Klasifikasi dengan Scikit-Learn

Des 2022

Yudi Wibisono (yudi@upi.edu)
Masayu L. Khodra (masayu@informatika.org; masayu@stei.itb.ac.id)

Modul Praktikum

Daftar Isi

Slide terkait klasifikasi:
https://docs.google.com/presentation/d/1FadP836Rde8Zb63nEQtTnc1sl__JVhQWNSwVPbxF91c/edit?usp=sharing

1. Pendahuluan

Sebelum mulai pembaca diharapkan sudah mencoba modul tentang praproses/EDA:
https://docs.google.com/document/d/1ehUlhdLeubEJz9qc3fvGeCRtZindHhowyGhg5Pbqq3w/edit?usp=sharing

Pembaca juga diharapkan telah melihat slide tentang klasifikasi:
https://docs.google.com/presentation/d/1FadP836Rde8Zb63nEQtTnc1sl__JVhQWNSwVPbxF91c/edit?usp=sharing

Klasifikasi adalah bagian dari task machine learning yang paling umum. Pada klasifikasi, algoritma belajar dari data. Sebagai contoh, klasifikasi apakah seseorang akan mendapat pinjaman atau tidak (label YA/TIDAK) berdasarkan berbagai data pribadi. Algoritma mengolah data pribadi yang memiliki label mendapat pinjaman atau tidak. Dari data tersebut algoritma menghasilkan model yang dapat memprediksi label (YA/TIDAK) jika diberi data pribadi.

Modul ini menggunakan library scikit-learn untuk membuat model klasifikasi.

scikit-learn

Library ini dapat digunakan untuk melakukan berbagai task dalam machine learning, termasuk klasifikasi. Library ini dibangun di atas library SciPy (library untuk matematika, sains dan rekayasa). Selain menyediakan berbagai algoritma untuk klasifikasi, regresi dan clustering, scikit learn juga menyediakan fasilitas untuk evaluasi model, tune parameter dan lainnya.

2. Klasifikasi Sederhana

Kita akan menggunakan data pada dataset dari modul praproses sebelumnya. Tujuan kita adalah membuat model yang memprediksi apakah atlet akan mendapatkan jenis medali berdasarkan data-data peserta olimpiade selama 120 tahun.

Download data di:

120-years-of-olympic-history-athletes-and-results.zip

atau http://bit.ly/yw_dataset_olimpiade

2.1 Persiapan Dataset

Eksplorasi (EDA) dan praproses untuk dataset ini dibahas rinci pada modul praproses

Pastikan package scikit-learn sudah terinstall

Load lagi dataset dari awal

```
df = pd.read_csv("../data/athlete_events.csv")
df.info()
```

Isi dari dataset ini terdiri dari 14 kolom dengan sekitar 270ribu baris.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271116 entries, 0 to 271115
Data columns (total 15 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   ID      271116 non-null  int64  
 1   Name    271116 non-null  object 
 2   Sex     271116 non-null  object 
 3   Age     261642 non-null  float64
 4   Height  210945 non-null  float64
 5   Weight  208241 non-null  float64
 6   Team    271116 non-null  object 
 7   NOC     271116 non-null  object 
 8   Games   271116 non-null  object 
 9   Year    271116 non-null  int64  
 10  Season  271116 non-null  object 
 11  City    271116 non-null  object 
 12  Sport   271116 non-null  object 
 13  Event   271116 non-null  object 
 14  Medal   39783 non-null   object 
dtypes: float64(3), int64(2), object(10)
memory usage: 31.0+ MB
```

Dengan contoh data sebagai berikut:

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	Sport	Event	Medal
0	1	A Dijiang	M	24.0	180.0	80.0	China	CHN	1992 Summer	1992	Summer	Barcelona	Basketball	Basketball Men's Basketball	NaN
1	2	A Lamusi	M	23.0	170.0	60.0	China	CHN	2012 Summer	2012	Summer	London	Judo	Judo Men's Extra-Lightweight	NaN
2	3	Gunnar Nielsen Aaby	M	24.0	NaN	NaN	Denmark	DEN	1920 Summer	1920	Summer	Antwerpen	Football	Football Men's Football	NaN
3	4	Edgar Lindenau Aabye	M	34.0	NaN	NaN	Denmark/Sweden	DEN	1900 Summer	1900	Summer	Paris	Tug-Of-War	Tug-Of-War Men's Tug-Of-War	Gold

Pembuangan Instances
Karena yang kita inginkan adalah memprediksi olahraga di waktu modern, maka kita akan ambil data dari tahun 1970 ke atas.

```
df = df[df.Year>1970]
```

Jumlah baris turun dari 270 ribuan ke 179 ribuan.

Penanganan Data Kosong

Dapat dilihat ada beberapa data yang kosong. Ada algoritma klasifikasi yang tidak memperbolehkan nilai kosong, kita tangani dengan mengisi data yang kosong dengan mediannya.

```
df['Height'].fillna(df['Height'].median(),inplace=True)
df['Age'].fillna(df['Age'].median(),inplace=True)
df['Weight'].fillna(df['Weight'].median(),inplace=True)
```

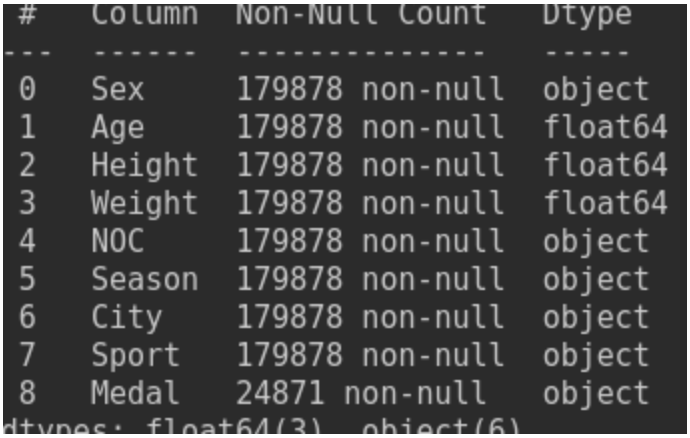
Pembuangan Atribut dan Instance

Dalam task klasifikasi ini, targetnya adalah memprediksi medali yang diterima suatu atlet, jadi ada atribut yang tidak diperlukan dan bisa dibuang. Ini pentingnya memahami dataset. Sebagai contoh ID dan nama atlet, bisa disimpulkan tidak ada hubungan antara nama atlet dengan medali yang diperolehnya. Dalam modul praproses sudah diperlihatkan bahwa Team dapat diganti dengan NOC. Karena games dan tahun sudah terlewat dan model klasifikasi ini digunakan untuk memprediksi kejadian di masa depan, maka atribut Games dan Year juga dapat dihilangkan. Event dapat dilihat sebagai rincian cabang olahraga, untuk tahap sekarang asumsikan kita hanya perlu Sport saja.

```
df = df.drop(['ID','Name','Team','Games','Year','Event'],axis=1)
```

Catatan: Hati-hati dalam membuang atribut! Dapat saja atribut yang tampaknya tidak penting ternyata memiliki pengaruh. Sebagai contoh City, ada beberapa kota yang terletak di dataran tinggi yang mungkin mempengaruhi kinerja atlet.

Hasilnya



One-hot encoding untuk tipe kategori

Tahap selanjutnya adalah mengkonversi atribut kategori menjadi format one-hot encoding. Pada one-hot encoding, gender misalnya akan menjadi dua atribut yaitu sex_female dan sex_male yang nilainya 0 atau 1.

Mengapa tidak dikodekan misalnya Male=0 dan F=1? Masalahnya, beberapa algoritma akan menganggap 0 dan 1 sebagai nilai numerik, bukan kategori. Apakah arti jika atribut gender bernilai 0.3? Apakah instance F (bernilai 1) bernilai lebih tinggi daripada M (bernilai 0)? Ini sebabnya diperlukan one-hot encoding.

Beberapa algoritma tree seperti Decision Tree dan Random Forest dapat memproses data bertipe kategori secara langsung, tetapi lebih aman jika atribut kategori, terutama yang bukan ordinal dijadikan one-hot. Kategori jenis ordinal yang memiliki keterurutan seperti pangkat, dapat dikonversi sebagai angka. Misalnya SD:1, SMP:2, SMA: 3, PT: 4.

Untuk mengubah suatu atribut menjadi one-hot, gunakan method get_dummies() dari pandas. Kode berikut mengubah atribut gender

```
df = pd.get_dummies(data=df, columns=['Sex'])
```

Hasilnya adalah atribut gender akan dihapus dan digantikan oleh dua atribut baru yaitu Sex_F dan Sex_M

```
 9   Sex_F   179878 non-null  uint8
 10  Sex_M   179878 non-null  uint8
```

Jika lihat contoh instance-nya, atribut ini hanya bernilai 0 atau 1, inilah yang disebut one-hot encoding. Hanya satu atribut yang akan bernilai satu dan sisanya nol.

	Name	Sex_F	Sex_M
0	A Dijiang	0	1
1	A Lamusi	0	1
2	Gunnar Nielsen Aaby	0	1
3	Edgar Lindenau Aabye	0	1
4	Christine Jacoba Aaftink	1	0
10	Per Knut Aaland	0	1
18	John Aalberg	0	1
26	Cornelia "Cor" Aalten (-Strannood)	1	0
28	Antti Sami Aalto	0	1
29	Einar Ferdinand "Einari" Aalto	0	1

Kita ubah beberapa atribut kategori lain menjadi hot encoding:

```
df = pd.get_dummies(data=df, columns=['NOC','Season','City','Sport'])
```

Menggunakan df.shape terlihat ukuran atribut bertambah menjadi 302.

Jika suatu atribut memiliki 50 kemungkinan nilai, maka jumlah kolom yang akan ditambahkan juga mencapai 50. Ini kelemahan dari format one-hot-encoding yang dapat menghasilkan jumlah atribut yang besar, walaupun nanti dapat direduksi dengan teknik seperti PCA.

Kelas Target

Kelas target (atribut yang akan diprediksi) untuk dataset ini adalah medali (Medal).

Ubah peserta yang tidak mendapat medali dari NaN menjadi "None"

```
df['Medal'].fillna("None", inplace=True)
```

Coba kita lihat distribusinya.

```
None      155007
Bronze      8511
Gold        8224
Silver      8136
Name: Medal, dtype: int64
```

Jumlah yang tidak mendapat medali sangat besar dibandingkan medali yang lain.

Selanjutnya ubah atribut yang akan menjadi label menggunakan LabelEncoder. LabelEncoder akan mengubah string "Gold", "Bronze" dst menjadi angka 0 sampai dengan 3.

Variabel Y akan berisi kelas yang sudah dikonversi menjadi angka 0..3

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(df.Medal)
Y = le.transform(df.Medal)
```

Coba lihat isi Y

Untuk mendapatkan kelas dari LabelEncoder gunakan `list(le.classes_)` maka hasilnya: `['Bronze', 'Gold', 'None', 'Silver']`. Untuk mengkonversi kembali dari indeks ke string kelas, gunakan `list(le.inverse_transform([2, 0, 1, 3]))` yang akan menghasilkan `['None', 'Bronze', 'Gold', 'Silver']`

Siapkan atribut training dengan membuang kelas label (kelas label sudah dipindahkan ke variabel Y sebelumnya)

```
X = df.drop("Medal",axis=1)
```

Sekarang sudah ada X berisi atribut yang akan menjadi feature dan Y berisi label yang akan dibuat model prediksinya.

Split Dataset

Untuk menyiapkan data train dan test, cara pertama adalah dengan teknik hold out. Secara random dipilih 70%-80% instance menjadi data train dan sisanya (30%-20%) menjadi data test. Menggunakan library train_test_split, code berikut mengambil 80% instance sebagai data train secara random dan 20% sisanya sebagai data test.

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=123)

#simpan nama kolom untuk keperluan prediksi nanti
import pickle
with open('C:\\data\\olympic_x_train_columns.pickle', 'wb') as fp:
    pickle.dump(X_train.columns, fp)
```

2.2 Pembelajaran (*Learning*) & Evaluasi

Selanjutnya pembuatan model sudah dapat dilakukan. Naive Bayes sering digunakan sebagai teknik baseline.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
clf = GaussianNB()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
print("Akurasi {}".format(acc))
print(classification_report(Y_test, Y_pred))
```

Hasilnya masih buruk

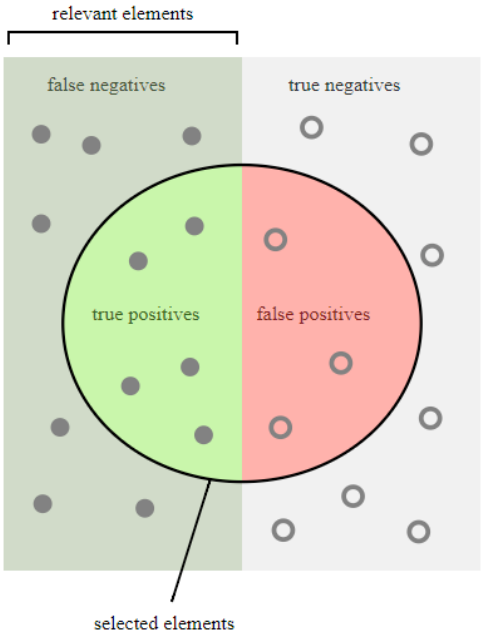
```
Akurasi 0.13700800533689125
              precision    recall  f1-score   support

           0       0.04      0.07      0.05      1676
           1       0.06      0.95      0.11      1641
           2       0.99      0.10      0.19     31033
           3       0.04      0.04      0.04      1626

    accuracy                           0.14     35976
   macro avg       0.28      0.29      0.09     35976
weighted avg       0.86      0.14      0.17     35976
```

Menggunakan list(le.inverse_transform([0, 1, 2, 3])), 0 adalah Bronze, 1:Gold, 2: None, 3: Silver. "None" (tidak mendapat medali) tinggi precision-nya karena jumlahnya besar.

Catatan: precision, recall dan F1 (F-Measure) adalah ukuran yang lebih tepat jika jumlah tiap kelas tidak seimbang

	Prediction		
Truth	+	-	Total
+	*tp*	*fn*	*p*
-	*fp*	*tn*	*n*
Total	p'	n'	*N*

```
Akurasi = (tp+tn)/N
Presisi = tp/p'
Recall  = tp/p
F1 = 2*Presisi*Recall / (Presisi+Recall)
```



How many selected items are relevant?

Precision =

How many relevant items are selected?

Recall =

Jika diganti dengan decision tree, hasilnya membaik, walaupun untuk kelas minoritas (Gold, Silver, Bronze) nilai F1 masih jauh lebih rendah dibandingkan kelas mayoritas (None).

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
print("Akurasi {}".format(acc))
print(classification_report(Y_test, Y_pred))
```

```
Akurasi 0.8517066933511229
              precision    recall  f1-score   support

           0       0.33      0.37      0.35      1676
           1       0.43      0.47      0.45      1641
           2       0.93      0.92      0.93     31033
           3       0.42      0.37      0.40      1626

    accuracy                           0.85     35976
   macro avg       0.53      0.53      0.53     35976
weighted avg       0.85      0.85      0.85     35976
```

Jika menggunakan RandomForest, hasilnya lebih baik lagi. Untuk mendapatkan hasil lebih tinggi, parameter n_estimator dapat dinaikkan tetapi proses training akan semakin lambat.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50, random_state=123)
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
print("Akurasi {}".format(acc))
print(classification_report(Y_test, Y_pred))
```

```
Akurasi 0.88289415165666
              precision    recall  f1-score   support

           0       0.52      0.28      0.37      1676
           1       0.60      0.40      0.48      1641
           2       0.91      0.97      0.94     31033
           3       0.53      0.31      0.39      1626

    accuracy                           0.88     35976
   macro avg       0.64      0.49      0.54     35976
weighted avg       0.86      0.88      0.87     35976
```

Untuk mengetahui fitur mana yang paling penting bagi model, gunakan code berikut:

```
feature_importances = pd.DataFrame(clf.feature_importances_,index = X_train.columns,
columns=['importance']).sort_values('importance',ascending=False)

print(feature_importances)
```

```
            importance
Weight    1.420008e-01
Height    1.361687e-01
Age       1.337881e-01
NOC_USA   1.953652e-02
NOC_URS   1.757147e-02
...                ...
NOC_STP   6.085159e-07
NOC_YAR   4.239041e-07
NOC_SSD   2.232222e-07
NOC_YMD   1.735638e-07
NOC_VNM   1.891887e-08
```

XGBOOST

Untuk menggunakan XGBoost, install terlebih dulu library xgboost. Kelebihan dari XGBoost adalah dapat menangani data null.

Kinerja lebih baik tetapi prosesnya lebih lama (antara 10-30 menit).

```
import xgboost as xgb
clf = xgb.XGBClassifier(objective = "multi:softprob", num_class = 4, eval_metric =
"mlogloss", max_depth = 24, gamma=0.1, subsample = 0.90,learning_rate=0.01,
n_estimators = 10, nthread=-1)
# num_class adalah jumlah kelas
clf.fit(X_train, Y_train)
Y_pred = clf.predict(X_test)
acc = accuracy_score(Y_test, Y_pred)
print("Akurasi {}".format(acc))
print(classification_report(Y_test, Y_pred))
```

```
Akurasi 0.8934845452523905
              precision    recall  f1-score   support

           0       0.80      0.22      0.34      1676
           1       0.69      0.36      0.47      1641
           2       0.90      0.99      0.94     31033
           3       0.73      0.27      0.39      1626

    accuracy                           0.89     35976
   macro avg       0.78      0.46      0.54     35976
weighted avg       0.88      0.89      0.87     35976
```

Catatan: 0:Bronze, 1:Gold, 2: None, 3: Silver. "None" (tidak mendapat medali) tinggi precision-nya karena jumlahnya besar.

Untuk menampilkan confusion matrix, tambahkan code berikut setelah print classification report. Confusion matrix berisi rangkuman hasil prediksi, berguna untuk melihat kesalahan dalam proses

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
```

Hasilnya akan seperti ini:

```
[[  363    56  1226    31]
 [   22   590   979    50]
 [   46   151 30751    85]
 [   20    59  1107   440]]
```

Baris pertama adalah Bronze (0), baris kedua adalah Gold (1) dan seterusnya. Tabel berikut lebih memperjelas:

	0	**1**	**2**	**3** ⟵ **prediksi**
0: Bronze	[[363	56	1226	31]
1: Gold	[22	590	979	50]
2: None	[46	151	30751	85]
3: Silver	[20	59	1107	440]]

Idealnya, **jika semua prediksi tepat**, confusion matrix adalah sebagai berikut:

```
[[ 1676     0     0     0]
 [    0  1641     0     0]
 [    0     0 31033     0]
 [    0     0     0  1626]]
```

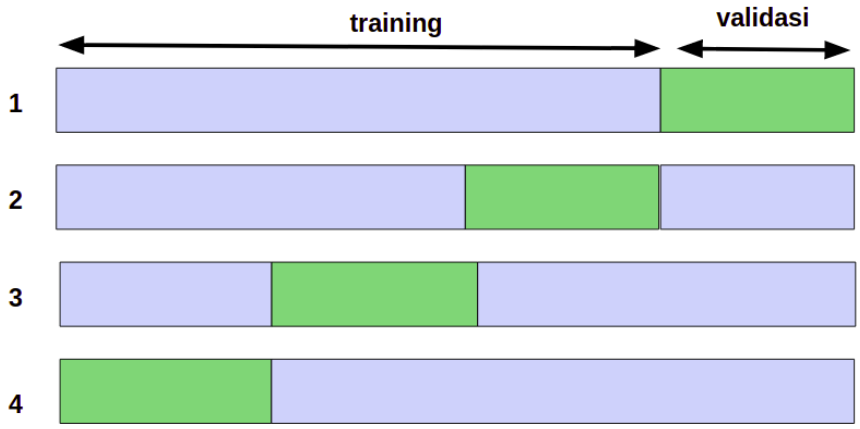
Berdasarkan confusion matrix hasil prediksi (gambar bawah), terlihat bahwa banyak prediksi yang "meleset" ke kelas 2 (None), ini disebabkan jumlah atlet yang masuk ke label 2 (None, tidak dapat medali), jauh lebih besar dibandingkan atlet yang mendapat medali.

	0	**1**	**2**	**3** ← **prediksi**
0: Bronze	[[363	56	**1226**	31]
1: Gold	[22	590	**979**	50]
2: None	[46	151	**30751**	85]
3: Silver	[20	59	**1107**	440]]

Latihan:
Bagaimanakah kinerja model jika one-hot tidak digunakan untuk tipe data kategori?

Cross Validation

Sebelumnya data train dan test dipilih secara random dengan rasio 80:20, kelemahan dari pendekatan ini adalah bisa saja data train dan test yang diambil kebetulan tidak ideal. Cara yang lebih baik adalah dengan menggunakan metode cross validation. Pada cross validation, semua data akan "digilir" menjadi data training dan data testing. Gambar berikut adalah contoh 4x cross validation.



Contoh implementasi 3x cross validation dengan sklearn adalah sebagai berikut (Decision Tree dipilih agar proses tidak terlalu lama):

```
from sklearn.model_selection import cross_val_score
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, X=X, y=Y, cv=3)
print(scores)
```

Hasilnya: [0.85475317 0.84918027 0.84689538]

Bagaimana jika kita ingin hasil kinerja yang lebih rinci? Dapat digunakan fungsi callback seperti code di bawah:

```
from sklearn.metrics import classification_report, accuracy_score, make_scorer
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

def classification_report_with_accuracy_score(y_true, y_pred):
    print(classification_report(y_true, y_pred))
    print(confusion_matrix(y_true, y_pred))
    return accuracy_score(y_true, y_pred)
clf = tree.DecisionTreeClassifier()
scores = cross_val_score(clf, X=X, y=Y, cv=3,

scoring=make_scorer(classification_report_with_accuracy_score))
print(scores)
```

```
             precision    recall  f1-score   support

          0       0.34      0.37      0.36      4432
          1       0.43      0.44      0.44      4458
          2       0.92      0.93      0.92     77111
          3       0.45      0.34      0.39      4372

avg / total       0.84      0.85      0.84     90373

[[ 1652   263  2295   222]
 [  363  1957  1902   236]
 [ 2487  1945 71335  1344]
 [  353   353  2169  1497]]
             precision    recall  f1-score   support

          0       0.33      0.37      0.35      4432
          1       0.43      0.44      0.44      4457
          2       0.92      0.92      0.92     77111
          3       0.44      0.33      0.37      4372

avg / total       0.84      0.84      0.84     90372

[[ 1646   285  2316   185]
 [  320  1967  1919   251]
 [ 2723  1924 71069  1395]
 [  315   378  2253  1426]]
             precision    recall  f1-score   support

          0       0.32      0.36      0.34      4431
          1       0.41      0.43      0.42      4457
          2       0.92      0.92      0.92     77111
          3       0.43      0.33      0.37      4372

avg / total       0.84      0.84      0.84     90371

[[ 1601   278  2371   181]
 [  351  1909  1922   275]
 [ 2738  2058 70849  1466]
 [  339   376  2227  1430]]
[0.84583891 0.8421635  0.83864293]
```

3. Perbaikan Kinerja

Jarang sekali saat pertama dijalankan kinerja model sesuai yang diharapkan. Ada empat sektor yang dapat digunakan untuk memperbaiki kinerja model (Jason, 2017).

1. Data
2. Algoritma
3. Tuning
4. Ensembles.

Tidak ada jaminan kinerja akan naik jika semua strategi dijalankan, bahkan dalam beberapa kasus dapat memperburuk. Eksperimen perlu dilakukan untuk melihat dampak usaha perbaikan kinerja model.

Terkait data, hal yang dapat dilakukan untuk meningkatkan kinerja adalah:

1. Menambah data. Teknik seperti deep learning memerlukan data dalam jumlah besar.
2. Resample, menggunakan resample untuk memperkecil jumlah data sehingga eksperimen dapat dipercepat, atau oversampling untuk memperbesar jumlah data (menambahkan data sintesis)
3. Normalisasi dan standarisasi atribut.
4. Transformasi data: mengubah distribusi data, membuat lebih gaussian, memasukkan ke fungsi eksponensial.
5. Proyeksi data ke dimensi yang lebih rendah.
6. Pemilihan atribut/fitur.
7. Kombinasi atau agregasi atribut menjadi atribut baru.

Terkait algoritma, strategi yang dapat digunakan:

1. Ganti algoritma klasifikasi. Saat in random forest dan XGBoost adalah algoritma yang kinerjanya paling tinggi untuk data terstruktur.
2. Jika diperlukan lakukan penyesuaian algoritma.
3. Metode pembentukan data test dan train. Dianjurkan menggunakan k-fold cross-validation ditambah hold-out untuk data validasi.
4. Mengganti jenis prediksi: regresi, binary/ multiclass, time series. Misalnya ternyata ada faktor urutan waktu, maka pendekatan time series lebih tepat.

Terkait tuning

1. Diagnostik, overfit atau underfit.
2. Random search untuk mencari hyperparameter.
3. Grid search, enumerasi setiap parameter.

Ensemble adalah mengkombinasikan beberapa model. Strategi yang dapat digunakan adalah:

1. Menggunakan beberapa model dari berbagai teknik untuk data yang sama. Lalu menggunakan modus atau mean untuk mengkombinasikan prediksinya.
2. Bagging: Mengkombinasikan beberapa model dengan representasi atau view **data yang berbeda**. RandomForest adalah teknik yang menggunakan pendekatan ini.
3. Boosting (Bootstrap aggregating): model dibuat secara berurutan. Data yang memberikan error diberi bobot lebih besar untuk model berikutnya. XGBoost adalah contoh teknik yang menggunakan boosting ini.

Berikut kita akan coba menggunakan dataset olimpiade.

3.1 Scale, Standarisasi & Normalisasi

Scale adalah mengubah range (biasanya menjadi 0 sampai dengan 1). Standarisasi mengubah nilai sehingga standar deviasi menjadi 1 dan normalisasi bisa berarti scale atau standarisasi.

Transformasi ini diperlukan karena beberapa algoritma akan memiliki kinerja lebih baik atau konvergen lebih cepat jika semua atribut memiliki skala yang sama atau terdistribusi normal. Contohnya linear & logistic regression, nearest neighbours, neural network, SVM.

MinMaxScaler

Nilai baru hasil dari transformasi MinMaxScaler dihitung dengan y = (x – min) / (max – min)

Jika Age, Height, Weight sebelumnya seperti gambar di bawah

Age	Height	Weight
25.0	165.0	47.0
24.0	170.0	58.0
26.0	189.0	83.0
19.0	175.0	70.0
22.0	179.0	72.0

Setelah ditransformasikan:

```
from sklearn import preprocessing
mm_scaler = preprocessing.MinMaxScaler()
X_train_minmax = mm_scaler.fit_transform(X_train)
X_test_minmax = mm_scaler.transform(X_test)
```

Hasilnya adalah sebagai berikut:

```
[[0.35       0.4375     0.17460317 ...
 [0.11666667 0.35416667 0.11640212 ...
 [0.16666667 0.46875    0.23809524 ...
 ...
 [0.28333333 0.32291667 0.20634921 ...
 [0.21666667 0.35416667 0.17460317 ...
 [0.38333333 0.44791667 0.22751323 ...
```

Setelah dicoba dengan dengan XGBoost ternyata tidak ada perubahan F1-score untuk setiap kelas.

StandardScaler

StandardScaler mengubah distribusi sehingga nilai rata-rata menjadi nol dan standar deviasi 1.

y = (x – mean) / standard_deviation

```
ss_scaler = preprocessing.StandardScaler()
X_train_ss = mm_scaler.fit_transform(X_train)
X_test_ss = mm_scaler.transform(X_test)
```

Catatan: fit hanya dilakukan pada data train (fit_transform adalah kombinasi fit lalu transform sehingga lebih cepat)

Setelah dicoba sama tidak ada perubahan F1-score untuk setiap kelas.

3.2 Feature Engineering

Feature engineering adalah praproses untuk memastikan input algoritma (atribut/feature) sudah tepat dan menghasilkan kinerja tertinggi. Fokus dari feature engineering adalah atribut. Beberapa hal yang dapat dilakukan:

Penambahan feature atau atribut baru
BMI yang mengkombinasikan antara berat badan dan tinggi badan

```
df["BMI"]  = df.Weight  / (df.Height/100 * df.Height/100)
```

Setelah dicoba hasilnya tidak meningkatkan kinerja yaitu F1-score untuk kelas emas, perak dan perunggu.

Bagaimana jika atribut Event yang tadinya dibuang tetap dimunculkan? (dijadikan tipe kategori dan one hot), Terjadi peningkatan kinerja yang signifikan. Ini menunjukkan event (cabang olahraga spesifik) punya peran penting dalam memprediksi medali yang akan diperoleh.

```
Akurasi 0.9022681787858572
              precision    recall  f1-score   support

           0       0.92      0.23      0.37      1676
           1       0.84      0.41      0.55      1641
           2       0.90      1.00      0.95     31033
           3       0.83      0.30      0.44      1626

    accuracy                           0.90     35976
   macro avg       0.87      0.48      0.58     35976
weighted avg       0.90      0.90      0.88     35976
```

Penanganan data null

Sebelumnya tinggi, berat dan umur diproses untuk data yang kosong, berbeda dengan RandomForest, library XGBoost dapat memproses data null. Kita coba jika null pada ketiga atribut itu dibiarkan, hasilnya:

```
Akurasi 0.901573271069602
              precision    recall  f1-score   support

           0       0.91      0.23      0.37      1676
           1       0.83      0.41      0.55      1641
           2       0.90      1.00      0.95     31033
           3       0.83      0.29      0.43      1626

    accuracy                           0.90     35976
   macro avg       0.87      0.48      0.57     35976
weighted avg       0.90      0.90      0.88     35976
```

Terlihat sama, tapi untuk kelas 3 (Medali Silver) terjadi penurunan kinerja, artinya penanganan null punya dampak positif terhadap kinerja dan diperlukan.

Latihan

Penanganan null untuk menghitung berat, tinggi dan umur masih kurang ideal karena hanya menggunakan rata-rata global tanpa memperhitungkan gender dan cabang olahraga.

Lakukan penanganan null dengan menggunakan rata-rata berdasarkan gender dan cabang olahraga, bandingkan hasilnya.

Kompresi ke dimensi yang lebih rendah
Setelah menggunakan one-hot encoding untuk atribut kategorik (termasuk Event), maka jumlah atribut menjadi besar (784 atribut). Reduksi dimensi dapat digunakan untuk mengatasi ini. Selain mempercepat proses training, untuk visualisasi, dan dalam beberapa kasus dapat meningkatkan kinerja model.

PCA (Principal Component Analysis) dapat digunakan untuk mereduksi dimensi.

Scaling perlu dilakukan terlebih dulu:

```
from sklearn.decomposition import PCA
ss_scaler = preprocessing.StandardScaler()
X_train_ss = ss_scaler.fit_transform(X_train) #fit hanya di train
X_test_ss = ss_scaler.transform(X_test)
pca = PCA(.75)
X_train_pca = pca.fit_transform(X_train_ss)
X_test_pca = pca.transform(X_test_ss)
```

Setelah PCA, dimensi berkurang menjadi 509 dan meningkatkan kinerja model sangat signifikan.

Catatan: kenapa proses training jadi lebih lama?

```
Akurasi 0.9262563931509895
              precision    recall  f1-score   support

           0       0.93      0.44      0.60      1676
           1       0.93      0.51      0.66      1641
           2       0.93      1.00      0.96     31033
           3       0.95      0.49      0.64      1626

    accuracy                           0.93     35976
   macro avg       0.93      0.61      0.72     35976
weighted avg       0.93      0.93      0.92     35976
```

Resampling
[todo]

3.3 Tuning

Saat menggunakan suatu library, ada parameter yang harus diberikan. Sebagai contoh, code di bawah memperlihatkan parameter pada XGBoost.

```
clf = xgb.XGBClassifier(objective = "multi:softprob", num_class = 4,
                  eval_metric = "mlogloss", max_depth = 24, gamma=0.1,
                  subsample = 0.90,learning_rate=0.01, n_estimators = 10, nthread=-1)
```

Parameter yang mempengaruhi proses learning disebut hyperparameter (bagian yang di-bold). Ada beberapa pendekatan untuk mencari hyperparameter yang optimal:

1. Random. Set secara random, hitung kinerja model.
2. Intuisi, rule of thumb. Berdasarkan pengalaman, set hyperparameter.
3. Grid search: sistematis mencari kombinasi hyperparameter yang ideal (kelemahan: lama). Gridsearch pada scikit-learn: `sklearn.grid_search`

Sebagai contoh, saat learning_rate diubah menjadi 0.1, maka ada perbaikan kinerja model:

```
Akurasi 0.9284245052257061
              precision    recall  f1-score   support

           0       0.92      0.47      0.62      1676
           1       0.92      0.53      0.67      1641
           2       0.93      1.00      0.96     31033
           3       0.93      0.50      0.65      1626

    accuracy                           0.93     35976
   macro avg       0.92      0.62      0.73     35976
weighted avg       0.93      0.93      0.92     35976
```

Artikel terkait untuk parameter tuning di XGBoost:
https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

4. Menyimpan Model dan Memprediksi Data Baru

[todo: perbaiki]

Setelah puas dengan kinerja model, model hasil train dapat disimpan dengan library joblib:

```
import joblib
import pandas as pd
joblib.dump(clf, "C:\\data\\dec_tree_olympic.joblib')
```

Model ini akan dicoba untuk memprediksi data yang belum diketahui kelasnya.

Buat notebook baru. Bagian yang tersulit adalah memastikan bahwa struktur data yang akan diprediksi sama dengan struktur data yang dibuat saat membuat model. Ini disebabkan one hot encoder dapat membuat jumlah kolom menjadi besar. Praproses yang dilakukan antara data train dan data yang akan diprediksi juga harus sama.

Pertama load struktur data train dan konversi tipe atribut nominal. Ini diperlukan untuk mendapatkan struktur kategorinya. Tahap ini dapat dihilangkan jika kategori data train disimpan ke dalam file.

```
df_train = pd.read_csv("C:\\data\\athlete_events.csv")
cat_col = ["Sex", "NOC", "Season", "City", "Sport"]
for col in cat_col:
    df_train[col] = df_train[col].astype("category")
```

Masukkan data yang akan diprediksi (bisa juga disimpan dalam file csv dan di-load dengan read_csv):

Lakukan praproses **yang sama** dengan data train (catatan: dapat dijadikan fungsi supaya tidak ada duplikasi code). Penting untuk menyamakan tahapan praproses karena model dibuat berdasarkan data train yang di praproses.

Konversi data yang akan diprediksi ke dalam kategori yang sama dengan data training. Walaupun misalnya di data prediksi hanya berisi dua negara CHN dan USA, tetapi tetap perlu dikonversi dengan jumlah negara yang sama dengan data training agar strukturnya sama persis.

```
cat_col = ["Sex", "NOC", "Season", "City", "Sport"]
for col in cat_col:
    df_prediksi[col] = df_prediksi[col].astype("category",
categories=df_train[col].cat.categories)

df_prediksi["Year"] =
df_prediksi["Year"].astype(pd.api.types.CategoricalDtype(categories =
                          df_train["Year"].unique()))
```

Konversi menjadi one hot encoder:

```
df_prediksi2 = pd.get_dummies(data=df_prediksi, columns=['Periode','Sex', 'NOC',
'Year', 'Season', 'City', 'Sport'])
```

Samakan urutan kolom dengan data training. Walaupun scikit-learn dapat menerima Dataframe sebagai input, tetapi scikit learn mengabaikan label atribut sehingga perlu dipastikan urutan atribut sama. Untuk memastikan itu, akan digunakan nama kolom yang disimpan sebelumnya saat proses training.

```
import pickle
with open (C:\\data\\olympic_x_train_columns.pickle, 'rb') as fp:
    X_train_column = list(pickle.load(fp))

df_prediksi2 = df_prediksi2[X_train_column]
```

Terakhir load model dan lakukan prediksi:

```
from sklearn.externals import joblib
clf = joblib.load("C:\\data\\dec_tree_olympic.joblib")
clf.predict(df_prediksi2)
```

Hasilnya:

```
Out[13]: array([2, 1])
```

Artinya berdasarkan data input dan model, maka untuk cabang Weightlifting pada olimpiade tahun 2004, orang pertama (Laki-laki, USA, 25 tahun, 170cm, 60Kg) akan diprediksi tidak akan mendapatkan medali sedangkan orang kedua (Perempuan, CHN, 25 tahun, 160cm, 70Kg) akan mendapatkan medali emas.

Daftar Pustaka

Brownlee, Jason. "Machine learning learning performance improvement cheat sheet ." Ebook. Edition 1 (2017): 4.

Todo: pipeline