sms-spam-v4-stop

April 23, 2019

V4: + Delete the stop words + LSTM uses the Tokenizert.fit_on_texts(data) then Tokenizert.texts_to_sequences(data) to do the preprocessing + Other models uses the CountVectorizer to do the the preprocessing

Goal of this notebook to test several classifiers on the data set with different features

0.0.1 Let's begin

First of all neccesary imports

```
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
```

Let's read the data from csv file

```
In [2]: sms = pd.read_csv('../data/sms-spam.csv',delimiter=',',encoding='latin-1')

        sms.head()

Out[2]:      v1                                                 v2 Unnamed: 2  \
        0   ham  Go until jurong point, crazy.. Available only ...        NaN
        1   ham                      Ok lar... Joking wif u oni...        NaN
        2  spam  Free entry in 2 a wkly comp to win FA Cup fina...        NaN
        3   ham  U dun say so early hor... U c already then say...        NaN
        4   ham  Nah I don't think he goes to usf, he lives aro...        NaN

          Unnamed: 3 Unnamed: 4
        0        NaN        NaN
        1        NaN        NaN
        2        NaN        NaN
```

```
3          NaN          NaN
4          NaN          NaN
```

Now drop "unnamed" columns and rename v1 and v2 to "label" and "message"

```
In [3]: sms = sms.drop(['Unnamed: 2','Unnamed: 3','Unnamed: 4'],axis=1)
        sms = sms.rename(columns = {'v1':'label','v2':'message'})
```

Let's look into our data

```
In [4]: sms.groupby('label').describe()
```

```
Out[4]:          message
                 count unique                                  top freq
         label
         ham     4825   4516                 Sorry, I'll call later   30
         spam     747    653  Please call our customer service representativ...    4
```

Intresting that "Sorry, I'll call later" appears only 30 times here =)
Now let's create new feature "message length" and plot it to see if it's of any interest

```
In [5]: sms['length'] = sms['message'].apply(len)
        sms.head()
```

```
Out[5]:  label                                  message  length
         0    ham  Go until jurong point, crazy.. Available only ...     111
         1    ham                      Ok lar... Joking wif u oni...      29
         2   spam  Free entry in 2 a wkly comp to win FA Cup fina...     155
         3    ham  U dun say so early hor... U c already then say...      49
         4    ham  Nah I don't think he goes to usf, he lives aro...      61
```
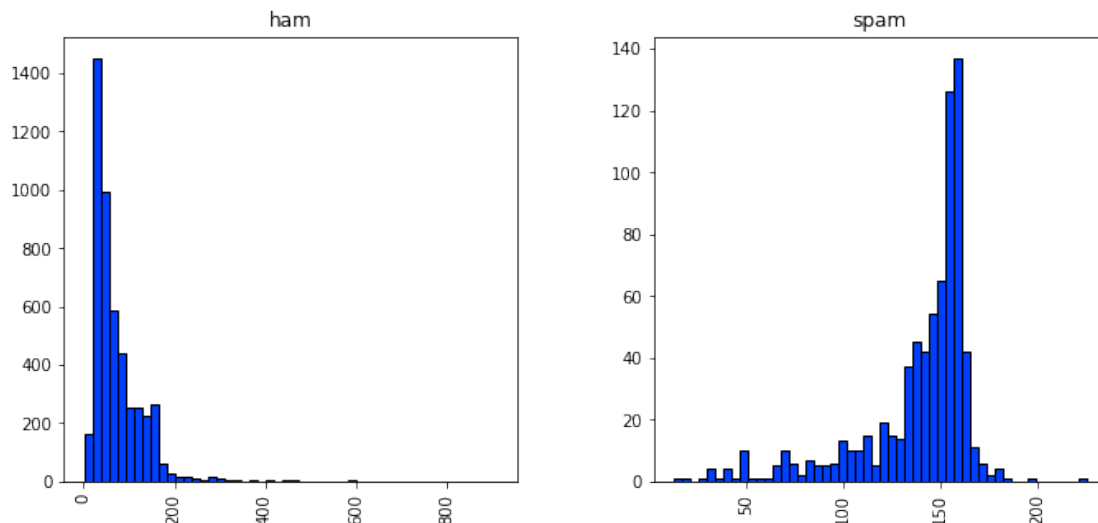
```
In [6]: mpl.rcParams['patch.force_edgecolor'] = True
        plt.style.use('seaborn-bright')
        sms.hist(column='length', by='label', bins=50,figsize=(11,5))
        plt.savefig("../img/sms-length-distribution.eps")
        plt.show()
```

Looks like the lengthy is the message, more likely it is a spam. Let's not forget this

0.0.2 Text processing and vectorizing our meddages

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

```
In [7]: text_feat = sms['message'].copy()
```

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

```
In [8]: 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('engl:

            return " ".join(text)
```

```
In [9]: text_feat = text_feat.apply(text_process)
```

```
In [10]: vectorizer = CountVectorizer()
```

```
In [11]: features = vectorizer.fit_transform(text_feat)
```

```
In [12]: labels = LabelEncoder().fit_transform(sms['label'])
         labels = labels.reshape(-1,1)
```

```
In [13]: text_feat.shape
```

```
Out[13]: (5572,)
```

```
In [14]: features.shape
```

```
Out[14]: (5572, 9403)
```

0.0.3 Classifiers and predictions

First of all let's split our features to test and train set

```
In [15]: features_train, features_test, labels_train, labels_test = train_test_split(features,
```

Now let's import bunch of classifiers, initialize them and make a dictionary to itereate through

```
In [16]: 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
```

3

```
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/ensemble/weight_boosting.py:29
  from numpy.core.umath_tests import inner1d
```

```python
In [17]: svc = SVC(kernel='sigmoid', gamma=1.0)
         knc = KNeighborsClassifier(n_neighbors=49)
         mnb = MultinomialNB(alpha=0.2)
         dtc = DecisionTreeClassifier(min_samples_split=7, random_state=111)
         lrc = LogisticRegression(solver='liblinear', penalty='l1')
         rfc = RandomForestClassifier(n_estimators=31, random_state=111)
         abc = AdaBoostClassifier(n_estimators=62, random_state=111)
         bc = BaggingClassifier(n_estimators=9, random_state=111)
         etc = ExtraTreesClassifier(n_estimators=9, random_state=111)
```

```python
In [18]: 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

```python
In [19]: def train_classifier(clf, feature_train, labels_train):
             clf.fit(feature_train, labels_train)
```

```python
In [20]: def predict_labels(clf, features):
             return (clf.predict(features))
```

Now iterate through classifiers and save the results

```python
In [21]: import time
```

```python
In [22]: 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/sklearn/metrics/classification.py:1135
  'precision', 'predicted', average, warn_for)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135
  'precision', 'predicted', average, warn_for)
/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: 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
```
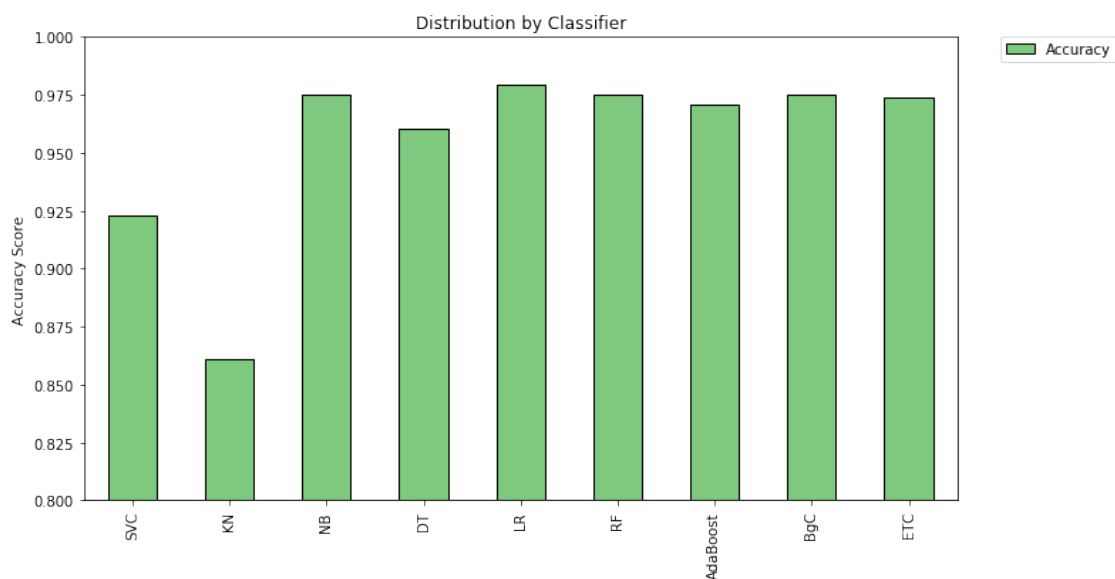
In [23]: *# pred_scores*

In [24]: df = pd.DataFrame.from_items(pred_scores,orient='index', columns=['Precision', 'Recall
         df

Out[24]:   Precision   Recall Accuracy   F1 Training Time (s)
         SVC   0.719149 0.728448 0.922847 0.723769   0m 0.1763s
         KN   0.000000 0.000000 0.861244 0.000000   0m 0.0010s
         NB   0.871094 0.961207 0.974880 0.913934   0m 0.0022s
         DT   0.873874 0.836207 0.960526 0.854626   0m 0.1386s
         LR   0.971292 0.875000 0.979067 0.920635   0m 0.0104s
         RF   1.000000 0.818966 0.974880 0.900474   0m 1.3429s
         AdaBoost   0.937799 0.844828 0.970694 0.888889   0m 2.4455s
         BgC   0.956731 0.857759 0.974880 0.904545   0m 0.8343s
         ETC   1.000000 0.810345 0.973684 0.895238   0m 0.7918s

In [25]: df.plot(kind='bar', y="Accuracy", ylim=(0.8,1.0), figsize=(11,6), align='center', col
         plt.xticks(np.arange(9), df.index)
         plt.ylabel('Accuracy Score')
         plt.title('Distribution by Classifier')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.savefig("../img/sms-acc-basemodel-v4-stop.eps")
         plt.show()
```

Looks like ensemble classifiers are not doing as good as expected.

### 0.0.4 Voting classifier

We are using ensemble algorithms here, but what about ensemble of ensembles? Will it beat NB?

```
In [26]: from sklearn.ensemble import VotingClassifier
```

```
In [27]: eclf = VotingClassifier(estimators=[('BgC', bc), ('ETC', etc), ('RF', rfc), ('Ada', a
```

```
In [28]: eclf.fit(features_train,labels_train)
```

```
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/preprocessing/label.py:95: Da
  y = column_or_1d(y, warn=True)
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/preprocessing/label.py:128: Da
  y = column_or_1d(y, warn=True)
```

```
Out[28]: VotingClassifier(estimators=[('BgC', BaggingClassifier(base_estimator=None, bootstrap=
                    bootstrap_features=False, max_features=1.0, max_samples=1.0,
                    n_estimators=9, n_jobs=1, oob_score=False, random_state=111,
                    verbose=0, warm_start=False)), ('ETC', ExtraTreesClassifier(bootstrap=False,
                     learning_rate=1.0, n_estimators=62, random_state=111))],
                    flatten_transform=None, n_jobs=1, voting='soft', weights=None)
```

```
In [29]: pred = eclf.predict(features_test)
```

```
/Users/alex/anaconda/envs/gc/lib/python3.6/site-packages/sklearn/preprocessing/label.py:151: De
  if diff:
```

```
In [30]: print(precision_score(labels_test,pred), recall_score(labels_test,pred), accuracy_sco
```

```
0.9848484848484849 0.8405172413793104 0.9760765550239234 0.9069767441860466
```

Better but nope.

0.0.5 RNN

Define the RNN structure.

```
In [31]: 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.
```

0.0.6 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 < 200)

```
In [32]: features_lstm = text_feat
         labels_lstm = labels
```

```
In [33]: max_words = 1000
         max_len = 200 # 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 [34]: features_lstm.shape
```

```
Out[34]: (5572, 200)
```

```
In [35]: labels_lstm.shape
```

```
Out[35]: (5572, 1)
```

```
In [36]: features_lstm_train, features_lstm_test, labels_lstm_train, labels_lstm_test = train_
```

```
In [37]: def RNN():
             inputs = Input(name='inputs',shape=[max_len])
             layer = Embedding(max_words,50,input_length=max_len)(inputs)
             layer = LSTM(64)(layer)
             layer = Dense(256,name='FC1')(layer)
             layer = Activation('relu')(layer)
             layer = Dropout(0.5)(layer)
             layer = Dense(1,name='out_layer')(layer)
             layer = Activation('sigmoid')(layer)
             model = Model(inputs=inputs,outputs=layer)
             return model
```

Call the function and compile the model.

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

```
_____
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 200)               0
_____
```

7

```
embedding_1 (Embedding)        (None, 200, 50)          50000
------------------------------------------------------------------
lstm_1 (LSTM)                  (None, 64)               29440
------------------------------------------------------------------
FC1 (Dense)                    (None, 256)              16640
------------------------------------------------------------------
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: 96,337
Trainable params: 96,337
Non-trainable params: 0
------------------------------------------------------------------
```

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

        model.fit(features_lstm_train, labels_lstm_train, epochs=10, batch_size=128,validation
                        callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)])

        time_elapsed = time.time() - since

```
Train on 3120 samples, validate on 780 samples
Epoch 1/10
3120/3120 [==============================] - 7s 2ms/step - loss: 0.3950 - acc: 0.8413 - val_los
Epoch 2/10
3120/3120 [==============================] - 6s 2ms/step - loss: 0.1586 - acc: 0.9513 - val_los
Epoch 3/10
3120/3120 [==============================] - 7s 2ms/step - loss: 0.0698 - acc: 0.9814 - val_los
Epoch 4/10
3120/3120 [==============================] - 7s 2ms/step - loss: 0.0505 - acc: 0.9840 - val_los
Epoch 5/10
3120/3120 [==============================] - 6s 2ms/step - loss: 0.0421 - acc: 0.9881 - val_los
Epoch 6/10
3120/3120 [==============================] - 7s 2ms/step - loss: 0.0331 - acc: 0.9897 - val_los
Epoch 7/10
3120/3120 [==============================] - 6s 2ms/step - loss: 0.0235 - acc: 0.9926 - val_los
```

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

Training complete in 0m 45.1065s

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

In [42]: pred_scores.append(("LSTM", [precision_score(labels_lstm_test,pred), recall_score(lab
```

0.0.7 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"] = 10
             ca_config["early_stopping_rounds"] = 3
             ca_config["n_classes"] = 2
             ca_config["estimators"] = []
             ca_config["estimators"].append({"n_folds": 5, "type": "RandomForestClassifier", "n
             ca_config["estimators"].append({"n_folds": 5, "type": "MultinomialNB", "alpha": 0

             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_train = features_train.toarray()
         labels_train = labels_train.reshape(-1)

         since = time.time()
         gc.fit_transform(features_train, labels_train)

         time_elapsed = time.time() - since

         # gc.fit_transform(features_train, labels_train, features_test, labels_test)
```
```
[ 2019-04-23 22:15:17,887][cascade_classifier.fit_transform] X_groups_train.shape=[(3900, 9403)
[ 2019-04-23 22:15:18,227][cascade_classifier.fit_transform] group_dims=[9403]
[ 2019-04-23 22:15:18,228][cascade_classifier.fit_transform] group_starts=[0]
[ 2019-04-23 22:15:18,229][cascade_classifier.fit_transform] group_ends=[9403]
[ 2019-04-23 22:15:18,230][cascade_classifier.fit_transform] X_train.shape=(3900, 9403),X_test
[ 2019-04-23 22:15:18,419][cascade_classifier.fit_transform] [layer=0] look_indexs=[0], X_cur_t
[ 2019-04-23 22:15:23,714][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 22:15:27,805][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 22:15:31,758][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 22:15:35,497][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
```

```
[ 2019-04-23 22:15:39,151][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 22:15:39,152][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_0 - 5_
[ 2019-04-23 22:15:39,389][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:39,636][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:39,864][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:40,100][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:40,331][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:40,333][kfold_wrapper.log_eval_metrics] Accuracy(layer_0 - estimator_1 - 5_
[ 2019-04-23 22:15:40,334][cascade_classifier.calc_accuracy] Accuracy(layer_0 - train.classifie
[ 2019-04-23 22:15:40,556][cascade_classifier.fit_transform] [layer=1] look_indexs=[0], X_cur_
[ 2019-04-23 22:15:43,144][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:45,762][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:48,183][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:50,782][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:53,413][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:53,414][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_0 - 5_
[ 2019-04-23 22:15:53,681][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:53,931][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:54,168][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:54,422][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:54,660][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:54,661][kfold_wrapper.log_eval_metrics] Accuracy(layer_1 - estimator_1 - 5_
[ 2019-04-23 22:15:54,662][cascade_classifier.calc_accuracy] Accuracy(layer_1 - train.classifie
[ 2019-04-23 22:15:54,869][cascade_classifier.fit_transform] [layer=2] look_indexs=[0], X_cur_
[ 2019-04-23 22:15:57,297][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:15:59,833][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:16:02,350][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:16:04,782][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:16:07,315][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:16:07,317][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_0 - 5_
[ 2019-04-23 22:16:07,573][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:07,815][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:08,060][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:08,308][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:08,547][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:08,548][kfold_wrapper.log_eval_metrics] Accuracy(layer_2 - estimator_1 - 5_
[ 2019-04-23 22:16:08,549][cascade_classifier.calc_accuracy] Accuracy(layer_2 - train.classifie
[ 2019-04-23 22:16:08,768][cascade_classifier.fit_transform] [layer=3] look_indexs=[0], X_cur_
[ 2019-04-23 22:16:12,105][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:14,970][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:17,944][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:20,360][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:23,080][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:23,083][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_0 - 5_
[ 2019-04-23 22:16:23,327][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 22:16:23,576][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 22:16:23,827][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 22:16:24,075][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
```

```
[ 2019-04-23 22:16:24,315][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 22:16:24,316][kfold_wrapper.log_eval_metrics] Accuracy(layer_3 - estimator_1 - 5_
[ 2019-04-23 22:16:24,317][cascade_classifier.calc_accuracy] Accuracy(layer_3 - train.classifi
[ 2019-04-23 22:16:24,319][cascade_classifier.fit_transform] [Result][Optimal Level Detected]
```

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

Training complete in 1m 6.5370s
```

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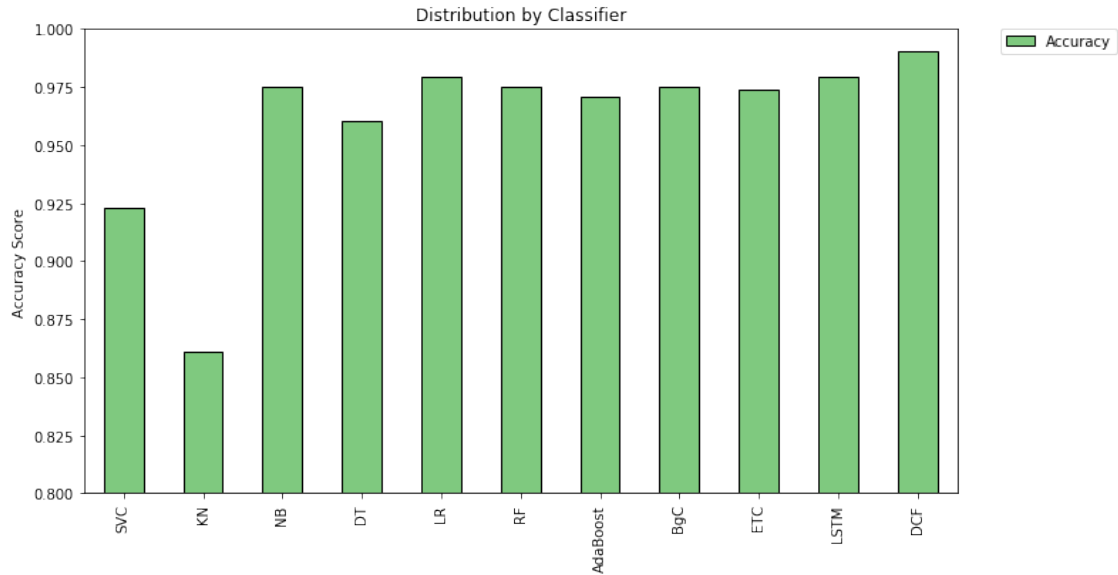         pred_scores.append(("DCF", [precision_score(labels_test,pred), recall_score(labels_te
```

```
[ 2019-04-23 22:16:24,472][cascade_classifier.transform] X_groups_test.shape=[(1672, 9403)]
[ 2019-04-23 22:16:24,595][cascade_classifier.transform] group_dims=[9403]
[ 2019-04-23 22:16:24,596][cascade_classifier.transform] X_test.shape=(1672, 9403)
[ 2019-04-23 22:16:24,681][cascade_classifier.transform] [layer=0] look_indexs=[0], X_cur_test
```

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

```
Out[48]:          Precision    Recall   Accuracy         F1 Training Time (s)
         SVC       0.719149  0.728448  0.922847   0.723769         0m 0.1763s
         KN        0.000000  0.000000  0.861244   0.000000         0m 0.0010s
         NB        0.871094  0.961207  0.974880   0.913934         0m 0.0022s
         DT        0.873874  0.836207  0.960526   0.854626         0m 0.1386s
         LR        0.971292  0.875000  0.979067   0.920635         0m 0.0104s
         RF        1.000000  0.818966  0.974880   0.900474         0m 1.3429s
         AdaBoost  0.937799  0.844828  0.970694   0.888889         0m 2.4455s
         BgC       0.956731  0.857759  0.974880   0.904545         0m 0.8343s
         ETC       1.000000  0.810345  0.973684   0.895238         0m 0.7918s
         LSTM      0.912664  0.933036  0.979067   0.922737        0m 45.1065s
         DCF       0.977876  0.952586  0.990431   0.965066         1m 6.5370s
```

```
In [50]: df.plot(kind='bar', y="Accuracy", ylim=(0.8,1.0), figsize=(11,6), align='center', col
         plt.xticks(np.arange(11), df.index)
         plt.ylabel('Accuracy Score')
         plt.title('Distribution by Classifier')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
         plt.savefig("../img/sms-acc-v4-stop.eps")
         plt.show()
```

Distribution by Classifier

```
In [51]: import pickle
         # dump
         with open("../pkl/sms-gc-v4-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)
```

0.0.8 Final verdict - gcForest is your friend in spam detection.

```
In [ ]:
```