youtube-spam-v2

April 23, 2019

V2: + All models uses the CountVectorizer 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 CountVectorizer
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	DATE	CONTENT	CLASS
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	1
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	1
2	LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8	Evgeny Murashkin	2013-11-08T17:34:21	just for test I have to say murdev.com	1
3	z13jhp0bxqncu512g22wvzkasxmvvzjaz04	ElNino Melendez	2013-11-09T08:28:43	me shaking my sexy ass on my channel enjoy ^_^	1
4	z13fwbwp1oujthgqj04chlngpvzmtt3r3dw	GsMega	2013-11-10T16:05:38	watch?v=vtaRGgvGtWQ Check this out .	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      LneaDw26bFvPh9xBHNw1btQoyP60ay_WWthtvXCx37s    2   760    760

                                          content         \
                                   top freq   count unique
         label
         0      2013-10-05T00:57:25.078000    2     951    919
         1             2014-01-19T17:20:58    1    1005    841


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

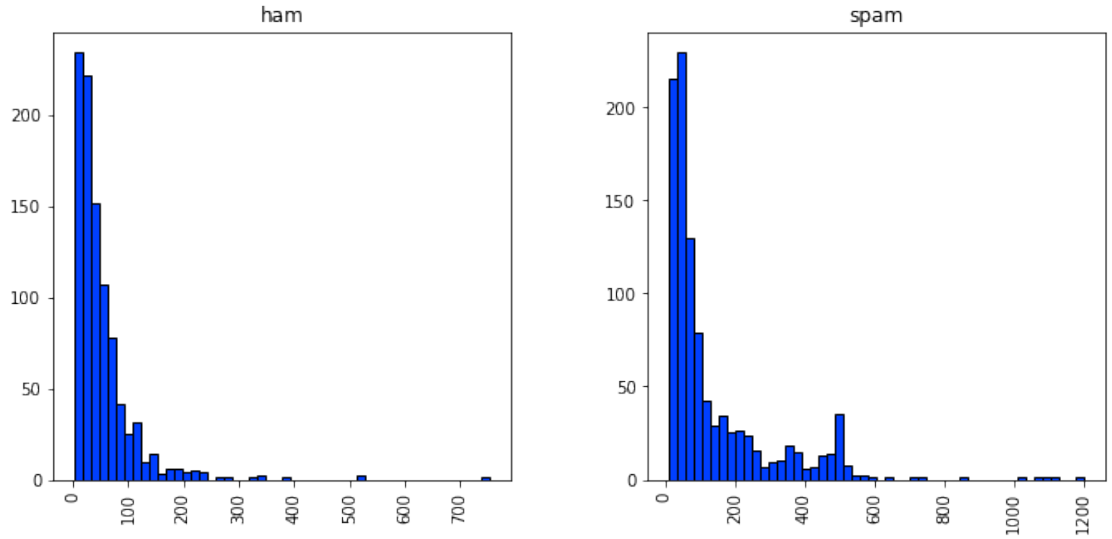
Now let's create new feature "message length" and plot it to see if it's of any interest

```
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
```

```
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 = CountVectorizer()
```

```
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:29
  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

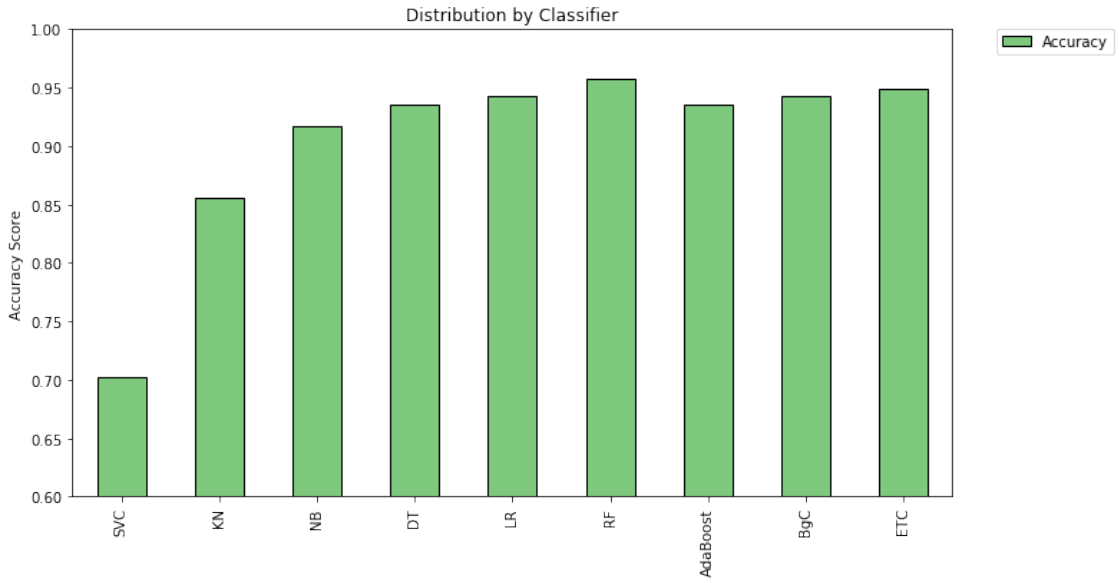
```
In [30]: import time

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: DataConversio
  from ipykernel import kernelapp as app
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversio
  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: DataConversio
  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	0.706840	0.718543	0.701874	0.712644	0m 0.1014s
KN	0.986547	0.728477	0.855196	0.838095	0m 0.0011s
NB	0.904153	0.937086	0.916525	0.920325	0m 0.0009s
DT	0.937086	0.937086	0.935264	0.937086	0m 0.0247s
LR	0.952703	0.933775	0.942078	0.943144	0m 0.0045s
RF	0.979239	0.937086	0.957411	0.957699	0m 3.3739s
AdaBoost	0.942953	0.930464	0.935264	0.936667	0m 0.4322s
BgC	0.958904	0.927152	0.942078	0.942761	0m 0.2488s
ETC	0.968966	0.930464	0.948893	0.949324	0m 0.1476s

```
In [34]: df.plot(kind='bar', y="Accuracy", ylim=(0.6,1.0), figsize=(11,6), align='center', colo
         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-v2.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.
                             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 [==============================] - 72s 66ms/step - loss: 0.6936 - acc: 0.5014 - val_
Epoch 2/10
1095/1095 [==============================] - 64s 58ms/step - loss: 0.6932 - acc: 0.4932 - val_
Epoch 3/10
1095/1095 [==============================] - 64s 58ms/step - loss: 0.6934 - acc: 0.5078 - val_
```

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

```
Training complete in 3m 20.1492s
```

```
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

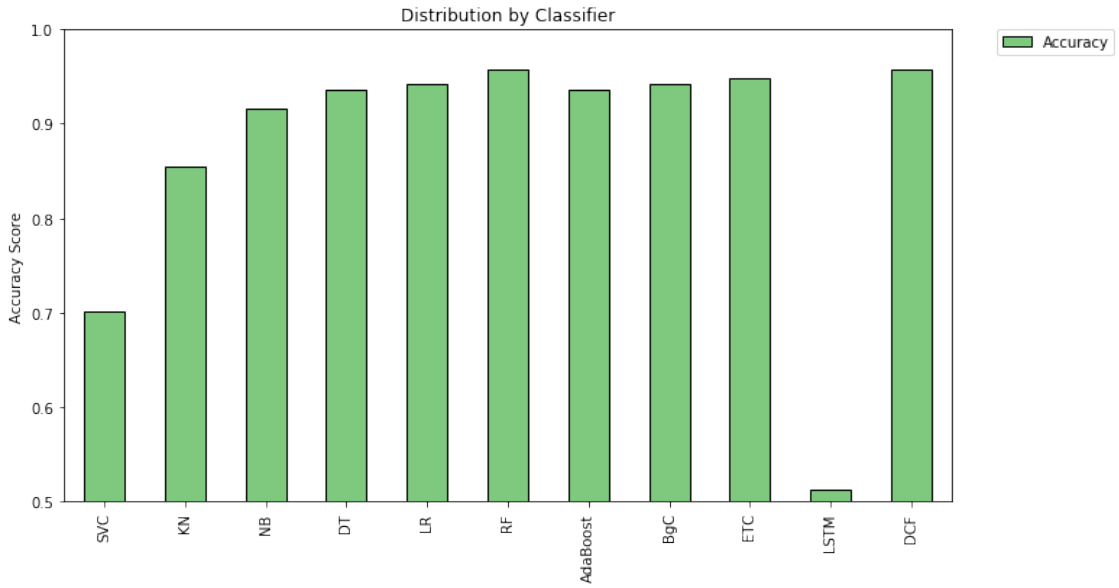
```
In [43]: import sys
         sys.path.append("..")
         from gcforest.gcforest import GCForest
         from gcforest.utils.config_utils import load_json

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

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

[ 2019-04-23 22:26:44,477][cascade_classifier.fit_transform] X_groups_train.shape=[(1369, 4454)
[ 2019-04-23 22:26:44,515][cascade_classifier.fit_transform] group_dims=[4454]
[ 2019-04-23 22:26:44,516][cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-23 22:26:44,518][cascade_classifier.fit_transform] group_ends=[4454]
[ 2019-04-23 22:26:44,521][cascade_classifier.fit_transform] X_train.shape=(1369, 4454),X_test
[ 2019-04-23 22:26:44,562][cascade_classifier.fit_transform] [layer=0] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:44,842][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,115][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,372][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,649][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,898][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,899][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 22:26:45,924][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:45,966][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:46,005][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:46,046][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:46,079][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:46,080][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 22:26:46,122][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,166][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,199][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,239][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,283][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,286][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 22:26:46,288][cascade_classifier.calc_accuracy] Accuracy(layer_0 - train.classifie
[ 2019-04-23 22:26:46,303][cascade_classifier.fit_transform] [layer=1] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:46,480][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 22:26:46,661][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 22:26:46,836][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 22:26:47,059][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 22:26:47,218][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
```

```
[ 2019-04-23 22:26:47,219][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 22:26:47,262][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,302][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,342][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,381][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,425][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,426][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 22:26:47,458][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,501][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,532][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,572][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,614][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,616][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 22:26:47,618][cascade_classifier.calc_accuracy] Accuracy(layer_1 - train.classifie
[ 2019-04-23 22:26:47,643][cascade_classifier.fit_transform] [layer=2] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:47,809][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:47,979][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:48,126][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:48,297][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:48,463][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:48,464][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 22:26:48,492][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,534][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,566][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,605][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,646][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,647][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 22:26:48,686][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,720][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,753][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,786][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,819][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,820][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 22:26:48,822][cascade_classifier.calc_accuracy] Accuracy(layer_2 - train.classifie
[ 2019-04-23 22:26:48,841][cascade_classifier.fit_transform] [layer=3] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:49,058][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:49,368][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:49,606][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:49,803][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:49,980][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:49,981][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 22:26:50,004][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,046][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,068][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,105][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,129][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,130][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_i
[ 2019-04-23 22:26:50,163][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
```

```
[ 2019-04-23 22:26:50,196][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
[ 2019-04-23 22:26:50,228][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
[ 2019-04-23 22:26:50,260][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
[ 2019-04-23 22:26:50,295][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
[ 2019-04-23 22:26:50,297][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_i
[ 2019-04-23 22:26:50,298][cascade_classifier.calc_accuracy] Accuracy(layer_3 - train.classifie
[ 2019-04-23 22:26:50,318][cascade_classifier.fit_transform] [layer=4] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:50,484][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:50,667][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:50,861][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:51,087][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:51,260][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:51,261][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_i
[ 2019-04-23 22:26:51,284][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,319][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,348][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,376][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,407][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,408][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_i
[ 2019-04-23 22:26:51,444][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,479][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,511][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,548][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,580][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,581][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_i
[ 2019-04-23 22:26:51,583][cascade_classifier.calc_accuracy] Accuracy(layer_4 - train.classifie
[ 2019-04-23 22:26:51,602][cascade_classifier.fit_transform] [layer=5] look_indexs=[0], X_cur_t
[ 2019-04-23 22:26:51,844][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,076][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,321][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,545][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,780][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,781][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_i
[ 2019-04-23 22:26:52,803][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,829][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,860][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,890][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,922][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,923][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_i
[ 2019-04-23 22:26:52,966][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,000][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,030][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,063][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,095][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,096][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_i
[ 2019-04-23 22:26:53,098][cascade_classifier.calc_accuracy] Accuracy(layer_5 - train.classifie
[ 2019-04-23 22:26:53,100][cascade_classifier.fit_transform] [Result][Optimal Level Detected] c
```

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

Training complete in 0m 8.6446s


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

[ 2019-04-23 22:26:53,147][cascade_classifier.transform] X_groups_test.shape=[(587, 4454)]
[ 2019-04-23 22:26:53,173][cascade_classifier.transform] group_dims=[4454]
[ 2019-04-23 22:26:53,175][cascade_classifier.transform] X_test.shape=(587, 4454)
[ 2019-04-23 22:26:53,208][cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
[ 2019-04-23 22:26:53,311][cascade_classifier.transform] [layer=1] look_indexs=[0], X_cur_test
[ 2019-04-23 22:26:53,373][cascade_classifier.transform] [layer=2] look_indexs=[0], X_cur_test


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

Out[48]:           Precision    Recall  Accuracy        F1 Training Time (s)
         SVC        0.706840  0.718543  0.701874  0.712644         0m 0.1014s
         KN         0.986547  0.728477  0.855196  0.838095         0m 0.0011s
         NB         0.904153  0.937086  0.916525  0.920325         0m 0.0009s
         DT         0.937086  0.937086  0.935264  0.937086         0m 0.0247s
         LR         0.952703  0.933775  0.942078  0.943144         0m 0.0045s
         RF         0.979239  0.937086  0.957411  0.957699         0m 3.3739s
         AdaBoost   0.942953  0.930464  0.935264  0.936667         0m 0.4322s
         BgC        0.958904  0.927152  0.942078  0.942761         0m 0.2488s
         ETC        0.968966  0.930464  0.948893  0.949324         0m 0.1476s
         LSTM       0.513652  0.996689  0.512777  0.677928       3m 20.1492s
         DCF        0.957096  0.960265  0.957411  0.958678         0m 8.6446s

In [49]: df.plot(kind='bar', y="Accuracy", ylim=(0.5,1.0), figsize=(11,6), align='center', colo
         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-v2.eps")
         plt.show()
```



In [50]:
```
import pickle
# dump
with open("../pkl/ytb-gc-v2.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 [ ]:

In [ ]: