youtube-spam-v2-stop

April 23, 2019

V2: + Delete the stop words + 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  \
        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      LneaDw26bFvPh9xBHNw1btQoyP60ay_WWthtvXCx37s    2   760    760

                                                content          \
                                      top freq   count unique
         label
         0      2013-10-05T00:57:25.078000    2     951    919
         1             2014-08-30T11:00:35    1    1005    841


                                              top freq
         label
         0                            Shakira :-*    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()
```

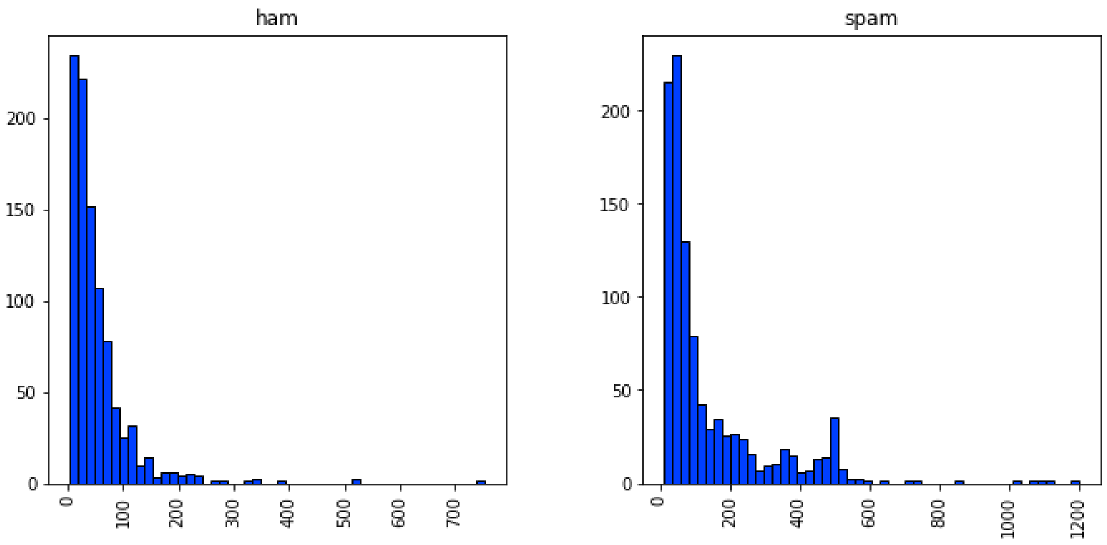
```
Out[14]:                                      COMMENT_ID            AUTHOR  \
         0  LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU         Julius NM
```

```
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('eng

             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, 4185)
```

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
```

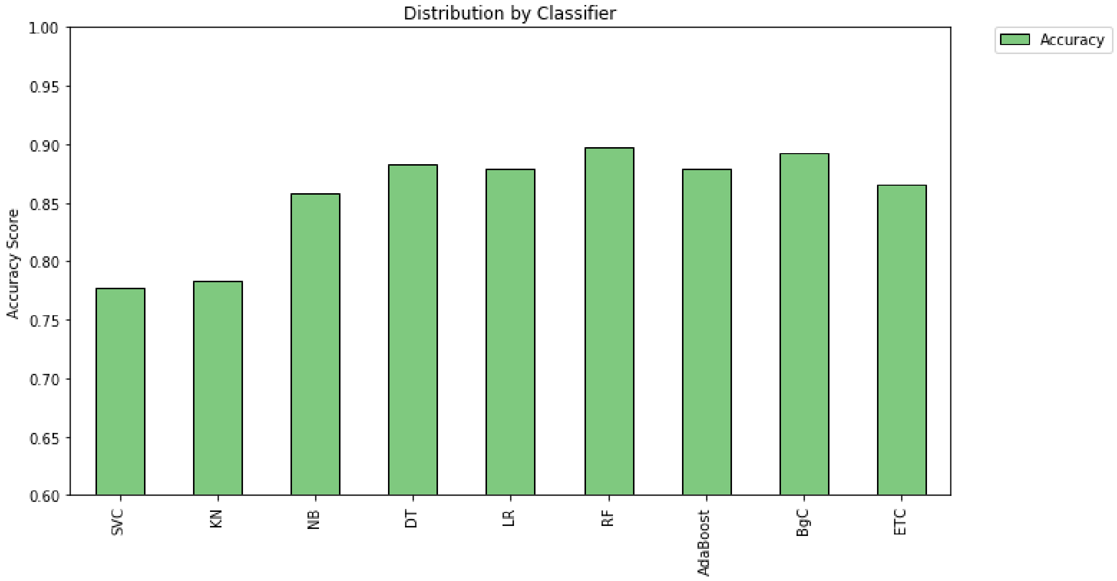
```
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: Data
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversio
  from ipykernel import kernelapp as app
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/ipykernel/__main__.py:2: DataConversio
  from ipykernel import kernelapp as app
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: Data
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/utils/validation.py:578: Data
  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        0.776699  0.794702  0.776831  0.785597         0m 0.0454s
         KN         0.983425  0.589404  0.783646  0.737060         0m 0.0018s
         NB         0.886926  0.831126  0.858603  0.858120         0m 0.0012s
         DT         0.936330  0.827815  0.882453  0.878735         0m 0.0391s
         LR         0.956522  0.801325  0.879046  0.872072         0m 0.0033s
         RF         0.972656  0.824503  0.897785  0.892473         0m 4.6762s
         AdaBoost   0.952941  0.804636  0.879046  0.872531         0m 0.5409s
         BgC        0.961390  0.824503  0.892675  0.887701         0m 0.2796s
         ETC        0.951417  0.778146  0.865417  0.856102         0m 0.1635s
```

```
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-stop.eps")
         plt.show()
```



0.0.3 RNN

Define the RNN structure.

```
In [37]: 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 [38]: max_words = features_train.shape[0]
         max_len = features_train.shape[1]
```

```
In [39]: 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 [40]: model = RNN()
         model.summary()
         model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 4185)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 4185, 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 [41]: 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 [==============================] - 91s 83ms/step - loss: 0.6942 - acc: 0.4849 - val_
Epoch 2/10
1095/1095 [==============================] - 71s 65ms/step - loss: 0.6935 - acc: 0.4868 - val_
```

```
In [42]: print('Training complete in {:.0f}m {:.4f}s'.format(
                 time_elapsed // 60, time_elapsed % 60))
```

```
Training complete in 2m 43.7192s
```

```
In [43]: pred = (np.asarray(model.predict(features_test, batch_size=128))).round()

In [44]: pred_scores.append(("LSTM", [precision_score(labels_test,pred), recall_score(labels_t
```

0.0.4 gcForest

```
In [45]: import sys
         sys.path.append("..")
         from gcforest.gcforest import GCForest
         from gcforest.utils.config_utils import load_json

In [46]: 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 [47]: 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 14:55:29,006][cascade_classifier.fit_transform] X_groups_train.shape=[(1369, 4185
[ 2019-04-23 14:55:29,036][cascade_classifier.fit_transform] group_dims=[4185]
[ 2019-04-23 14:55:29,038][cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-23 14:55:29,040][cascade_classifier.fit_transform] group_ends=[4185]
[ 2019-04-23 14:55:29,041][cascade_classifier.fit_transform] X_train.shape=(1369, 4185),X_test
[ 2019-04-23 14:55:29,081][cascade_classifier.fit_transform] [layer=0] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:29,698][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:30,247][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:30,744][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:31,283][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:31,722][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:31,723][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 14:55:31,750][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,788][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,819][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,852][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,885][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,887][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 14:55:31,923][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:31,956][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:31,987][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:32,017][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:32,050][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:32,052][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 14:55:32,054][cascade_classifier.calc_accuracy] Accuracy(layer_0 - train.classifi
[ 2019-04-23 14:55:32,074][cascade_classifier.fit_transform] [layer=1] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:32,240][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:32,395][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:32,544][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:32,699][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:32,941][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:32,942][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 14:55:33,008][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,060][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,123][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,179][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,246][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,250][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 14:55:33,305][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,357][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,405][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,428][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,469][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,470][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_
[ 2019-04-23 14:55:33,471][cascade_classifier.calc_accuracy] Accuracy(layer_1 - train.classifi
[ 2019-04-23 14:55:33,486][cascade_classifier.fit_transform] [layer=2] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:33,672][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:33,826][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:33,981][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:34,189][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:34,349][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:34,350][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 14:55:34,375][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,421][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,445][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,493][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,547][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,551][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 14:55:34,602][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,654][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,698][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,743][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,769][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,770][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_
[ 2019-04-23 14:55:34,773][cascade_classifier.calc_accuracy] Accuracy(layer_2 - train.classifi
[ 2019-04-23 14:55:34,792][cascade_classifier.fit_transform] [layer=3] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:34,975][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,159][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,330][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,520][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,690][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,691][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 14:55:35,715][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,756][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,779][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,822][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,845][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,846][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 14:55:35,885][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:35,906][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:35,932][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:35,963][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:35,996][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:35,997][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 14:55:36,000][cascade_classifier.calc_accuracy] Accuracy(layer_3 - train.classifi
[ 2019-04-23 14:55:36,014][cascade_classifier.fit_transform] [layer=4] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:36,166][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:36,333][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:36,582][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:36,756][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:37,073][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:37,074][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 14:55:37,100][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,142][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,175][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,225][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,274][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,276][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 14:55:37,302][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,345][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,377][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,425][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,472][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,473][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 14:55:37,475][cascade_classifier.calc_accuracy] Accuracy(layer_4 - train.classifi
[ 2019-04-23 14:55:37,490][cascade_classifier.fit_transform] [layer=5] look_indexs=[0], X_cur_
[ 2019-04-23 14:55:37,620][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:37,803][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:37,953][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:38,075][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:38,215][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:38,216][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 14:55:38,241][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,290][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,311][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,350][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,373][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,374][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 14:55:38,407][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,437][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,467][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,499][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,534][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,536][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 14:55:38,537][cascade_classifier.calc_accuracy] Accuracy(layer_5 - train.classifi
[ 2019-04-23 14:55:38,539][cascade_classifier.fit_transform] [Result][Optimal Level Detected]
```

```
In [48]: print('Training complete in {:.0f}m {:.4f}s'.format(
                 time_elapsed // 60, time_elapsed % 60))
```

```
Training complete in 0m 9.5503s
```

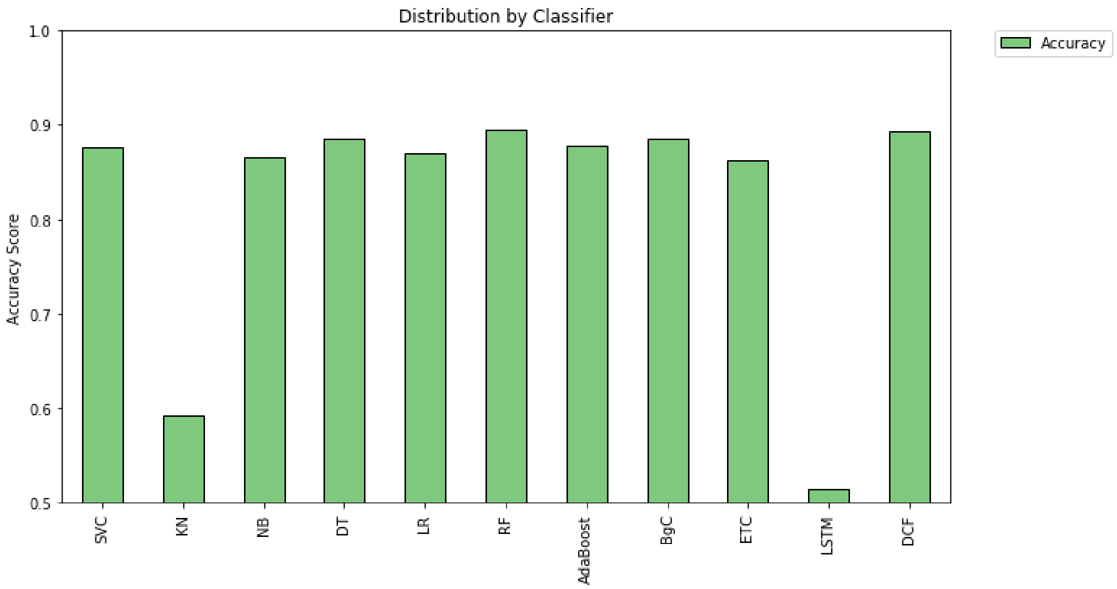
```
In [49]: pred = predict_labels(gc,features_test.toarray())
         pred_scores.append(("DCF", [precision_score(labels_test,pred), recall_score(labels_tes
```

```
[ 2019-04-23 14:55:38,577][cascade_classifier.transform] X_groups_test.shape=[(587, 4185)]
[ 2019-04-23 14:55:38,599][cascade_classifier.transform] group_dims=[4185]
[ 2019-04-23 14:55:38,600][cascade_classifier.transform] X_test.shape=(587, 4185)
[ 2019-04-23 14:55:38,617][cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
[ 2019-04-23 14:55:38,714][cascade_classifier.transform] [layer=1] look_indexs=[0], X_cur_test
[ 2019-04-23 14:55:38,778][cascade_classifier.transform] [layer=2] look_indexs=[0], X_cur_test
```

```
In [50]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall
         df
```

```
Out[50]:           Precision    Recall  Accuracy        F1 Training Time (s)
         SVC        0.963563  0.788079  0.875639  0.867031        0m 0.1015s
         KN         0.984615  0.211921  0.592845  0.348774        0m 0.0056s
         NB         0.896797  0.834437  0.865417  0.864494        0m 0.0011s
         DT         0.950192  0.821192  0.885860  0.880995        0m 0.0646s
         LR         0.966942  0.774834  0.870528  0.860294        0m 0.0032s
         RF         0.976190  0.814570  0.894378  0.888087        0m 4.5514s
         AdaBoost   0.949219  0.804636  0.877342  0.870968        0m 0.4722s
         BgC        0.975709  0.798013  0.885860  0.877960        0m 0.3183s
         ETC        0.940239  0.781457  0.862010  0.853526        0m 0.1676s
         LSTM       0.514480  1.000000  0.514480  0.679415       2m 43.7192s
         DCF        0.944238  0.841060  0.892675  0.889667        0m 9.5503s
```

```
In [54]: 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-stop.eps")
         plt.show()
```



```
In [52]: import pickle
         # dump
         with open("../pkl/ytb-gc-v2-stop.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 [ ]:
```