x20122136_ResearchProject_HKDataset_Part_2

Table of Contents

1. Objective

In this notebook applied deep learning algorithm on HK dataset to extend its accuracy using SVM, Randomforest, Naive Bayes, KNN, Decision tree, XGBoost algorithms.

2. Import packages

Here, we import common packages for Machine Learning.

In [ ]:

```
!pip install pandas
!pip install numpy
!pip install tensorflow
!pip install scikit-learn
!pip install matplotlib
!pip install pandas-profiling
!pip install xgboost
!pip install hyperopt
```

In [ ]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
import sklearn.preprocessing as preprocessing
import sklearn.model_selection as model_selection
import matplotlib.pyplot as plt
import xgboost as xgb
from pandas_profiling import ProfileReport
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
```

3. Load CSV's

In [ ]:

```
races_df = pd.read_csv(r"races.csv", delimiter=",", header=0, index_col='race_id')
runs_df = pd.read_csv(r"runs.csv", delimiter=",", header=0)
print(races_df.shape)
print(runs_df.shape)
```

4. EDA

4.1. EDA report on reaces data

In [ ]:

```
# races_profile = ProfileReport(races_df, title="Pandas Profiling Races Report")

# races_profile.to_file("eda_races.html")

# races_profile
```

4.2. EDA report on runs data

In [ ]:

```
# runs_profile = ProfileReport(runs_df, title="Pandas Profiling Runs Report")

# runs_profile.to_file("eda_runs.html")

# runs_profile
```

5. Data Pre-Processing

5.1. Prepare races data from races.csv

Only select several, then use different encoders for different types of attribute.

In [ ]:

```
races_df = races_df[['venue', 'config', 'surface', 'distance', 'going', 'race_class']]

# check to see if we have NaN, then drop NaN
print(races_df.isnull().sum())
print(races_df[races_df.isnull().any(axis=1)])
races_df = races_df.dropna()


print(races_df.head())
```

In [ ]:

```
# encode ordinal columns: config, going,
config_encoder = preprocessing.OrdinalEncoder()
races_df['config'] = config_encoder.fit_transform(races_df['config'].values.reshape(-1, 1
))
going_encoder = preprocessing.OrdinalEncoder()
races_df['going'] = going_encoder.fit_transform(races_df['going'].values.reshape(-1, 1))
```

In [ ]:

```
# encode nominal column: venue
venue_encoder = preprocessing.LabelEncoder()
races_df['venue'] = venue_encoder.fit_transform(races_df['venue'])

print(races_df.dtypes)
print(races_df.shape)
print(races_df.head())
```

5.2. Prepare races data from runs.csv

Similar to races data, only select columns that are relevant to the model.

In [ ]:

```
runs_df = runs_df[['race_id', 'draw',
                   'horse_age', 'horse_country', 'horse_type', 'horse_rating', 'declared
_weight', 'actual_weight', 'win_odds',
                   'result']]

# check to see if we have NaN, then drop NaN
print(runs_df[runs_df.isnull().any(axis=1)])
runs_df = runs_df.dropna()


# not sure why, but we got some strange draw in the dataset. Maximum shall be 14
strange_draw_index = runs_df[runs_df['draw'] > 14].index
# delete these row indexes from dataFrame
runs_df = runs_df.drop(strange_draw_index)
```

In [ ]:

```
# encode nominal columns: horse_country, horse_type
horse_country_encoder = preprocessing.LabelEncoder()
runs_df['horse_country'] = horse_country_encoder.fit_transform(runs_df['horse_country'])
horse_type_encoder = preprocessing.LabelEncoder()
runs_df['horse_type'] = horse_type_encoder.fit_transform(runs_df['horse_type'])

print(runs_df.dtypes)
print(runs_df.shape)
print('Label encoded Dataframe:', runs_df.head())
```

5.3. Further preprocessing for runs data

In [ ]:

```
def group_horse_and_result(element):
    if element[0] == 'result':
        return 100 + element[1] # to make sure results are put near the end
    else:
        return element[1]

runs_df = runs_df.pivot(index='race_id', columns='draw', values=runs_df.columns[2:])
rearranged_columns = sorted(list(runs_df.columns.values), key=group_horse_and_result)
runs_df = runs_df[rearranged_columns]

# quite some NaNs appreared in the dataframe, reason is some races didnt have full 14 hor
ses participating
# fill with 0
runs_df = runs_df.fillna(0)

print('After pivot, all the 14 horses',runs_df.head())
```

6. Prepare training and test data

Here, we combine races data and runs data by `join` two data frames above.

Split data into train/test sets

sklearn comes with such a handy method `train_test_split`. We split the data as following:

- **70% for training**
- **30% for testing(validation)**

In [ ]:

```
data = races_df.join(runs_df, on='race_id', how='right')
X = data[data.columns[:-14]]
ss = preprocessing.StandardScaler()
X = pd.DataFrame(ss.fit_transform(X),columns = X.columns)

y_won = data[data.columns[-14:]].applymap(lambda x: 1.0 if 0.5 < x < 1.5 else 0.0)

y_won.columns = ['1','2','3','4','5','6','7','8','9','10','11','12','13','14']
y_won.iloc[1]
y = []
for i in range(y_won.shape[0]):
    for n,j in enumerate(y_won.iloc[i]):
        if j==1:
            y.append(n+1)
            break

print(X.shape)
print(y_won.shape)

# split data into train and test sets
y_won2 = pd.Series(y)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y_won2, train_siz
e=0.7, test_size=0.3, random_state=1)

print("For machine learning:")
print('X_train', X_train.shape)
print('y_train', y_train.shape)
print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

7. SVM

In [ ]:

```
model = SVC()
```

In [ ]:

```
model.fit(X_train, y_train)
```

In [ ]:

```
print('Base SVM model: ', model.score(X_test, y_test))
```

7.1 Tune Parameter

1. Regularization (C)

In [ ]:

```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
print('Regularization SVM model C=1: ', model_C.score(X_test, y_test))
```

In [ ]:

```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
print('Regularization SVM model C=10: ', model_C.score(X_test, y_test))
```

2. Gamma

In [ ]:

```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
print('Gamma SVM model gamma=10: ', model_g.score(X_test, y_test))
```

3. Kernel

In [ ]:

```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
```

In [ ]:

```
yPredict = model_linear_kernal.predict(X_test)
```

In [ ]:

```
print('Kernel SVM model linear ', model_linear_kernal.score(X_test, y_test))
```

In [ ]:

```
print(classification_report(y_test, yPredict))
```

In [ ]:

```
model_rbf_kernal = SVC(kernel='rbf')
model_rbf_kernal.fit(X_train, y_train)
print('Kernel SVM model rbf ', model_rbf_kernal.score(X_test, y_test))
```

8. Random Forest

In [ ]:

```
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)
```

In [ ]:

```
print('Random forest base model: ', model.score(X_test, y_test))
```

In [ ]:

```
y_predicted = model.predict(X_test)
```

Confusion Matrix

In [ ]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

8.1 Tune Parameter

In [ ]:

```
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

In [ ]:

```
print('Random forest model n-estimater = 100 : ', model.score(X_test, y_test))
```

In [ ]:

```
scores = []

params = {'bootstrap': [True, False],
 'max_depth': [10,50], # 100
#  'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2],
 'min_samples_split': [2, 5],
 'n_estimators': [100,500]} # 1000, 2000

clf =  GridSearchCV(RandomForestClassifier(), param_grid=params, cv=3, return_train_scor
e=False)

clf.fit(X_train, y_train)

clf.best_params_,clf.best_score_
```

In [ ]:

```
y_predicted = model.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicted)
cm
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
```

```
plt.figure(figsize=(10,7))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

In [ ]:

```
print(classification_report(y_test, y_predicted))
```

9. Naive Bayes

In [ ]:

```
model = GaussianNB()
```

In [ ]:

```
naiveModel = model.fit(X_train,y_train)
```

In [ ]:

```
y_predicted = naiveModel.predict(X_test)
```

In [ ]:

```
naiveModel.score(X_test,y_test)
```

In [ ]:

```
naiveModel.predict_proba(X_test)
```

9.1 Tune Parameter

Calculate the score using cross validation

In [ ]:

```
cross_val_score(GaussianNB(),X_train, y_train, cv=5)
```

In [ ]:

```
print(classification_report(y_test, y_predicted))
```

10. KNN

In [ ]:

```
knn = KNeighborsClassifier(n_neighbors=10)
```

In [ ]:

```
knn.fit(X_train, y_train)
```

In [ ]:

```
print('KNN with n_neighbors=10 :', knn.score(X_test, y_test))
```

Confusion Matrix

In [ ]:

```
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

In [ ]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sn
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Print classification report for precesion, recall and f1-score for each classes

In [ ]:

```
print(classification_report(y_test, y_pred))
```

11. Decision Tree

In [ ]:

```
model = tree.DecisionTreeClassifier()
```

In [ ]:

```
model = model.fit(X_train, y_train)
```

In [ ]:

```
model.score(X_test, y_test)
```

In [ ]:

```
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

12. XGBoost

In [ ]:

```
space={'max_depth': hp.quniform("max_depth", 3, 18, 1),
        'gamma': hp.uniform ('gamma', 1,9),
        'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
        'reg_lambda' : hp.uniform('reg_lambda', 0,1),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
        'n_estimators': 180,
        'seed': 0
    }
```

In [ ]:

```
def objective(space):
    clf=xgb.XGBClassifier(
                    n_estimators =space['n_estimators'], max_depth = int(space['max_dept
h']), gamma = space['gamma'],
                    reg_alpha = int(space['reg_alpha']),min_child_weight=int(space['min_
child_weight']),
                    colsample_bytree=int(space['colsample_bytree']))

    evaluation = [( X_train, y_train), ( X_test, y_test)]
```

```python
    clf.fit(X_train, y_train,
            eval_set=evaluation, eval_metric="auc",
            early_stopping_rounds=10,verbose=False)


    pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, pred>0.5)
    print ("SCORE:", accuracy)
    return {'loss': -accuracy, 'status': STATUS_OK }
```

In [ ]:

```python
trials = Trials()

best_hyperparams = fmin(fn = objective,
                        space = space,
                        algo = tpe.suggest,
                        max_evals = 100,
                        trials = trials)
```

In [ ]:

```python
print("The best hyperparameters are : ","\n")
print(best_hyperparams)
```

In [ ]:

```python

```