

import libraries

In [1]:

```
import pandas as pd
import numpy as np

# Load data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')

print(train.shape)
print(test.shape)
```

(300000, 25)
(200000, 24)

Look at the data

In [2]:

```
train.head()
```

Out[2]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_9
0	0	0	0	0	T	Y	Green	Triangle	Snake	Finland	...	2f4cb3d51
1	1	0	1	0	T	Y	Green	Trapezoid	Hamster	Russia	...	f83c56c21
2	2	0	0	0	F	Y	Blue	Trapezoid	Lion	Russia	...	ae6800dd0
3	3	0	1	0	F	Y	Red	Trapezoid	Snake	Canada	...	8270f0d71
4	4	0	0	0	F	N	Red	Trapezoid	Lion	Canada	...	b164b72a7

5 rows × 25 columns



In [3]:

```
test.head()
```

Out[3]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_4
0	300000	0	0	1	T	Y	Blue	Triangle	Axolotl	Finland	...	9d1173
1	300001	0	0	0	T	N	Red	Square	Lion	Canada	...	46ae30
2	300002	1	0	1	F	Y	Blue	Square	Dog	China	...	b759e3
3	300003	0	0	1	T	Y	Red	Star	Cat	China	...	0b6ec
4	300004	0	1	1	F	N	Red	Trapezoid	Dog	China	...	f91f3b

5 rows × 24 columns



Binary feature

In [4]:

```
train.head().iloc[:, 1:6]
```

Out[4]:

	bin_0	bin_1	bin_2	bin_3	bin_4
0	0	0	0	T	Y
1	0	1	0	T	Y
2	0	0	0	F	Y
3	0	1	0	F	Y
4	0	0	0	F	N

Nominal Features

In [5]:

```
train.head().iloc[:, 6:16]
```

Out[5]:

	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom
0	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000
1	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd9202
2	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9fc
3	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6at
4	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab1

nom_0~4 has fewer categories, so we will use OneHot encoding on them

In [6]:

```
train['nom_0'].unique(), train['nom_1'].unique(), train['nom_2'].unique(), train['nom_3'].unique(), train['nom_4'].unique()
```

Out[6]:

```
(array(['Green', 'Blue', 'Red'], dtype=object),
 array(['Triangle', 'Trapezoid', 'Polygon', 'Square', 'Star', 'Circle'],
      dtype=object),
 array(['Snake', 'Hamster', 'Lion', 'Cat', 'Dog', 'Axolotl'], dtype=object),
 array(['Finland', 'Russia', 'Canada', 'Costa Rica', 'China', 'India'],
      dtype=object),
 array(['Bassoon', 'Piano', 'Theremin', 'Oboe'], dtype=object))
```

nom_5~9 are High Cardinality Features, if we directly encode them as onehot, this will result in a super sparse matrix(which will take more memory and longer time to train our model), for these features, we will use hash encoding

In [7]:

```
len(train['nom_5'].unique()), len(train['nom_6'].unique()), len(train['nom_7'].unique())
```

Out[7]:

```
(222, 522, 1220)
```

In [8]:

```
len(train['nom_8'].unique()), len(train['nom_9'].unique())
```

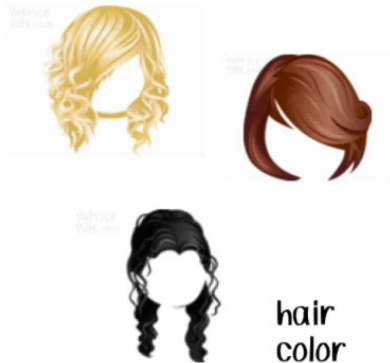
Out[8]:

```
(2215, 11981)
```

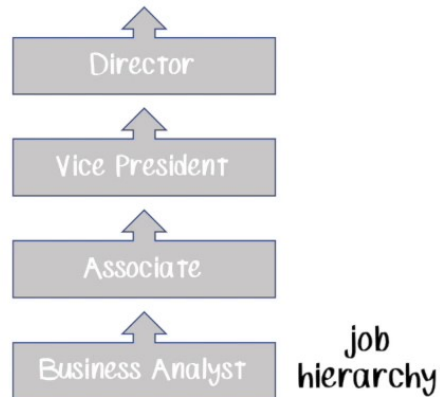
Ordinal Features

CATEGORICAL DATA

NOMINAL DATA



ORDINAL DATA



In [9]:

```
train.head().iloc[:, 16:-3]
```

Out[9]:

	ord_0	ord_1	ord_2	ord_3	ord_4	ord_5
0	2	Grandmaster	Cold	h	D	kr
1	1	Grandmaster	Hot	a	A	bF
2	1	Expert	Lava Hot	h	R	Jc
3	1	Grandmaster	Boiling Hot	i	D	kW
4	1	Grandmaster	Freezing	a	R	qP

In [10]:

```
train['ord_1'].unique()
```

Out[10]:

```
array(['Grandmaster', 'Expert', 'Novice', 'Contributor', 'Master'],  
      dtype=object)
```

In [11]:

```
train['ord_5'].unique()
```

Out[11]:

```
array(['kr', 'bF', 'Jc', 'kW', 'qP', 'PZ', 'wy', 'Ed', 'qo', 'CZ', 'qX',  
      'su', 'dP', 'aP', 'MV', 'oC', 'RL', 'fh', 'gJ', 'HJ', 'TR', 'CL',  
      'Sc', 'eQ', 'kC', 'qK', 'dh', 'gM', 'Jf', 'fO', 'Eg', 'KZ', 'Vx',  
      'Fo', 'sV', 'eb', 'YC', 'RG', 'Ye', 'qA', 'lL', 'Qh', 'Bd', 'be',  
      'hT', 'lF', 'nX', 'kK', 'av', 'uS', 'Jt', 'PA', 'Er', 'Qb', 'od',  
      'ut', 'Dx', 'Xi', 'on', 'Dc', 'sD', 'rZ', 'Uu', 'sn', 'yc', 'Gb',  
      'Kq', 'dQ', 'hp', 'kL', 'je', 'CU', 'Fd', 'PQ', 'Bn', 'ex', 'hh',  
      'ac', 'rp', 'dE', 'oG', 'oK', 'cp', 'mm', 'vK', 'ek', 'dO', 'XI',  
      'CM', 'Vf', 'aO', 'qv', 'jp', 'Zq', 'Qo', 'DN', 'TZ', 'ke', 'cG',  
      'tP', 'ud', 'tv', 'aM', 'xy', 'lx', 'To', 'uy', 'ZS', 'vy', 'ZR',  
      'AP', 'GJ', 'Wv', 'ri', 'qw', 'Xh', 'FI', 'nh', 'KR', 'dB', 'BE',  
      'Bb', 'mc', 'MC', 'tM', 'NV', 'ih', 'IK', 'Ob', 'RP', 'dN', 'us',  
      'dZ', 'yN', 'Nf', 'QM', 'jV', 'sY', 'wu', 'SB', 'UO', 'Mx', 'JX',  
      'Ry', 'Uk', 'uJ', 'LE', 'ps', 'kE', 'MO', 'kw', 'yY', 'zU', 'bJ',  
      'Kf', 'ck', 'mb', 'Os', 'Ps', 'Ml', 'Ai', 'Wc', 'GD', 'll', 'aF',  
      'iT', 'cA', 'WE', 'Gx', 'Nk', 'OR', 'Rm', 'BA', 'eG', 'cW', 'jS',  
      'DH', 'hL', 'Mf', 'Yb', 'Aj', 'oH', 'Zc', 'qJ', 'eg', 'xP', 'vq',  
      'Id', 'pa', 'ux', 'kU', 'Cl'], dtype=object)
```

Cyclic Features

In [12]:

```
train.head().iloc[:, -3:-1]
```

Out[12]:

	day	month
0	2	2
1	7	8
2	7	2
3	2	1
4	7	8

In [13]:

```
sorted(train.day.unique())
```

Out[13]:

```
[1, 2, 3, 4, 5, 6, 7]
```

In [14]:

```
sorted(train.month.unique())
```

Out[14]:

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

In [15]:

```
def cyclical_encode(data, col, max_val):
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    return data
```

Apply data transform

Binary Features

In [16]:

```
train['bin_3'] = train['bin_3'].apply(lambda x: 0 if x == 'F' else 1)
train['bin_4'] = train['bin_4'].apply(lambda x: 0 if x == 'N' else 1)

test['bin_3'] = test['bin_3'].apply(lambda x: 0 if x == 'F' else 1)
test['bin_4'] = test['bin_4'].apply(lambda x: 0 if x == 'N' else 1)
```

In [17]:

```
train.head().iloc[:, 1:6]
```

Out[17]:

	bin_0	bin_1	bin_2	bin_3	bin_4
0	0	0	0	1	1
1	0	1	0	1	1
2	0	0	0	0	1
3	0	1	0	0	1
4	0	0	0	0	0

Nominal Features

In [18]:

```
one_hot_train = pd.get_dummies(train[['nom_0', 'nom_1', 'nom_2', 'nom_3', 'nom_4']], drop_first = True)

one_hot_test = pd.get_dummies(test[['nom_0', 'nom_1', 'nom_2', 'nom_3', 'nom_4']], drop_first = True)

'''Dropping the variables'''
train.drop(['nom_0', 'nom_1', 'nom_2', 'nom_3', 'nom_4'], axis=1, inplace=True)
test.drop(['nom_0', 'nom_1', 'nom_2', 'nom_3', 'nom_4'], axis=1, inplace=True)
```

In [19]:

```
train = pd.concat([one_hot_train, train], axis = 1)
test = pd.concat([one_hot_test, test], axis = 1)
```

In [20]:

```
train.head().iloc[:, :20]
```

Out[20]:

	nom_0_Green	nom_0_Red	nom_1_Polygon	nom_1_Square	nom_1_Star	nom_1_Trapezoid
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	0	0	0	0	0	0
3	0	1	0	0	0	0
4	0	1	0	0	0	0

Feature Hashing

In [21]:

```
high_card = ['nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9']
for col in high_card:
    train[f'hash_{col}'] = train[col].apply( lambda x: hash(str(x)) % 5000 )
    test[f'hash_{col}'] = test[col].apply( lambda x: hash(str(x)) % 5000 )
```

In [22]:

```
train.drop(['nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'], axis=1, inplace=True)
test.drop(['nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'], axis=1, inplace=True)
```

In [23]:

```
train.head().iloc[:, -5:]
```

Out[23]:

	hash_nom_5	hash_nom_6	hash_nom_7	hash_nom_8	hash_nom_9
0	4343	2620	3593	2645	3085
1	3099	2086	3862	4352	1796
2	4921	2309	4941	2771	4965
3	2895	723	1646	1694	1803
4	3517	1369	3183	694	2598

Ordinal Features

In [24]:

```
"""let's begin the manual process of ordinal encoding."""
# ordinal encoding on train data
train.ord_1.replace(to_replace = ['Novice', 'Contributor', 'Expert', 'Master', 'Grandmaster'],
                    value = [0, 1, 2, 3, 4], inplace = True)

train.ord_2.replace(to_replace = ['Freezing', 'Cold', 'Warm', 'Hot', 'Boiling Hot', 'Lava Hot'],
                    value = [0, 1, 2, 3, 4, 5], inplace = True)

train.ord_3.replace(to_replace = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
                                   'n', 'o'],
                    value = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], inplace = True)

train.ord_4.replace(to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
                                   'N', 'O',
                                   'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'],
                    value = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
                              18, 19, 20, 21,
                              22, 23, 24, 25], inplace = True)

# ordinal encoding on test data
test.ord_1.replace(to_replace = ['Novice', 'Contributor', 'Expert', 'Master', 'Grandmaster'],
                  value = [0, 1, 2, 3, 4], inplace = True)

test.ord_2.replace(to_replace = ['Freezing', 'Cold', 'Warm', 'Hot', 'Boiling Hot', 'Lava Hot'],
                  value = [0, 1, 2, 3, 4, 5], inplace = True)

test.ord_3.replace(to_replace = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
                                   'n', 'o'],
                  value = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], inplace = True)

test.ord_4.replace(to_replace = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
                                   'N', 'O',
                                   'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'],
                  value = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
                            18, 19, 20, 21,
                            22, 23, 24, 25], inplace = True)
```

In [25]:

```
train.head().iloc[:, 26:32]
```

Out[25]:

	ord_0	ord_1	ord_2	ord_3	ord_4	ord_5
0	2	4	1	7	3	kr
1	1	4	3	0	0	bF
2	1	2	5	7	17	Jc
3	1	4	4	8	3	kW
4	1	4	0	0	17	qP

In [26]:

```
len(train.ord_5.unique())
```

Out[26]:

192

In [27]:

```
'''Simply sort their values by string of ord 5'''
# Source:- https://www.kaggle.com/c/cat-in-the-dat/discussion/105702#latest-607652
ord_5 = sorted(list(set(train['ord_5'].values)))
ord_5 = dict(zip(ord_5, range(len(ord_5))))
train.loc[:, 'ord_5'] = train['ord_5'].apply(lambda x: ord_5[x]).astype(float)
test.loc[:, 'ord_5'] = test['ord_5'].apply(lambda x: ord_5[x]).astype(float)
```

In [28]:

```
train.head().iloc[:, 26:32]
```

Out[28]:

	ord_0	ord_1	ord_2	ord_3	ord_4	ord_5
0	2	4	1	7	3	136.0
1	1	4	3	0	0	93.0
2	1	2	5	7	17	31.0
3	1	4	4	8	3	134.0
4	1	4	0	0	17	158.0

Cyclic Features

In [29]:

```
def cyclical_encode(data, col, max_val):
    data[col + '_sin'] = np.sin(2 * np.pi * data[col]/max_val)
    data[col + '_cos'] = np.cos(2 * np.pi * data[col]/max_val)
    return data
```

In [30]:

```
train = cyclical_encode(train, 'day', 7)
test = cyclical_encode(test, 'day', 7)

train = cyclical_encode(train, 'month', 12)
test = cyclical_encode(test, 'month', 12)

'''Dropping the variables'''
train.drop(['day', 'month'], axis=1, inplace=True)
test.drop(['day', 'month'], axis=1, inplace=True)
```

In [31]:

```
train.head().iloc[:, -4:]
```

Out[31]:

	day_sin	day_cos	month_sin	month_cos
0	9.749279e-01	-0.222521	0.866025	0.500000
1	-2.449294e-16	1.000000	-0.866025	-0.500000
2	-2.449294e-16	1.000000	0.866025	0.500000
3	9.749279e-01	-0.222521	0.500000	0.866025
4	-2.449294e-16	1.000000	-0.866025	-0.500000

Final training data

In [32]:

```
X_train, y_train = train.drop(['id', 'target'], axis=1), train['target']
X_test = test.drop(['id'], axis=1)
```

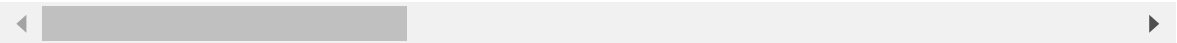
In [33]:

```
X_train.head()
```

Out[33]:

	nom_0_Green	nom_0_Red	nom_1_Polygon	nom_1_Square	nom_1_Star	nom_1_Trapezoid
0	1	0	0	0	0	0
1	1	0	0	0	0	0
2	0	0	0	0	0	0
3	0	1	0	0	0	0
4	0	1	0	0	0	0

5 rows × 40 columns



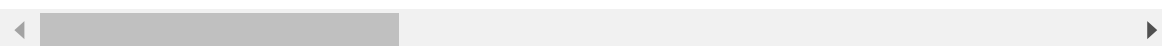
In [34]:

```
X_test.head()
```

Out[34]:

	nom_0_Green	nom_0_Red	nom_1_Polygon	nom_1_Square	nom_1_Star	nom_1_Trapezoid
0	0	0	0	0	0	(
1	0	1	0	1	0	(
2	0	0	0	1	0	(
3	0	1	0	0	1	(
4	0	1	0	0	0	.

5 rows × 40 columns



In [35]:

```
import numpy as np
```

In [36]:

```
X_train, y_train = np.array(X_train), np.array(y_train).astype(np.float)
```

Build model

Data preprocessing

In [37]:

```
# StandardScaler will do (X - X_mean) / X_std  
# MinMaxScaler will do (X - X_min) / (X_max - X_min)  
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

In [38]:

```
scaler = StandardScaler()
```



In [39]:

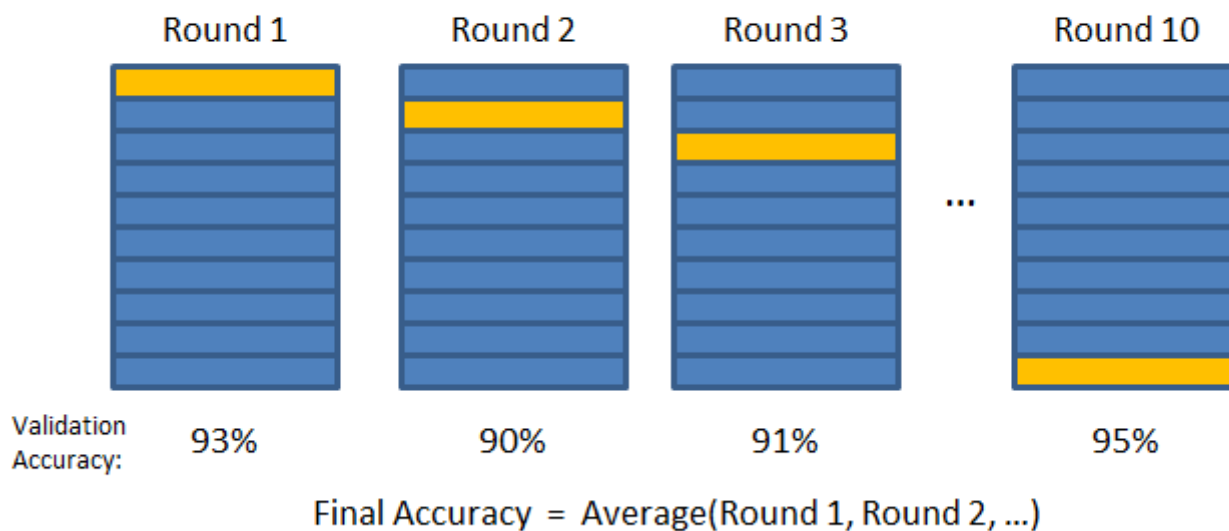
```
X_train = scaler.fit_transform(X_train)
```

KFold validation

In [40]:

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

 Validation Set
 Training Set



In [41]:

```
kf = KFold(n_splits=5)
```

In [42]:

```
fold_splits = kf.split(X_train, y_train)
```

In [43]:

```
fold_splits = list(fold_splits)
```

In [44]:

```
def get_model(X_train, y_train):
    model = LogisticRegression(solver='lbfgs')
    model.fit(X_train, y_train)
    return model
```

In [45]:

```
cv_models, cv_scores = [], []
for idx, (train_idx, val_idx) in enumerate(fold_splits):
    X_train_cv, X_val = X_train[train_idx], X_train[val_idx]
    y_train_cv, y_val = y_train[train_idx], y_train[val_idx]

    model = get_model(X_train_cv, y_train_cv)

    pred_val = model.predict_proba(X_val)[:, 1]

    acc = ((pred_val > 0.5).astype(np.float) == y_val).mean()
    print("Cross validation [{}/{}] Acc: {}".format(idx+1, len(fold_splits), acc))
    cv_scores.append(acc)
    cv_models.append(model)
```

```
Cross validation [1/5] Acc: 0.7438833333333333
Cross validation [2/5] Acc: 0.7407
Cross validation [3/5] Acc: 0.7468166666666667
Cross validation [4/5] Acc: 0.7411666666666666
Cross validation [5/5] Acc: 0.7448
```

In [53]:

```
np.mean(cv_scores), np.std(cv_scores)
```

Out[53]:

```
(0.7434733333333333, 0.0022855244960888633)
```

In [47]:

```
X_test.shape
```

Out[47]:

```
(200000, 40)
```

In [48]:

```
X_test = np.array(X_test)
```

In [49]:

```
X_test = scaler.transform(X_test)
```

In [50]:

```
pred_test = 0
for model in cv_models:
    pred = model.predict_proba(X_test)[:, 1]
    pred_test += pred
pred_test /= len(cv_models)
```

In [51]:

```
pred_test.shape
```

Out[51]:

```
(200000,)
```

In [52]:

```
submission = pd.DataFrame({'id': test.id, 'target': pred_test})  
submission.to_csv('submission.csv', index=False)
```

In []: