'], color = 'red')
plt.plot(history.history['val_loss'], color = 'blue')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





In [ ]: