youtube-spam-v1

April 24, 2019

V1: + All models uses the TfidfVectorizer to do the the preprocessing

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import seaborn as sns
        import string
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.model_selection import train_test_split
        from nltk.corpus import stopwords
        from sklearn.preprocessing import LabelEncoder
        %matplotlib inline
```

```
In [2]: # Dataset from https://archive.ics.uci.edu/ml/datasets/YouTube+Spam+Collection#
        df1 = pd.read_csv("../data/UCI-YouTube-Spam-Collection/Youtube01-Psy.csv")
```

```
In [3]: df1.head()
```

```
Out[3]:                              COMMENT_ID                 AUTHOR  \
        0  LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU        Julius NM
        1  LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A       adam riyati
        2  LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8  Evgeny Murashkin
        3          z13jhp0bxqncu512g22wvzkasxmvvzjaz04   ElNino Melendez
        4          z13fwbwp1oujthgqj04chlngpvzmtt3r3dw            GsMega

                          DATE                                            CONTENT  \
        0  2013-11-07T06:20:48  Huh, anyway check out this you[tube] channel: ...
        1  2013-11-07T12:37:15  Hey guys check out my new channel and our firs...
        2  2013-11-08T17:34:21               just for test I have to say murdev.com
        3  2013-11-09T08:28:43   me shaking my sexy ass on my channel enjoy ^_^
        4  2013-11-10T16:05:38        watch?v=vtaRGgvGtWQ   Check this out .

           CLASS
        0      1
        1      1
        2      1
        3      1
        4      1
```

```
In [4]: # Load all our dataset to merge them
        df2 = pd.read_csv("../data/UCI-YouTube-Spam-Collection/Youtube02-KatyPerry.csv")
        df3 = pd.read_csv("../data/UCI-YouTube-Spam-Collection/Youtube03-LMFAO.csv")
        df4 = pd.read_csv("../data/UCI-YouTube-Spam-Collection/Youtube04-Eminem.csv")
        df5 = pd.read_csv("../data/UCI-YouTube-Spam-Collection/Youtube05-Shakira.csv")

In [5]: frames = [df1,df2,df3,df4,df5]

In [6]: # Merging or Concatenating our DF
        df_merged = pd.concat(frames)

In [7]: df_merged.head()

Out[7]:                                  COMMENT_ID               AUTHOR  \
        0  LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU         Julius NM
        1  LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A       adam riyati
        2  LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8  Evgeny Murashkin
        3           z13jhp0bxqncu512g22wvzkasxmvvzjaz04   ElNino Melendez
        4           z13fwbwp1oujthgqj04chlngpvzmtt3r3dw            GsMega

                        DATE                                        CONTENT  \
        0  2013-11-07T06:20:48  Huh, anyway check out this you[tube] channel: ...
        1  2013-11-07T12:37:15  Hey guys check out my new channel and our firs...
        2  2013-11-08T17:34:21               just for test I have to say murdev.com
        3  2013-11-09T08:28:43    me shaking my sexy ass on my channel enjoy ^_^
        4  2013-11-10T16:05:38          watch?v=vtaRGgvGtWQ   Check this out .

           CLASS
        0      1
        1      1
        2      1
        3      1
        4      1

In [8]: # Total Size
        df_merged.shape

Out[8]: (1956, 5)
```

Now let's create new feature "message length" and plot it to see if it's of any interest

```
In [9]: # Save and Write Merged Data to csv
        df_merged.to_csv("../data/youtube-spam-merged.csv")

In [10]: df = df_merged
```

Data Cleaning

```
In [11]: # Check for missing nan
         df.isnull().isnull().sum()
```

```
Out[11]: COMMENT_ID    0
         AUTHOR        0
         DATE          0
         CONTENT       0
         CLASS         0
         dtype: int64
```

Now drop "COMMENT_ID", 'AUTHOR', 'DATE', columns and rename CLASS and CONTENT to "label" and "content"

```
In [12]: ytb = df[["CONTENT","CLASS"]]
         ytb = df.rename(columns = {'CONTENT':'content','CLASS':'label'})
```

Let's look into our data

```
In [13]: ytb.groupby('label').describe()
```

```
Out[13]:        AUTHOR                          COMMENT_ID          \
                count unique      top  freq      count unique
         label
         0         951    922  5000palo     7        951    950
         1        1005    871     M.E.S     8       1005   1003


                                                       DATE        \
                                                top freq count unique
         label
         0      _2viQ_Qnc68fX3dYsfYuM-m4ELMJvxOQBmBOFHqGOk0     2    951    950
         1      LneaDw26bFuH6iFsSrjlJLJIX3qD4R8-emuZ-aGUj0o     2    760    760


                                          content        \
                                  top freq   count unique
         label
         0      2013-10-05T00:57:25.078000     2     951    919
         1      2013-10-12T09:13:19.285000     1    1005    841



                                          top freq
         label
         0                                     Like     4
         1      Check out this video on YouTube:    97
```

Now let's create new feature "message length" and plot it to see if it's of any interest

```
In [14]: ytb['length'] = ytb['content'].apply(len)
         ytb['label'] = ytb['label'].apply(lambda x: 'spam' if x==1 else 'ham')
         ytb.head()
```

```
Out[14]:                                 COMMENT_ID          AUTHOR  \
         0  LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU    Julius NM
```

3

```
        1   LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A          adam riyati
        2   LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8   Evgeny Murashkin
        3            z13jhp0bxqncu512g22wvzkasxmvvzjaz04    ElNino Melendez
        4            z13fwbwp1oujthgqj04chlngpvzmtt3r3dw             GsMega

                    DATE                                            content  \
0   2013-11-07T06:20:48   Huh, anyway check out this you[tube] channel: ...
1   2013-11-07T12:37:15   Hey guys check out my new channel and our firs...
2   2013-11-08T17:34:21                    just for test I have to say murdev.com
3   2013-11-09T08:28:43    me shaking my sexy ass on my channel enjoy ^_^
4   2013-11-10T16:05:38              watch?v=vtaRGgvGtWQ    Check this out .

    label   length
0   spam        56
1   spam       166
2   spam        38
3   spam        48
4   spam        39
```
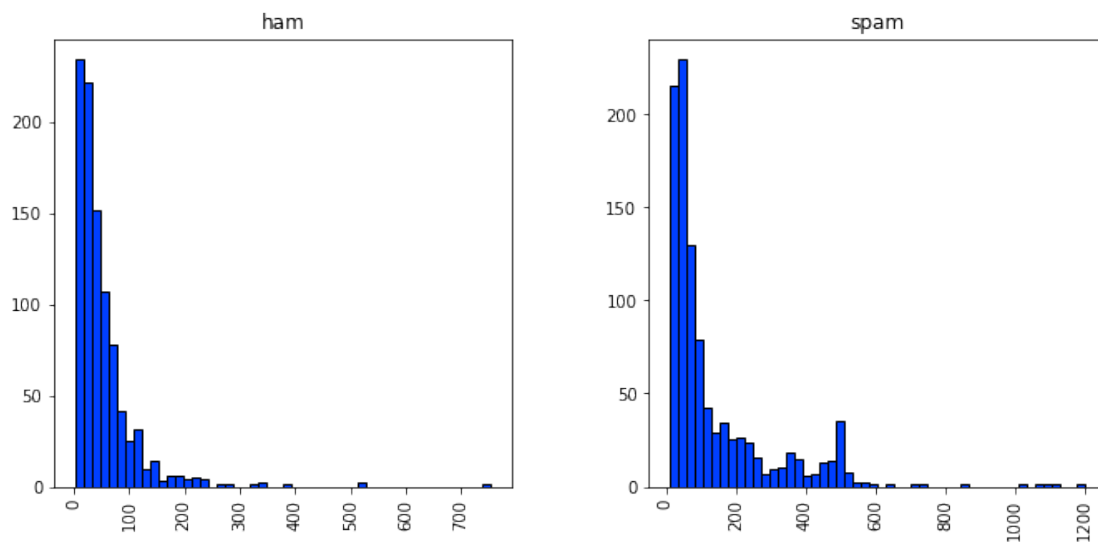
```
In [15]: mpl.rcParams['patch.force_edgecolor'] = True
         plt.style.use('seaborn-bright')
         ytb.hist(column='length', by='label', bins=50,figsize=(11,5))
         plt.savefig("../img/ytb-length-distribution.eps")
         plt.show()
```


0.0.1 Text processing and vectorizing our meddages

Let's create new data frame. We'll need a copy later on

```
In [16]: text_feat = ytb['content'].copy()
```

Now define our tex precessing function. It will remove any punctuation and stopwords aswell.

```
In [17]: # def text_process(text):

         #     text = text.translate(str.maketrans('', '', string.punctuation))
         #     text = [word for word in text.split() if word.lower() not in stopwords.words('e:

         #     return " ".join(text)
```

```
In [18]: # text_feat = text_feat.apply(text_process)
```

```
In [19]: vectorizer = TfidfVectorizer("english")
```

```
In [20]: features = vectorizer.fit_transform(text_feat)
```

```
In [21]: labels = LabelEncoder().fit_transform(ytb['label'])
         labels = labels.reshape(-1,1)
```

```
In [22]: text_feat.shape
```

```
Out[22]: (1956,)
```

```
In [23]: features.shape
```

```
Out[23]: (1956, 4454)
```

0.0.2 Classifiers and predictions

First of all let's split our features to test and train set
Now let's import bunch of classifiers, initialize them and make a dictionary to itereate through

```
In [24]: from sklearn.linear_model import LogisticRegression
         from sklearn.svm import SVC
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.ensemble import AdaBoostClassifier
         from sklearn.ensemble import BaggingClassifier
         from sklearn.ensemble import ExtraTreesClassifier
         from sklearn.metrics import precision_score
         from sklearn.metrics import recall_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import f1_score
```

```
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:2:
  from numpy.core.umath_tests import inner1d
```

```
In [25]: svc = SVC(kernel='sigmoid', gamma=1.0)
         knc = KNeighborsClassifier()
         mnb = MultinomialNB()
         dtc = DecisionTreeClassifier(random_state=111)
         lrc = LogisticRegression(solver='liblinear', penalty='l1')
         rfc = RandomForestClassifier(n_estimators=500, random_state=111)
         abc = AdaBoostClassifier(random_state=111)
         bc = BaggingClassifier(random_state=111)
         etc = ExtraTreesClassifier(random_state=111)

In [26]: features_train, features_test, labels_train, labels_test = train_test_split(features,

In [27]: clfs = {'SVC' : svc,'KN' : knc, 'NB': mnb, 'DT': dtc, 'LR': lrc, 'RF': rfc, 'AdaBoost
```

Let's make functions to fit our classifiers and make predictions

```
In [28]: def train_classifier(clf, feature_train, labels_train):
             clf.fit(feature_train, labels_train)

In [29]: def predict_labels(clf, features):
             return (clf.predict(features))
```

Now iterate through classifiers and save the results

```
In [30]: import time

In [31]: pred_scores = []
         for k,v in clfs.items():
             since = time.time()

             train_classifier(v, features_train, labels_train)
             time_elapsed = time.time() - since

             pred = predict_labels(v,features_test)
             pred_scores.append((k, [precision_score(labels_test,pred), recall_score(labels_te

/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataC
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversi
  from ipykernel import kernelapp as app
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversi
  from ipykernel import kernelapp as app
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataC
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataC
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversi
  from ipykernel import kernelapp as app
```

```
In [32]: # pred_scores
```

```
In [33]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall
         df
```

```
Out[33]:           Precision    Recall  Accuracy        F1 Training Time (s)
         SVC        0.958621  0.920530  0.938671  0.939189         0m 0.1041s
         KN         0.988095  0.274834  0.625213  0.430052         0m 0.0017s
         NB         0.907937  0.947020  0.923339  0.927066         0m 0.0018s
         DT         0.958763  0.923841  0.940375  0.940978         0m 0.0450s
         LR         0.953405  0.880795  0.916525  0.915663         0m 0.0081s
         RF         0.979021  0.927152  0.952300  0.952381         0m 3.5051s
         AdaBoost   0.941980  0.913907  0.926746  0.927731         0m 0.4922s
         BgC        0.972222  0.927152  0.948893  0.949153         0m 0.2403s
         ETC        0.944828  0.907285  0.925043  0.925676         0m 0.1329s
```

```
In [34]: df.plot(kind='bar', y="Accuracy", ylim=(0.3,1.0), figsize=(11,6), align='center', col
         plt.xticks(np.arange(9), df.index)
         plt.ylabel('Accuracy Score')
         plt.title('Distribution by Classifier')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.savefig("../img/ytb-acc-basemodel-v1.eps")
         plt.show()
```
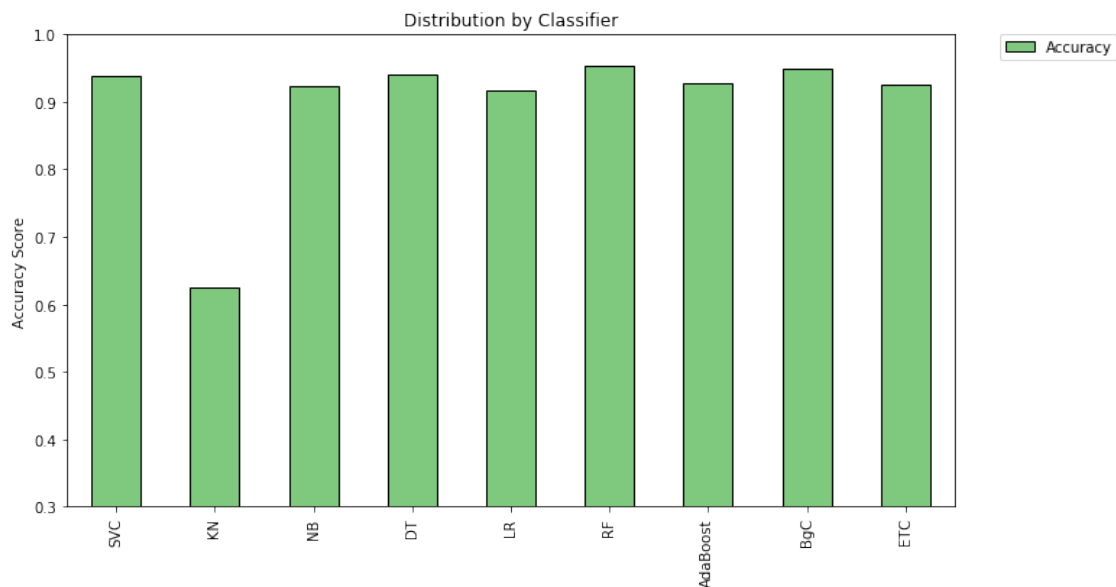

0.0.3 RNN

Define the RNN structure.

```
In [35]: from keras.models import Model
         from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
         from keras.optimizers import RMSprop
         from keras.preprocessing.text import Tokenizer
         from keras.preprocessing import sequence
         from keras.utils import to_categorical
         from keras.callbacks import EarlyStopping
         from keras.callbacks import Callback

Using TensorFlow backend.


In [36]: max_words = features_train.shape[0]
         max_len = features_train.shape[1]

In [37]: def RNN():
             inputs = Input(name='inputs',shape=[max_len])
             layer = Embedding(max_words,50,input_length=max_len)(inputs)
             layer = LSTM(100)(layer)
             layer = Dense(256,name='FC1')(layer)
             layer = Activation('relu')(layer)
             layer = Dropout(0.1)(layer)
             layer = Dense(1,name='out_layer')(layer)
             layer = Activation('sigmoid')(layer)
             model = Model(inputs=inputs,outputs=layer)
             return model

In [38]: model = RNN()
         model.summary()
         model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
```

```
_____
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 4454)              0
_____
embedding_1 (Embedding)      (None, 4454, 50)          68450
_____
lstm_1 (LSTM)                (None, 100)               60400
_____
FC1 (Dense)                  (None, 256)               25856
_____
activation_1 (Activation)    (None, 256)               0
_____
dropout_1 (Dropout)          (None, 256)               0
_____
out_layer (Dense)            (None, 1)                 257
_____
activation_2 (Activation)    (None, 1)                 0
```

```
================================================================
Total params: 154,963
Trainable params: 154,963
Non-trainable params: 0

_____
```

In [39]: since = time.time()

        model.fit(features_train, labels_train, epochs=10, batch_size=128,validation_split=0.2
                            callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)])

        time_elapsed = time.time() - since

```
Train on 1095 samples, validate on 274 samples
Epoch 1/10
1095/1095 [==============================] - 98s 90ms/step - loss: 0.6941 - acc: 0.4721 - val_l
Epoch 2/10
1095/1095 [==============================] - 78s 71ms/step - loss: 0.6934 - acc: 0.5032 - val_l
```

In [40]: print('Training complete in {:.0f}m {:.4f}s'.format(
                time_elapsed // 60, time_elapsed % 60))

Training complete in 2m 56.7070s

In [41]: pred = (np.asarray(model.predict(features_test, batch_size=128))).round()

In [42]: pred_scores.append(("LSTM", [precision_score(labels_test,pred), recall_score(labels_te

0.0.4 gcForest

In [43]: import sys
        sys.path.append("..")
        from gcforest.gcforest import GCForest
        from gcforest.utils.config_utils import load_json

In [44]: def get_toy_config():
            config = {}
            ca_config = {}
            ca_config["random_state"] = 111
            ca_config["max_layers"] = 20
            ca_config["early_stopping_rounds"] = 3
            ca_config["n_classes"] = 2
            ca_config["estimators"] = []
            ca_config["estimators"].append({"n_folds": 5, "type": "DecisionTreeClassifier"})
            ca_config["estimators"].append({"n_folds": 5, "type": "MultinomialNB"})
            ca_config["estimators"].append({"n_folds": 5, "type": "LogisticRegression"})
            config["cascade"] = ca_config
            return config

9

```
In [45]: config = get_toy_config()
         gc = GCForest(config)

         # features_train  ndarraylabels_train  (n_samples, )(n_samples, 1)
         features_gc_train = features_train.toarray()
         labels_gc_train = labels_train.reshape(-1)

         since = time.time()
         gc.fit_transform(features_gc_train, labels_gc_train)

         time_elapsed = time.time() - since
```

```
[ 2019-04-24 11:55:18,299][cascade_classifier.fit_transform] X_groups_train.shape=[(1369, 4454)
[ 2019-04-24 11:55:18,349][cascade_classifier.fit_transform] group_dims=[4454]
[ 2019-04-24 11:55:18,350][cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-24 11:55:18,351][cascade_classifier.fit_transform] group_ends=[4454]
[ 2019-04-24 11:55:18,353][cascade_classifier.fit_transform] X_train.shape=(1369, 4454),X_test
[ 2019-04-24 11:55:18,394][cascade_classifier.fit_transform] [layer=0] look_indexs=[0], X_cur_t
[ 2019-04-24 11:55:18,783][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:19,107][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:19,340][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:19,696][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:20,094][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:20,095][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-24 11:55:20,134][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,160][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,203][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,249][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,277][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,279][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-24 11:55:20,349][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,376][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,415][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,450][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,480][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,481][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-24 11:55:20,482][cascade_classifier.calc_accuracy] Accuracy(layer_0 - train.classifie
[ 2019-04-24 11:55:20,504][cascade_classifier.fit_transform] [layer=1] look_indexs=[0], X_cur_t
[ 2019-04-24 11:55:20,683][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:20,980][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:21,167][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:21,347][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:21,482][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:21,483][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-24 11:55:21,519][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-24 11:55:21,594][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-24 11:55:21,641][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-24 11:55:21,668][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
```

```
[ 2019-04-24 11:55:21,719][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-24 11:55:21,721][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-24 11:55:21,778][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,820][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,863][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,909][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,954][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,956][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-24 11:55:21,958][cascade_classifier.calc_accuracy] Accuracy(layer_1 - train.classifie
[ 2019-04-24 11:55:22,025][cascade_classifier.fit_transform] [layer=2] look_indexs=[0], X_cur_
[ 2019-04-24 11:55:22,294][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:22,500][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:22,701][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:23,068][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:23,338][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:23,339][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-24 11:55:23,377][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,439][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,490][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,532][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,570][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,582][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-24 11:55:23,609][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,642][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,684][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,737][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,780][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,783][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-24 11:55:23,787][cascade_classifier.calc_accuracy] Accuracy(layer_2 - train.classifie
[ 2019-04-24 11:55:23,804][cascade_classifier.fit_transform] [layer=3] look_indexs=[0], X_cur_
[ 2019-04-24 11:55:23,966][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:24,250][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:24,442][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:24,570][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:24,937][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:24,939][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-24 11:55:25,002][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,025][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,069][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,091][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,135][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,136][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-24 11:55:25,160][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-24 11:55:25,195][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-24 11:55:25,228][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-24 11:55:25,262][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-24 11:55:25,297][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-24 11:55:25,301][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
```

```
[ 2019-04-24 11:55:25,305][cascade_classifier.calc_accuracy] Accuracy(layer_3 - train.classifi
[ 2019-04-24 11:55:25,334][cascade_classifier.fit_transform] [layer=4] look_indexs=[0], X_cur_
[ 2019-04-24 11:55:25,500][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:25,631][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:25,758][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:26,094][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:26,403][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:26,405][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-24 11:55:26,429][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,497][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,542][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,607][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,672][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,673][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-24 11:55:26,737][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,800][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,878][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,928][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,974][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,979][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-24 11:55:26,981][cascade_classifier.calc_accuracy] Accuracy(layer_4 - train.classifi
[ 2019-04-24 11:55:26,982][cascade_classifier.fit_transform] [Result][Optimal Level Detected]
```

```
In [46]: print('Training complete in {:.0f}m {:.4f}s'.format(
              time_elapsed // 60, time_elapsed % 60))

Training complete in 0m 8.7054s
```

```
In [47]: pred = predict_labels(gc,features_test.toarray())
         pred_scores.append(("DCF", [precision_score(labels_test,pred), recall_score(labels_te

[ 2019-04-24 11:55:27,054][cascade_classifier.transform] X_groups_test.shape=[(587, 4454)]
[ 2019-04-24 11:55:27,089][cascade_classifier.transform] group_dims=[4454]
[ 2019-04-24 11:55:27,090][cascade_classifier.transform] X_test.shape=(587, 4454)
[ 2019-04-24 11:55:27,122][cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
[ 2019-04-24 11:55:27,215][cascade_classifier.transform] [layer=1] look_indexs=[0], X_cur_test
```

```
In [48]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recal
         df

Out[48]:         Precision    Recall   Accuracy        F1  Training Time (s)
         SVC      0.958621  0.920530  0.938671  0.939189        0m 0.1041s
         KN       0.988095  0.274834  0.625213  0.430052        0m 0.0017s
         NB       0.907937  0.947020  0.923339  0.927066        0m 0.0018s
         DT       0.958763  0.923841  0.940375  0.940978        0m 0.0450s
         LR       0.953405  0.880795  0.916525  0.915663        0m 0.0081s
```

```
RF          0.979021  0.927152  0.952300  0.952381      0m 3.5051s
AdaBoost    0.941980  0.913907  0.926746  0.927731      0m 0.4922s
BgC         0.972222  0.927152  0.948893  0.949153      0m 0.2403s
ETC         0.944828  0.907285  0.925043  0.925676      0m 0.1329s
LSTM        0.514480  1.000000  0.514480  0.679415     2m 56.7070s
DCF         0.956522  0.947020  0.950596  0.951747      0m 8.7054s
```

```python
In [49]: df.plot(kind='bar', y="Accuracy", ylim=(0.5,1.0), figsize=(11,6), align='center', col
         plt.xticks(np.arange(11), df.index)
         plt.ylabel('Accuracy Score')
         plt.title('Distribution by Classifier')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.savefig("../img/ytb-acc-v1.eps")
         plt.show()
```
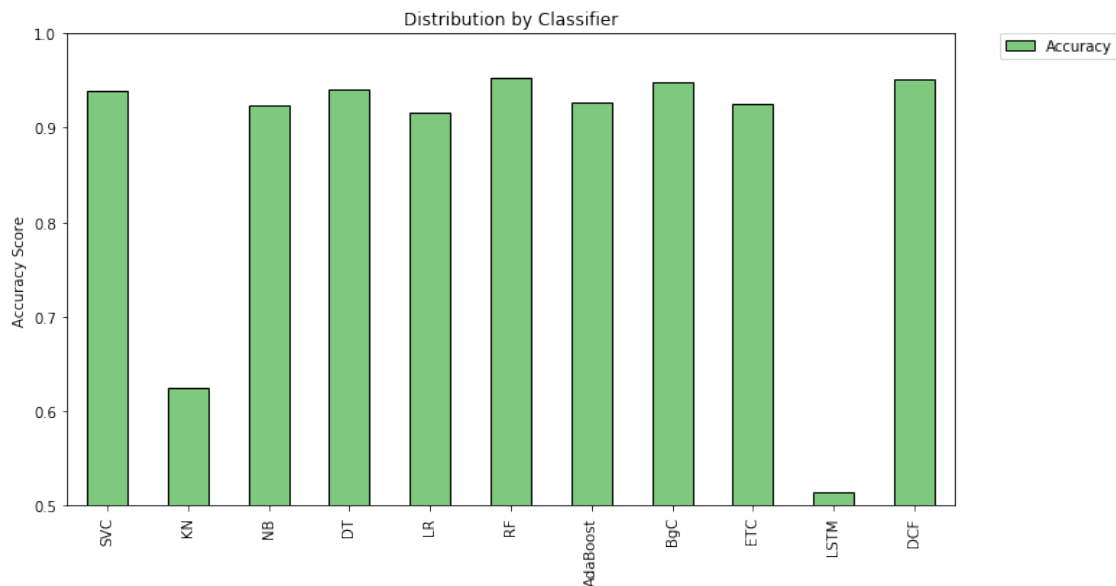


```python
In [50]: import pickle
         # dump
         with open("../pkl/ytb-gc-v1.pkl", "wb") as f:
             pickle.dump(gc, f, pickle.HIGHEST_PROTOCOL)

         # # load
         # with open("../pkl/2018_gc.pkl", "rb") as f:
         #     gc = pickle.load(f)

In [ ]:
```