models

June 20, 2021

[1]:
```
%load_ext autoreload
import pandas as pd
import numpy as np
```

1 Loading the Data

[2]:
```
X_train_1 = pd.read_csv('data/dengue_features_train.csv')
y_train = pd.read_csv('data/dengue_labels_train.csv')['total_cases']
attr = list(X_train_1)
attr
```

[2]:
```
['city',
 'year',
 'weekofyear',
 'week_start_date',
 'ndvi_ne',
 'ndvi_nw',
 'ndvi_se',
 'ndvi_sw',
 'precipitation_amt_mm',
 'reanalysis_air_temp_k',
 'reanalysis_avg_temp_k',
 'reanalysis_dew_point_temp_k',
 'reanalysis_max_air_temp_k',
 'reanalysis_min_air_temp_k',
 'reanalysis_precip_amt_kg_per_m2',
 'reanalysis_relative_humidity_percent',
 'reanalysis_sat_precip_amt_mm',
 'reanalysis_specific_humidity_g_per_kg',
 'reanalysis_tdtr_k',
 'station_avg_temp_c',
 'station_diur_temp_rng_c',
 'station_max_temp_c',
 'station_min_temp_c',
 'station_precip_mm']
```

1.1 Cleaning the noisy training data

[3]:
```
def bools_to_indexes(booleans):
    r = []
    for idx, x in enumerate(booleans):
        if x:
            r.append(idx)
    return r

idx = bools_to_indexes(X_train_1['weekofyear'] == 53)
y_train.drop(idx, inplace=True)
y_train.reset_index(drop=True, inplace=True)
X_train_1.drop(idx, inplace=True)
X_train_1.reset_index(drop=True, inplace=True)
X_train_1.shape
```

[3]:
```
(1451, 24)
```

2 Model Selection

[4]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
score_metric='neg_mean_absolute_error'
jobs=-1 #-1 to make it execute in parallel
verbose_level = 0
random_n = 42
base_args = {'estimator': None, 'param_distributions': None, 'n_iter': None,␣
 ↪'scoring': score_metric, 'n_jobs': jobs, 'cv': None, 'verbose':␣
 ↪verbose_level, 'random_state': random_n, 'return_train_score': True, 'iid':␣
 ↪True}
```

2.1 SVR

- The results with the kernel *sigmoid* and *poly* were too bad, so we removed them.

[58]:
```
k_folds=5
n_iter_search = 10
C = sp_randint(0, 10000)
params = {'kernel':['linear'], 'gamma':['scale'], 'C': C}
```

```
SVR_optimizer = RandomizedSearchCV(estimator=SVR(), param_distributions=params,␣
↪n_iter=n_iter_search, scoring=score_metric, n_jobs=jobs, cv=k_folds,␣
↪verbose=verbose_level, random_state=random_n, return_train_score=True,␣
↪iid=True)
```

2.2 Regression Trees

- 18.01 - with 2 previous weeks & without PCA & with (max_depth=6, min_samples_leaf=0.1611807565247405, min_samples_split=0.11193019906931466)
- 18.29 - With PCA at 0.9
- 18.27 - With PCA at 0.95
- 18.36 - With PCA at 0.65. PCA appears to be only making the model worse.
- 18.38 - Without PCA and with previous weeks. Clearly the previous weeks are useful
- 17.87 - Without PCA and with 3 previous weeks
- 17.86 - Without PCA and with 4 previous weeks
- 18.28 - With PCA 0.95 and 3 previous weeks fixed
- 9.16 - Without PCA, with 3 weeks and 1 last infection (max_depth=5, min_samples_leaf=0.03, min_samples_split=0.108)
- **9.04** - Without PCA, with 3 weeks and 1 last infection (max_depth=5, min_samples_leaf=0.03, min_samples_split=0.108)

[52]:
```
k_folds=10
n_iter_search = 100
min_samples = sp_uniform(0.01, 0.35)
params = {'criterion':['mae'], 'max_depth': sp_randint(2, 10),␣
↪'min_samples_split': min_samples, 'min_samples_leaf': min_samples}
Tree_optimizer = RandomizedSearchCV(estimator=DecisionTreeRegressor(),␣
↪param_distributions=params, n_iter=n_iter_search, scoring=score_metric,␣
↪n_jobs=jobs, cv=k_folds, verbose=verbose_level, random_state=random_n,␣
↪return_train_score=True, iid=True)
```

2.3 Random Forests

- 18.34 With 4 previous weeks and without PCA
- 17.79 With fixed 3 previous weeks and PCA at 0.95 (n_estimators= ?, max_depth = 2, min_samples_leaf=0.112, min_samples_split=0.224)
- 17.74 With fixed 3 previous weeks and without PCA (n_estimators= 13 max_depth = 5, min_samples_leaf=0.09, min_samples_split=0.24)
- **9.13** with 3 previous weeks and 1 last infected (n_estimators=9 max_depth = 9, min_samples_leaf=0.014, min_samples_split=0.07)
- 9.22 with 3 previous weeks and 3 last infected (n_estimators=9 max_depth = 9, min_samples_leaf=0.014, min_samples_split=0.08)

[53]:
```
k_folds=10
n_iter_search = 40
params = {'n_estimators': sp_randint(2,50), 'criterion':['mae'], 'max_depth':␣
↪sp_randint(2, 10)}
```

```
Forest_optimizer =␣
 ↪RandomizedSearchCV(estimator=RandomForestRegressor(n_jobs=-1),␣
 ↪param_distributions=params, n_iter=n_iter_search, scoring=score_metric,␣
 ↪n_jobs=jobs, cv=k_folds, verbose=verbose_level, random_state=random_n,␣
 ↪return_train_score=True, iid=True)
```

2.4 Adaboost of Trees

- 10.78 - With 3 last weeks a 3 last infected
- **8.49** - With 3 last weeks a 3 last infected and only max_depth tuned.

[54]:
```
k_folds=10
n_iter_search = 20
params = {'n_estimators': sp_randint(40, 100), 'base_estimator__criterion':
 ↪['mae'], 'base_estimator__max_depth': sp_randint(2,7)}
AdaTree_optimizer =␣
 ↪RandomizedSearchCV(estimator=AdaBoostRegressor(base_estimator=DecisionTreeRegressor()),␣
 ↪param_distributions=params, n_iter=n_iter_search, scoring=score_metric,␣
 ↪n_jobs=jobs, cv=k_folds, verbose=verbose_level, random_state=random_n,␣
 ↪return_train_score=True, iid=True)
```

2.5 KNN

- 21.349 - with PCA at 0.65 & 2 previous weeks
- 20.36 - without PCA

[55]:
```
k_folds=10
n_iter_search = 100
params = {'n_neighbors': sp_randint(3,150), 'weights': ['uniform', 'distance']}
KNN_optimizer = RandomizedSearchCV(estimator=KNeighborsRegressor(n_jobs=-1),␣
 ↪param_distributions=params, n_iter=n_iter_search, scoring=score_metric,␣
 ↪n_jobs=jobs, cv=k_folds, verbose=verbose_level, random_state=random_n,␣
 ↪return_train_score=True, iid=True)
```

3 The most simple prediction

- Our first attempt consists of simply adding weather information from the previous weeks and finding the optimal the optimal parameter through exaustive search (coded by us) and find its optimal hyper-parameters (using `RandomSearchCV`).
- Interestingly, PCA makes all the models worst in this case.
- It turned out to be a `RandomForestRegressor` as you can see in the `best_attempt` variable. By using this model and adding the 3 previous weeks of weather to each entry, we obtained a MAE of approximately 17 by 10-folded cross validation.
- Unfortunatly, this model (when trained with all the train data) resulted in an 27 MAE when submitted to the platform. This indicates overfitting and that there must be considerable differences between the train and test data.

[ ]:
```
%autoreload
from utils.OurPipeline import create_pipeline
from sklearn.decomposition import PCA

optimizers=[Tree_optimizer, Forest_optimizer, AdaTree_optimizer, KNN_optimizer,␣
 ↪SVR_optimizer]
weeks = [1,2,3,4]

n_total = len(optimizers) * len(weeks)


results=[]
best_attempt = None
best_score = np.inf
idx=0
for opt in optimizers:
    for w in weeks:
        pipeline = create_pipeline(attr, n_weeks=w, estimator_optimizer=opt,␣
 ↪pca=None)
        pipeline.fit(X_train_1, y_train)
        score = pipeline.named_steps['est_opt'].best_score_
        best_estimator = pipeline.named_steps['est_opt'].best_estimator_
        attempt = [best_estimator, w, score]
        if abs(score) < best_score:
            best_score = abs(score)
            best_attempt = attempt
            print('\nBest score of {} with the estimator {}'.format(best_score,␣
 ↪best_estimator))
        idx+=1
        print(str(idx) + '/' + str(n_total), end='\t')
        results.append(attempt)
```

[62]:
```
best_attempt
```

[62]:
```
[RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=2,
            max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=13, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0, warm_start=False),
 3,
 -17.87464878333245]
```

3.0.1 Train

[69]:
```
%autoreload
from utils.OurPipeline import create_pipeline
pipeline = create_pipeline(attr, n_weeks=3, pca=None)
X_train = pipeline.fit_transform(X_train_1)

model = RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=2,␣
 ↪n_estimators=13, n_jobs=-1, random_state=random_n)
model.fit(X_train, y_train)
```

[69]:
```
RandomForestRegressor(bootstrap=True, criterion='mae', max_depth=2,
            max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=13, n_jobs=-1,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

3.0.2 Predict

[76]:
```
X_test = pipeline.transform(X_test_1)
pred = model.predict(X_test)
pred = list(map(lambda x: int(np.round(x)), pred))
```

3.0.3 Submit

[78]:
```
submit = pd.DataFrame(pred, columns=['total_cases'])
x_3 = X_test_1.iloc[:,:3].copy()
submit = pd.concat([x_3, submit], axis=1)
submit.to_csv('data/submit.csv', index=False)
```

4 Prediction with the last infected

- As we could see on the analysis notebook, the number of infected on any week is highly linked to the number of infected at its previous weeks. Including the number of infected (or at least an approximation) on the previous weeks should be key to very accurate predictions.
- For this sake, we created the `LastInfected` module which is included in the pipeline.
- After the exaustive search, the best model was the SVR which obtained an MAE of 6.52 on the training dataset, which is a great improvement.
- Given that we are making sequential predictions, i.e.: the prediction from one week relies on the prediction from the previous weeks, we must make the transformations and predictions one by one.
- The submission MAE was approximately 26, which is an improvement and is not bad given that the `total_cases` feature on the training set ranges from 0 to 400. However, we were expecting a much smaller result.

4.0.1 Optimization

[ ]:
```
%autoreload
from OurPipeline import create_pipeline
from sklearn.decomposition import PCA

optimizers=[Tree_optimizer, Forest_optimizer, AdaTree_optimizer, KNN_optimizer,␣
↪SVR_optimizer]
weeks = [1, 2, 3]
weeks_infected = [2, 3, 4]
pca = [PCA(0.95), None]

n_total = len(optimizers) * len(weeks) * len(weeks_infected) * len(pca)

results=[]
best_attempt = None
best_score = np.inf
idx=0
for opt in optimizers:
    for w in weeks:
        for wi in weeks_infected:
            for p in pca:
                pipeline = create_pipeline(attr, n_weeks=w,␣
↪n_weeks_infected=wi, estimator_optimizer=opt, pca=None)
                pipeline.fit(X_train_1, y_train)
                score = pipeline.named_steps['est_opt'].best_score_
                best_estimator = pipeline.named_steps['est_opt'].best_estimator_
                attempt = [best_estimator, w, wi, p, score]
                if abs(score) < best_score:
                    best_score = abs(score)
                    best_attempt = attempt
                    print('\nBest score of {} with the estimator {}'.
↪format(best_score, best_estimator))
                idx+=1
                print(str(idx) + '/' + str(n_total), end='\t')
                results.append(attempt)
```

[37]:
```
best_attempt
```

[37]:
```
[SVR(C=5191, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False),
 1,
 3,
 None,
 -6.522347109745663]
```

5 Predict

[171]:
```
%autoreload
from OurPipeline import create_pipeline

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3, pca=None)
X_train = pipeline.fit_transform(X_train_1, y_train)
```

[56]:
```
model = SVR(kernel= 'linear', C=5191, gamma='scale')
model.fit(X_train, y_train)
```

[56]:
```
SVR(C=5191, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

5.0.1 Loading test data

[39]:
```
X_test_1 = pd.read_csv('data/dengue_features_test.csv')
print(X_test_1.shape)
```

```
(416, 24)
```

5.1 One by one prediction

[172]:
```
from utils.predict_in_order import predict_in_order
predictions = predict_in_order(X_test_1, model, pipeline)
len(predictions)
```

[172]:
```
416
```

[148]:
```
submit = pd.DataFrame(predictions, columns=['total_cases'])
x_3 = X_test_1.iloc[:,:3].copy()
submit = pd.concat([x_3, submit], axis=1)
submit.to_csv('data/submit.csv', index=False)
```

6 One by one prediction with noise

- We believe the reason why our predictions were not so great, was because this kind of prediction is very prone to a snowball effect on errors.
- To solve this we came up with an idea: Our problem was currently being trained on data which has all `last_infected` columns with the exact correct values. However, when we are predicting with the test set, the values we use on `last_infected` are mere predictions. By adding random noise to the `last_infected` columns on the training data we would make our model more "prepared" to accept entries in which the `last_infected` columns are not so accurate.
- However for this solution we need to know both: the mean of the error and its standard deviation (*std*), so that we can reproduce the error by a gaussian distribution. We already know the mean (MAE), we just need to know the *std*

- When dealing with the test data, the noise adding feature of the pipeline must be disabled, otherwise our predictions will be based on 2 layers of noise: our "synthetic" noise and the one created by the predictive model.
- The submission's MAE increased again to approximately 27.
- A very for why it isn't working is that the error when y is low is much smaller than when y is high.

6.0.1 Calculating an approximation of the *std*

[216]:
```
%autoreload
from OurPipeline import create_pipeline
from sklearn.model_selection import ShuffleSplit

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3, pca=None)
X_train = pipeline.fit_transform(X_train_1, y_train)
```

[217]:
```
sp = ShuffleSplit(n_splits=1, train_size=1000, test_size=None,␣
↪random_state=random_n)
for train, test in sp.split(X_train, y_train):
    X_train_std = X_train[train]
    y_train_std = y_train[train]
    X_test_std = X_train[test]
    y_test_std = y_train[test]
X_train_std.shape, y_train_std.shape
X_test_std.shape, y_test_std.shape
```

[222]:
```
model = SVR(kernel= 'linear', C=5191, gamma='scale')
model.fit(X_train_std, y_train_std)
```

[222]:
```
SVR(C=5191, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
  kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

[234]:
```
predictions = model.predict(X_test_std)
predictions = list(map(lambda x: int(np.round(x)), predictions))
errors = list(map(abs, predictions - y_test_std))
np.mean(errors), np.std(errors)
```

[234]:
```
(6.7785087719298245, 10.959317651673116)
```

6.0.2 Adding the noise and training

[157]:
```
%autoreload
from OurPipeline import create_pipeline

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3, add_noise=True,␣
↪noise_mean=6.78, noise_std=10.96, pca=None)
X_train = pipeline.fit_transform(X_train_1, y_train)
```

[ ]:
```
model.fit(X_train, y_train)
```

6.0.3 Disabling the noise and predicting

[161]:
```
%autoreload
from utils.OurPipeline import create_pipeline
from utils.predict_in_order import predict_in_order

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3,␣
 ↪add_noise=False, pca=None)
pipeline.fit_transform(X_train_1, y_train)

predictions = predict_in_order(X_test_1, model, pipeline)
len(predictions)
```

[161]:
```
416
```

6.1 Submission

[148]:
```
submit = pd.DataFrame(predictions, columns=['total_cases'])
x_3 = X_test_1.iloc[:,:3].copy()
submit = pd.concat([x_3, submit], axis=1)
submit.to_csv('data/submit.csv', index=False)
```

7 Test split of tail

- To simulate what we are doing with the test data, we are going to split the train data, for each city, by sampling N entries from the tail of each city for testing.
- We now have 580 entries of train data and 871 entres of test data, to figure out what is wrong.
- Since we can't use `RandomizedSearchCV` with this prediction mode (the one-by-one explained before), we opted to implement our own exhaustive search tool.
- Here we only worked with the `RandomForestRegressor` because it brought results almost as good as the `SVR` model and took far less time training.
- The optimal model turned out to be `RandomForestRegressor` with with 50 estimators and a maximum depth of 5.
- Even though we obtain a MAE of approximately 17 on our custom test set (which has twice as many entries as the one from the competition), when we submit the data with that model we obtain a MAE of approximately 30.
- We are hoping to be able to improve this result on the phase 2 of the project.

[5]:
```
idx_sj = X_train_1['city'] == 'sj'
X_sj = X_train_1[idx_sj]
y_sj = y_train[idx_sj]

idx_iq = X_train_1['city'] == 'iq'
X_iq = X_train_1[idx_iq]
y_iq = y_train[idx_iq]
```

```
X_sj.shape, y_sj.shape, X_iq.shape, y_iq.shape
```

[5]:
```
((933, 24), (933,), (518, 24), (518,))
```

[18]:
```
from sklearn.model_selection import train_test_split

l = train_test_split(X_sj, y_sj, train_size=0.4, test_size=None, shuffle=False)
X_train_sj = l[0]
X_test_sj = l[1]
y_train_sj = l[2]
y_test_sj = l[3]

l = train_test_split(X_iq, y_iq, train_size=0.4, test_size=None, shuffle=False)
X_train_iq = l[0]
X_test_iq = l[1]
y_train_iq = l[2]
y_test_iq = l[3]

X_train_sj.shape, X_test_sj.shape, y_train_sj.shape, y_test_sj.shape,␣
↪X_train_iq.shape, X_test_iq.shape, y_train_iq.shape, y_test_iq.shape
```

[18]:
```
((373, 24), (560, 24), (373,), (560,), (207, 24), (311, 24), (207,), (311,))
```

[19]:
```
X_train_2 = pd.concat([X_train_sj, X_train_iq])
y_train_2 = pd.concat([y_train_sj, y_train_iq])
X_test_2 = pd.concat([X_test_sj, X_test_iq])
y_test_2 = pd.concat([y_test_sj, y_test_iq])

X_train_2.reset_index(drop=True, inplace=True)
X_test_2.reset_index(drop=True, inplace=True)
y_train_2.reset_index(drop=True, inplace=True)
y_test_2.reset_index(drop=True, inplace=True)
X_train_2.shape, y_train_2.shape, X_test_2.shape, y_test_2.shape
```

[19]:
```
((580, 24), (580,), (871, 24), (871,))
```

7.0.1 Pipeline

[44]:
```
%autoreload
from utils.OurPipeline import create_pipeline

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3,␣
↪add_noise=False, pca=None)
X_train = pipeline.fit_transform(X_train_2, y_train_2)
```

7.0.2 Train

[37]:
```
%autoreload
from utils.OurPipeline import create_pipeline
from utils.predict_in_order import predict_in_order
from sklearn.metrics import mean_absolute_error

estimators = [25, 50, 75]
depth = [2,3,4,5]

best_mae=np.inf
best=None
for est in estimators:
    for d in depth:
        model = RandomForestRegressor(criterion='mae', n_estimators=est,␣
↪max_depth=d)
        model.fit(X_train, y_train_2)


        #pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3,␣
↪add_noise=False, pca=None)
        #pipeline.fit_transform(X_train_2, y_train_2)

        pred = predict_in_order(X_test_2, model=model, pipeline=pipeline)

        mae = mean_absolute_error(pred, y_test_2)
        if mae < best_mae:
            best_mae = mae
            best = (est, d)

best_mae, best
```

[37]:
```
(17.044776119402986, (50, 5))
```

7.0.3 Submit

[46]:
```
%autoreload
from utils.OurPipeline import create_pipeline
from utils.predict_in_order import predict_in_order

pipeline = create_pipeline(attr, n_weeks=1, n_weeks_infected=3,␣
↪add_noise=False, pca=None)
X_train = pipeline.fit_transform(X_train_2, y_train_2)

model = RandomForestRegressor(criterion='mae', n_estimators=50, max_depth=5)
model.fit(X_train, y_train_2)

pred = predict_in_order(X_test_1, model=model, pipeline=pipeline)
```

[47]:
```
submit = pd.DataFrame(pred, columns=['total_cases'])
x_3 = X_test_1.iloc[:,:3].copy()
submit = pd.concat([x_3, submit], axis=1)
submit.to_csv('data/submit.csv', index=False)
```