

# youtube-spam-v3-stop

April 23, 2019

V3: + Delete the stop words + LSTM uses the Tokenizer.fit\_on\_texts(data) then Tokenizer.texts\_to\_sequences(data) to do the preprocessing + Other models uses the TfidfVectorizer to do 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	\
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]: # 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-m4ELMJvx0QBmBOFHqG0k0	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	2015-02-17T04:25:01.940000	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 [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-qCcZlY8	Evgeny Murashkin	
3	z13jhp0bxqncu512g22wvzkasxmvvzjaz04	ElNino Melendez	
4	z13fbwbp1oujthgqj04chlmgpvzmtt3r3dw	GsMega	

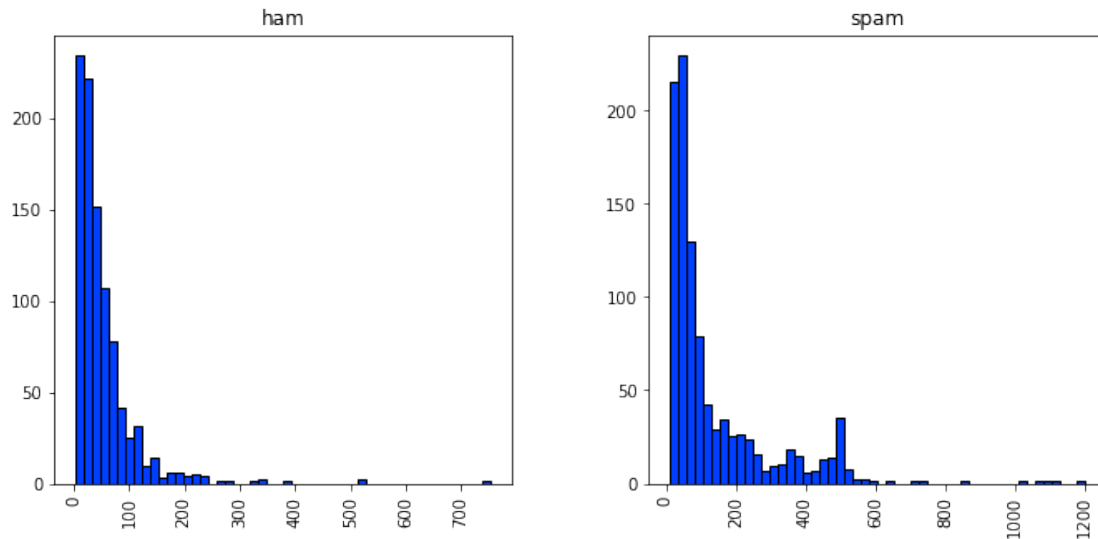
  

	DATE	content	\
0	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	
1	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	
2	2013-11-08T17:34:21	just for test I have to say murdev.com	
3	2013-11-09T08:28:43	me shaking my sexy ass on my channel enjoy ^_^	
4	2013-11-10T16:05:38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

    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, 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 iterate 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:2
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_test, pred)]))

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

```

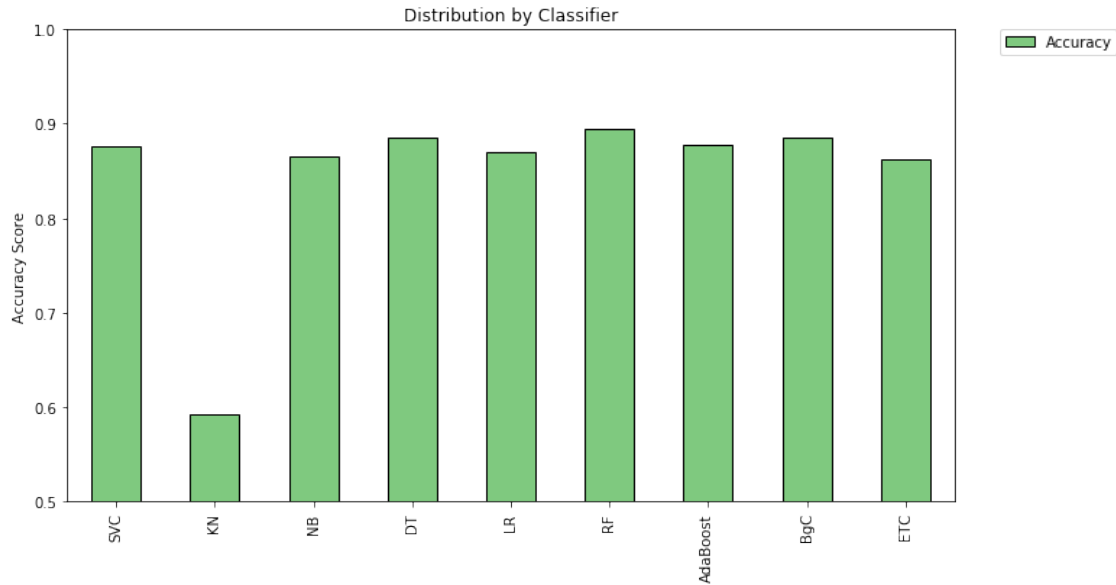
```
In [34]: # pred_scores
```

```
In [35]: df = pd.DataFrame.from_items(pred_scores, orient='index', columns=['Precision', 'Recall', 'Accuracy', 'F1', 'Training Time (s)'])
df
```

```
Out [35]:
```

	Precision	Recall	Accuracy	F1	Training Time (s)
SVC	0.963563	0.788079	0.875639	0.867031	0m 0.0908s
KN	0.984615	0.211921	0.592845	0.348774	0m 0.0008s
NB	0.896797	0.834437	0.865417	0.864494	0m 0.0010s
DT	0.950192	0.821192	0.885860	0.880995	0m 0.0481s
LR	0.966942	0.774834	0.870528	0.860294	0m 0.0030s
RF	0.976190	0.814570	0.894378	0.888087	0m 4.3453s
AdaBoost	0.949219	0.804636	0.877342	0.870968	0m 0.4404s
BgC	0.975709	0.798013	0.885860	0.877960	0m 0.2903s
ETC	0.940239	0.781457	0.862010	0.853526	0m 0.1577s

```
In [37]: df.plot(kind='bar', y="Accuracy", ylim=(0.5,1.0), figsize=(11,6), align='center', color='red')
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-stop.eps")
plt.show()
```



### 0.03 RNN

Define the RNN structure.

```
In [38]: 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.

### 0.04 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 [39]: features_lstm = text_feat
         labels_lstm = labels
```

```
In [40]: 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 [41]: features_lstm.shape
```

```
Out[41]: (1956, 500)
```

```
In [42]: labels_lstm.shape
```

```
Out[42]: (1956, 1)
```

```
In [43]: features_lstm_train, features_lstm_test, labels_lstm_train, labels_lstm_test = train_test_split(features_lstm, labels_lstm, test_size=0.1, random_state=42)
```

```
In [44]: 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 [45]: 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 [46]: since = time.time()
```

```
model.fit(features_lstm_train, labels_lstm_train, epochs=10, batch_size=128, validation_data=(features_lstm_test, labels_lstm_test),
          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: 0.6709 - acc: 0.6447 - val_loss: 0.5578
Epoch 2/10
1095/1095 [=====] - 9s 8ms/step - loss: 0.5578 - acc: 0.7653 - val_loss: 0.3865
Epoch 3/10
1095/1095 [=====] - 9s 8ms/step - loss: 0.3865 - acc: 0.8667 - val_loss: 0.2837
Epoch 4/10
1095/1095 [=====] - 11s 10ms/step - loss: 0.2837 - acc: 0.8995 - val_loss: 0.2243
Epoch 5/10
1095/1095 [=====] - 11s 10ms/step - loss: 0.2243 - acc: 0.9242 - val_loss: 0.1810
Epoch 6/10
1095/1095 [=====] - 9s 8ms/step - loss: 0.1810 - acc: 0.9269 - val_loss: 0.1474
Epoch 7/10
1095/1095 [=====] - 12s 11ms/step - loss: 0.1474 - acc: 0.9397 - val_loss: 0.1474
```

```
In [47]: print('Training complete in {:.0f}m {:.4f}s'.format(
          time_elapsed // 60, time_elapsed % 60))
```

```
Training complete in 1m 10.3903s
```

```
In [48]: pred = (np.asarray(model.predict(features_lstm_test, batch_size=128))).round()
```

```
In [49]: pred_scores.append(("LSTM", [precision_score(labels_lstm_test, pred), recall_score(labels_lstm_test, pred)]))
```

## 0.0.5 gcForest

```
In [50]: import sys
          sys.path.append("..")
          from gcforest.gcforest import GCForest
          from gcforest.utils.config_utils import load_json
```

```

In [51]: 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 [52]: 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 23:15:28,820] [cascade_classifier.fit_transform] X_groups_train.shape=[(1369, 4185)
[ 2019-04-23 23:15:28,869] [cascade_classifier.fit_transform] group_dims=[4185]
[ 2019-04-23 23:15:28,873] [cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-23 23:15:28,882] [cascade_classifier.fit_transform] group_ends=[4185]
[ 2019-04-23 23:15:28,885] [cascade_classifier.fit_transform] X_train.shape=(1369, 4185),X_test
[ 2019-04-23 23:15:28,938] [cascade_classifier.fit_transform] [layer=0] look_indexes=[0], X_cur
[ 2019-04-23 23:15:29,422] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:29,830] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:30,235] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:30,890] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:31,831] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:31,833] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 23:15:31,876] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:31,920] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:31,957] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:32,001] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:32,023] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:32,025] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 23:15:32,050] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_
[ 2019-04-23 23:15:32,094] [kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_2 - 5_

```

[illegible]



```

[ 2019-04-23 23:15:38,181] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_)
[ 2019-04-23 23:15:38,182] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_1 - 5_)
[ 2019-04-23 23:15:38,204] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,225] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,247] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,268] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,289] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,290] [kfold_wrapper.log_eval_metrics] Accuracy(layer_5 - estimator_2 - 5_)
[ 2019-04-23 23:15:38,291] [cascade_classifier.calc_accuracy] Accuracy(layer_5 - train.classifi
[ 2019-04-23 23:15:38,302] [cascade_classifier.fit_transform] [layer=6] look_indexs=[0], X_cur_t
[ 2019-04-23 23:15:38,433] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:38,564] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:38,702] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:38,867] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:39,014] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:39,015] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_0 - 5_)
[ 2019-04-23 23:15:39,038] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,067] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,089] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,111] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,137] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,139] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_1 - 5_)
[ 2019-04-23 23:15:39,171] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,199] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,224] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,245] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,266] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,267] [kfold_wrapper.log_eval_metrics] Accuracy(layer_6 - estimator_2 - 5_)
[ 2019-04-23 23:15:39,269] [cascade_classifier.calc_accuracy] Accuracy(layer_6 - train.classifi
[ 2019-04-23 23:15:39,270] [cascade_classifier.fit_transform] [Result] [Optimal Level Detected]

```

```

In [53]: print('Training complete in {:.0f}m {:.4f}s'.format(
           time_elapsed // 60, time_elapsed % 60))

```

Training complete in 0m 10.4687s

```

In [54]: pred = predict_labels(gc,features_test.toarray())
         pred_scores.append(("DCF", [precision_score(labels_test,pred), recall_score(labels_test,

```

```

[ 2019-04-23 23:15:39,306] [cascade_classifier.transform] X_groups_test.shape=[(587, 4185)]
[ 2019-04-23 23:15:39,322] [cascade_classifier.transform] group_dims=[4185]
[ 2019-04-23 23:15:39,323] [cascade_classifier.transform] X_test.shape=(587, 4185)
[ 2019-04-23 23:15:39,335] [cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
[ 2019-04-23 23:15:39,406] [cascade_classifier.transform] [layer=1] look_indexs=[0], X_cur_test
[ 2019-04-23 23:15:39,476] [cascade_classifier.transform] [layer=2] look_indexs=[0], X_cur_test
[ 2019-04-23 23:15:39,524] [cascade_classifier.transform] [layer=3] look_indexs=[0], X_cur_test

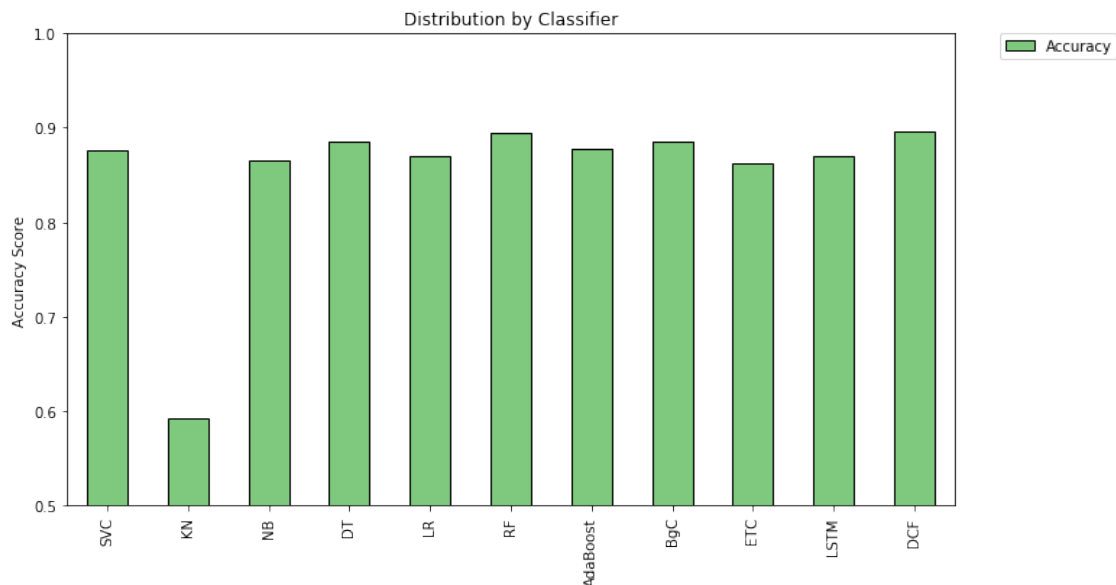
```

```
In [55]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall', 'Accuracy', 'F1', 'Training Time (s)'])
df
```

```
Out [55]:
```

	Precision	Recall	Accuracy	F1	Training Time (s)
SVC	0.963563	0.788079	0.875639	0.867031	0m 0.0908s
KN	0.984615	0.211921	0.592845	0.348774	0m 0.0008s
NB	0.896797	0.834437	0.865417	0.864494	0m 0.0010s
DT	0.950192	0.821192	0.885860	0.880995	0m 0.0481s
LR	0.966942	0.774834	0.870528	0.860294	0m 0.0030s
RF	0.976190	0.814570	0.894378	0.888087	0m 4.3453s
AdaBoost	0.949219	0.804636	0.877342	0.870968	0m 0.4404s
BgC	0.975709	0.798013	0.885860	0.877960	0m 0.2903s
ETC	0.940239	0.781457	0.862010	0.853526	0m 0.1577s
LSTM	0.889655	0.854305	0.870528	0.871622	1m 10.3903s
DCF	0.958175	0.834437	0.896082	0.892035	0m 10.4687s

```
In [56]: df.plot(kind='bar', y="Accuracy", ylim=(0.5,1.0), figsize=(11,6), align='center', color='green')
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-stop.eps")
plt.show()
```



```
In [57]: import pickle
# dump
with open("../pkl/ytb-gc-v3-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 [ ]: