youtube-spam-v3

April 23, 2019

V3: + LSTM uses the Tokenizert.fit_on_texts(data) then Tokenizert.texts_to_sequences(data) to do the preprocessing + Other 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 CountVectorizer
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	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

```
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]: # Checking for Consistent Column Name
         df.columns

Out[11]: Index(['COMMENT_ID', 'AUTHOR', 'DATE', 'CONTENT', 'CLASS'], dtype='object')

In [12]: # Checking for Datatypes
         df.dtypes

Out[12]: COMMENT_ID    object
         AUTHOR        object
         DATE          object
         CONTENT       object
         CLASS          int64
         dtype: object

In [13]: # Check for missing nan
         df.isnull().isnull().sum()

Out[13]: 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 [14]: ytb = df[["CONTENT","CLASS"]]
         ytb = df.rename(columns = {'CONTENT':'content','CLASS':'label'})
```

Let's look into our data

```
In [15]: ytb.groupby('label').describe()

Out[15]:       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         \
```

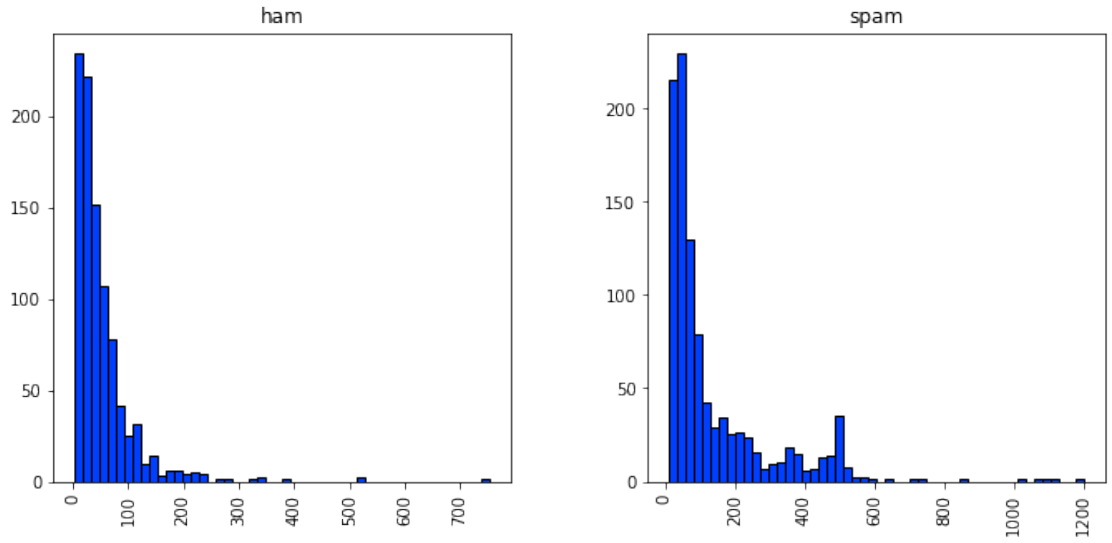
```
                                  top freq   count unique
label
0      2013-10-05T00:57:25.078000    2     951    919
1            2014-09-10T22:58:23     1    1005    841


                                       top freq
label
0                                      wow    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 [16]: ytb['length'] = ytb['content'].apply(len)

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

```
Out[16]:                                    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 [17]: 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 [18]: text_feat = ytb['content'].copy()
```

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

```
In [19]: # 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 [20]: # text_feat = text_feat.apply(text_process)
```

```
In [21]: vectorizer = TfidfVectorizer("english")
```

```
In [22]: features = vectorizer.fit_transform(text_feat)
```

```
In [23]: labels = LabelEncoder().fit_transform(ytb['label'])
         labels = labels.reshape(-1,1)
```

```
In [24]: text_feat.shape
```

```
Out[24]: (1956,)
```

```
In [25]: features.shape
```

```
Out[25]: (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 [26]: 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, recall_score, accuracy_score, 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 [27]: 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 [28]: features_train, features_test, labels_train, labels_test = train_test_split(features,
```

```
In [29]: 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 [30]: def train_classifier(clf, feature_train, labels_train):
             clf.fit(feature_train, labels_train)
```

```
In [31]: def predict_labels(clf, features):
             return (clf.predict(features))
```

Now iterate through classifiers and save the results

```
In [32]: import time
```

```
In [33]: 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_tes
```

```
/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
```

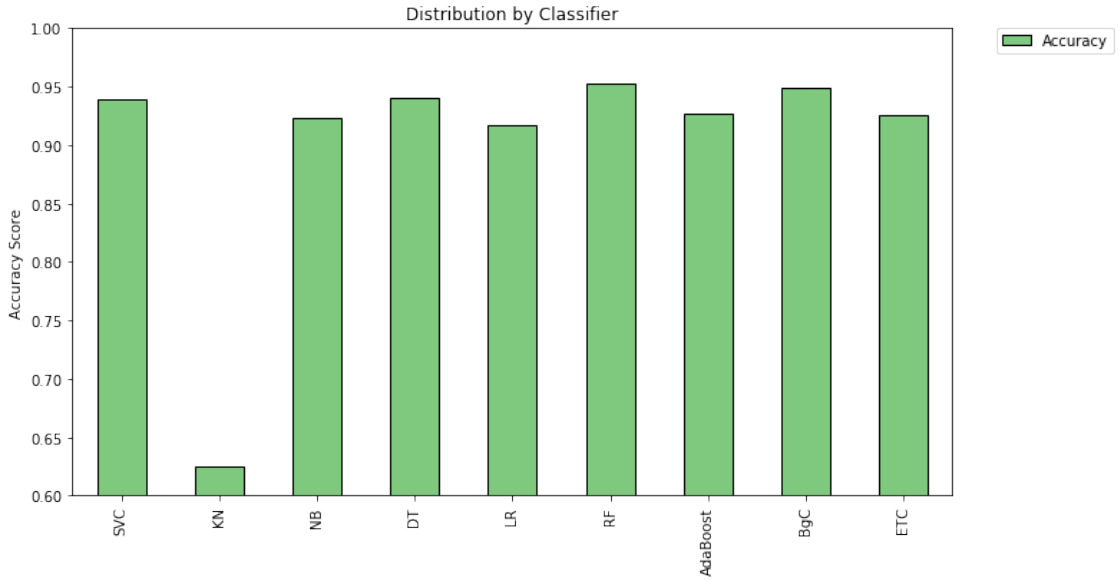
```
In [34]: # pred_scores
```

```
In [35]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall
         df
```

Out[35]:

	Precision	Recall	Accuracy	F1	Training Time (s)
SVC	0.958621	0.920530	0.938671	0.939189	0m 0.1067s
KN	0.988095	0.274834	0.625213	0.430052	0m 0.0014s
NB	0.907937	0.947020	0.923339	0.927066	0m 0.0010s
DT	0.958763	0.923841	0.940375	0.940978	0m 0.0329s
LR	0.953405	0.880795	0.916525	0.915663	0m 0.0040s
RF	0.979021	0.927152	0.952300	0.952381	0m 3.4356s
AdaBoost	0.941980	0.913907	0.926746	0.927731	0m 0.4562s
BgC	0.972222	0.927152	0.948893	0.949153	0m 0.2124s
ETC	0.944828	0.907285	0.925043	0.925676	0m 0.1247s

```
In [36]: 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-v3.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.
```

0.0.4 Process the data

- Tokenize the data and convert the text to sequences.
- Add padding to ensure that all the sequences have the same shape.
- There are many ways of taking the *max_len* and here an arbitrary length of 500 is chosen. (From the Fig, almost all the sentences have the length < 500)

```
In [38]: features_lstm = text_feat
         labels_lstm = labels
```

```
In [39]: max_words = 1000
         max_len = 500 # n_features
         tok = Tokenizer(num_words=max_words)
```

```
        tok.fit_on_texts(features_lstm)
        sequences = tok.texts_to_sequences(features_lstm)
        features_lstm = sequence.pad_sequences(sequences,maxlen=max_len)
```

```
In [40]: features_lstm.shape
```

```
Out[40]: (1956, 500)
```

```
In [41]: labels_lstm.shape
```

```
Out[41]: (1956, 1)
```

```
In [42]: features_lstm_train, features_lstm_test, labels_lstm_train, labels_lstm_test = train_t
```

```
In [43]: 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 [44]: model = RNN()
         model.summary()
         model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 500)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 500, 50)           50000
_________________________________________________________________
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: 136,513
Trainable params: 136,513
Non-trainable params: 0
_________________________________________________________________
```

```
In [45]: since = time.time()


         model.fit(features_lstm_train, labels_lstm_train, epochs=10, batch_size=128,validatio
                             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 [==============================] - 9s 8ms/step - loss: 0.6675 - acc: 0.6447 - val_los
Epoch 2/10
1095/1095 [==============================] - 8s 7ms/step - loss: 0.5575 - acc: 0.8073 - val_los
Epoch 3/10
1095/1095 [==============================] - 7s 7ms/step - loss: 0.3036 - acc: 0.9059 - val_los
Epoch 4/10
1095/1095 [==============================] - 7s 7ms/step - loss: 0.1928 - acc: 0.9379 - val_los
Epoch 5/10
1095/1095 [==============================] - 7s 7ms/step - loss: 0.1250 - acc: 0.9589 - val_los
Epoch 6/10
1095/1095 [==============================] - 7s 7ms/step - loss: 0.0891 - acc: 0.9717 - val_los
```

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

```
Training complete in 0m 47.4993s
```

```
In [47]: pred = (np.asarray(model.predict(features_lstm_test, batch_size=128))).round()

In [48]: pred_scores.append(("LSTM", [precision_score(labels_lstm_test,pred), recall_score(labe
```

0.0.5 gcForest

```
In [49]: import sys
         sys.path.append("..")
         from gcforest.gcforest import GCForest
         from gcforest.utils.config_utils import load_json

In [50]: 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 [51]: 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

         # gc.fit_transform(features_train, labels_train, features_test, labels_test)

[ 2019-04-23 23:12:18,984][cascade_classifier.fit_transform] X_groups_train.shape=[(1369, 4454)
[ 2019-04-23 23:12:19,036][cascade_classifier.fit_transform] group_dims=[4454]
[ 2019-04-23 23:12:19,038][cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-23 23:12:19,039][cascade_classifier.fit_transform] group_ends=[4454]
[ 2019-04-23 23:12:19,040][cascade_classifier.fit_transform] X_train.shape=(1369, 4454),X_test
[ 2019-04-23 23:12:19,083][cascade_classifier.fit_transform] [layer=0] look_indexs=[0], X_cur_t
[ 2019-04-23 23:12:19,346][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:19,609][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:19,887][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:20,164][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:20,430][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:20,431][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_i
[ 2019-04-23 23:12:20,459][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,495][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,534][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,570][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,605][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,607][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_i
[ 2019-04-23 23:12:20,650][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 23:12:20,688][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 23:12:20,722][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 23:12:20,756][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 23:12:20,790][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
```

```
[ 2019-04-23 23:12:20,791][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_i
[ 2019-04-23 23:12:20,793][cascade_classifier.calc_accuracy] Accuracy(layer_0 - train.classifie
[ 2019-04-23 23:12:20,849][cascade_classifier.fit_transform] [layer=1] look_indexs=[0], X_cur_t
[ 2019-04-23 23:12:20,967][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,102][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,276][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,421][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,551][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,552][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_i
[ 2019-04-23 23:12:21,579][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,618][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,655][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,692][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,718][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,721][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_i
[ 2019-04-23 23:12:21,760][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,797][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,835][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,869][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,904][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,906][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_2 - 5_i
[ 2019-04-23 23:12:21,908][cascade_classifier.calc_accuracy] Accuracy(layer_1 - train.classifie
[ 2019-04-23 23:12:21,925][cascade_classifier.fit_transform] [layer=2] look_indexs=[0], X_cur_t
[ 2019-04-23 23:12:22,067][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,205][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,357][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,475][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,591][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,592][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_i
[ 2019-04-23 23:12:22,621][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,666][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,706][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,741][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,768][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,773][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_i
[ 2019-04-23 23:12:22,811][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,843][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,876][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,909][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,951][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,952][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_2 - 5_i
[ 2019-04-23 23:12:22,954][cascade_classifier.calc_accuracy] Accuracy(layer_2 - train.classifie
[ 2019-04-23 23:12:22,972][cascade_classifier.fit_transform] [layer=3] look_indexs=[0], X_cur_t
[ 2019-04-23 23:12:23,114][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 23:12:23,233][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 23:12:23,412][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 23:12:23,559][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
[ 2019-04-23 23:12:23,711][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_i
```

```
[ 2019-04-23 23:12:23,712][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 23:12:23,736][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,766][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,798][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,838][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,861][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,862][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 23:12:23,900][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:23,922][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:23,953][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:23,984][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:24,015][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:24,016][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_2 - 5_
[ 2019-04-23 23:12:24,017][cascade_classifier.calc_accuracy] Accuracy(layer_3 - train.classifi
[ 2019-04-23 23:12:24,037][cascade_classifier.fit_transform] [layer=4] look_indexs=[0], X_cur_
[ 2019-04-23 23:12:24,175][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,331][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,454][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,575][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,703][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,705][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_0 - 5_
[ 2019-04-23 23:12:24,728][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,771][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,793][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,834][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,856][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,857][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_1 - 5_
[ 2019-04-23 23:12:24,888][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:24,918][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:24,949][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:24,982][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:25,014][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:25,015][kfold_wrapper.log_eval_metrics] Accuracy(layer_4 - estimator_2 - 5_
[ 2019-04-23 23:12:25,016][cascade_classifier.calc_accuracy] Accuracy(layer_4 - train.classifi
[ 2019-04-23 23:12:25,038][cascade_classifier.fit_transform] [layer=5] look_indexs=[0], X_cur_
[ 2019-04-23 23:12:25,210][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,353][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,486][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,619][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,774][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,777][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_0 - 5_
[ 2019-04-23 23:12:25,800][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,832][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,864][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,896][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,938][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,939][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_
[ 2019-04-23 23:12:25,963][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
```

```
[ 2019-04-23 23:12:25,998][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 23:12:26,021][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 23:12:26,046][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 23:12:26,080][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 23:12:26,082][kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_
[ 2019-04-23 23:12:26,083][cascade_classifier.calc_accuracy] Accuracy(layer_5 - train.classifi
[ 2019-04-23 23:12:26,104][cascade_classifier.fit_transform] [layer=6] look_indexs=[0], X_cur_
[ 2019-04-23 23:12:26,291][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,422][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,619][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,811][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,943][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,944][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_
[ 2019-04-23 23:12:26,967][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:26,999][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:27,032][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:27,064][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:27,096][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:27,097][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_
[ 2019-04-23 23:12:27,139][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,169][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,200][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,232][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,264][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,265][kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_
[ 2019-04-23 23:12:27,266][cascade_classifier.calc_accuracy] Accuracy(layer_6 - train.classifi
[ 2019-04-23 23:12:27,267][cascade_classifier.fit_transform] [Result][Optimal Level Detected] c
```

```
In [52]: print('Training complete in {:.0f}m {:.4f}s'.format(
                 time_elapsed // 60, time_elapsed % 60))

Training complete in 0m 8.3085s
```

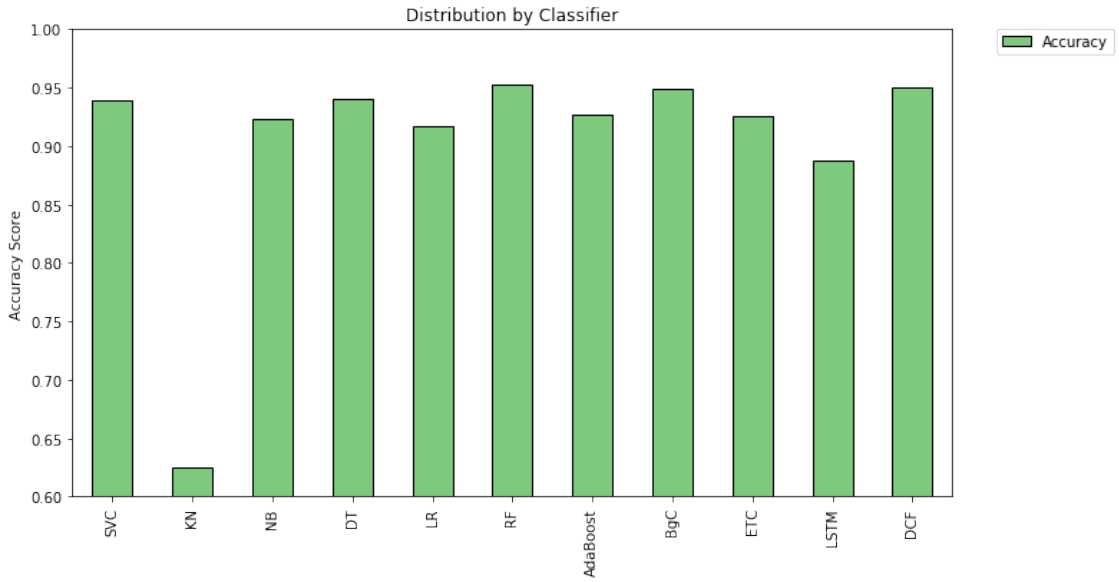
```
In [53]: pred = predict_labels(gc,features_test.toarray())
         pred_scores.append(("DCF", [precision_score(labels_test,pred), recall_score(labels_tes

[ 2019-04-23 23:12:27,325][cascade_classifier.transform] X_groups_test.shape=[(587, 4454)]
[ 2019-04-23 23:12:27,348][cascade_classifier.transform] group_dims=[4454]
[ 2019-04-23 23:12:27,349][cascade_classifier.transform] X_test.shape=(587, 4454)
[ 2019-04-23 23:12:27,373][cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
[ 2019-04-23 23:12:27,463][cascade_classifier.transform] [layer=1] look_indexs=[0], X_cur_test
[ 2019-04-23 23:12:27,526][cascade_classifier.transform] [layer=2] look_indexs=[0], X_cur_test
[ 2019-04-23 23:12:27,587][cascade_classifier.transform] [layer=3] look_indexs=[0], X_cur_test
```

```
In [54]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall
         df
```

Out[54]:

	Precision	Recall	Accuracy	F1	Training Time (s)
SVC	0.958621	0.920530	0.938671	0.939189	0m 0.1067s
KN	0.988095	0.274834	0.625213	0.430052	0m 0.0014s
NB	0.907937	0.947020	0.923339	0.927066	0m 0.0010s
DT	0.958763	0.923841	0.940375	0.940978	0m 0.0329s
LR	0.953405	0.880795	0.916525	0.915663	0m 0.0040s
RF	0.979021	0.927152	0.952300	0.952381	0m 3.4356s
AdaBoost	0.941980	0.913907	0.926746	0.927731	0m 0.4562s
BgC	0.972222	0.927152	0.948893	0.949153	0m 0.2124s
ETC	0.944828	0.907285	0.925043	0.925676	0m 0.1247s
LSTM	0.960938	0.814570	0.887564	0.881720	0m 47.4993s
DCF	0.953488	0.950331	0.950596	0.951907	0m 8.3085s

In [55]:
```
df.plot(kind='bar', y="Accuracy", ylim=(0.6,1.0), 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-v3.eps")
plt.show()
```



In [56]:
```
import pickle
# dump
with open("../pkl/ytb-gc-v3.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 [ ]: