Capstone Project 2: **Milestone Report 2**

Machine Learning For Detecting Spam Email

Problem Statement

Our digital world, everyone communicates through emails to reach personal and business contacts. One main problem is to manage time spent on reading and responding to emails that are of importance. In this project, various supervised models are used to detecting spam emails and evaluate how the selected models may contribute to this global issue.

Exploratory Data Analysis & Data Wrangling

The clean release of Enron email dataset published by EDRM is utilized. The emails are in text format and organized in various subdirectories. To avoid unbalanced data, email subdirectories were selected to provide enough ham (5709) and spam (5241) emails to compose a balanced dataset with total of about 10,949 emails. To use this dataset, each subset of emails was imported in Jupyter notebook and labeled accordingly. Finally after joining the two set of emails, the order of observations was shuffled to create randomness in the order. To prepare an unseen test set, this process was repeated for a second set of subdirectories for additional evaluation.

	Text	Label	Ham1/Spam0	Line Count	Text Length
2628	Subject: cruise 3 nts mexico only $ 197 ! - - ...	spam	0	23	1427
1005	Subject: ba & sao paulo\r\n- - - - - - - - - - -...	ham	1	10	452
3675	Subject: re : status\r\nclayton ,\r\nwe can di...	ham	1	87	5656
2219	Subject: enlarge your penls\r\nenlarge your pe...	spam	0	3	60
1377	Subject: urgent business\r\n> > from the desk ...	spam	0	26	2538

These datasets were saved as csv files. The first dataset is saved as ‘EmailList.csv’ and the unseen testset is saved as ‘UnseenEmailList.csv’.

Machine Learning Models

Considering the dataset has labeled data as 'ham' and 'spam', it is appropriate to use supervised machine learning. Since the content to be processed is text, integrating NLP preprocessing techniques is required.

After splitting train and test sets, to simplify the process, pipelines were defined to preprocess train data with TFIDF and then apply each of the selected models. Then each model was evaluated calculated F1-score for test set and the secondary unseen test set.
Furthermore, TFIDF hyperparameters were selected and iterated for training, fitting and evaluating to observe the model performance corresponding to each hyperparameter.

Models:

- Logistic Regression
- Naive Bayes Classifier
- Support Vector Machine
- Linear Support Vector Machine
- SGD

Steps of analysis:

- **Select relevant ML model for classification of labeled data**
- **Create pipeline for each model**
- **List TFIDF and ML model in pipeline**
- **Evaluate model performance F1- score**
- **Select Hyperparameters and values of interest**
- **Review and evaluate model performance F1- score to finalize parameter settings of interest**
- **Create and display table containing model applied, performance score, and corresponding hyperparameters**
- **Plot model performance for visualization**

```
# Change vocabulary size by reducing the min/max number of document term
frequency
TFIDF_ngram = [1, 2]
TFIDF_min = [0.01, 0.05, 0.1]
TFIDF_max = [0.11, 0.5, 0.8]
TFIDF_max_feat = [100, None]
model_randomState = 40
```

Logistic Regression

In [8]:

```
pkList = pd.DataFrame()
paramList = pd.DataFrame()
pList=[]
pLst = []
# All Logistic Regression solvers produced same score/result
for n in TFIDF_ngram:
    for tfidf_min in TFIDF_min:
        for tfidf_max in TFIDF_max:
            for tfidf_max_feat in TFIDF_max_feat:
                # TfidfVectorizer
                pipelineLR = Pipeline([('tfidf_vect',
                  TfidfVectorizer(stop_words=None, ngram_range = [1,n], \
                                 min_df = tfidf_min, max_df = tfidf_max, \
                                 max_features = tfidf_max_feat)), \
                                      ('to_dens', ToDenseTransformer()), \
                                      ('pca', PCA()), \
                                      ('log_reg', \
                                        LogisticRegression(solver = 'lbfgs',\
                                  random_state = model_randomState))])
                pipelineLR = pipelineLR.fit(X_train, y_train)
                y_pred_train_LR = pipelineLR.predict(X_train)
                y_pred_test_LR = pipelineLR.predict(X_test)
                # Predict independent emails, unseen_X_test
                unseen_y_pred_test_LR = pipelineLR.predict(unseen_X_test)
                # Accuracy score prediction performance
                train_score = np.round(metrics.accuracy_score(y_train,\
                                                          y_pred_train_LR),4)
                # Test predict F1 Score for comparison
                mean_f1 = np.round(metrics.f1_score(y_test, y_pred_test_LR,\
                                                      average='micro'),4)
                unseen_mean_f1 = np.round(metrics.f1_score(unseen_y_test,\
                                  unseen_y_pred_test_LR, average='micro'),4)
                pLst.append(['Logistic Regression', train_score, mean_f1,\
                                              unseen_mean_f1])
                pList.append(pipelineLR.get_params(deep = True).values())

pKeys = pipelineLR.get_params(deep = True).keys()
# append list to DataFrame
LR_score_cols = ['Model', 'Train Score', 'F1-Score', 'unseen_F1-Score']
pkList = pd.DataFrame(pLst, columns = LR_score_cols)
paramList = pd.DataFrame(pList, columns = pKeys)
LR_performance = pkList.join(paramList)
```

Data Storytelling

Comparing model score will be used to present the supervised model with best prediction performance.

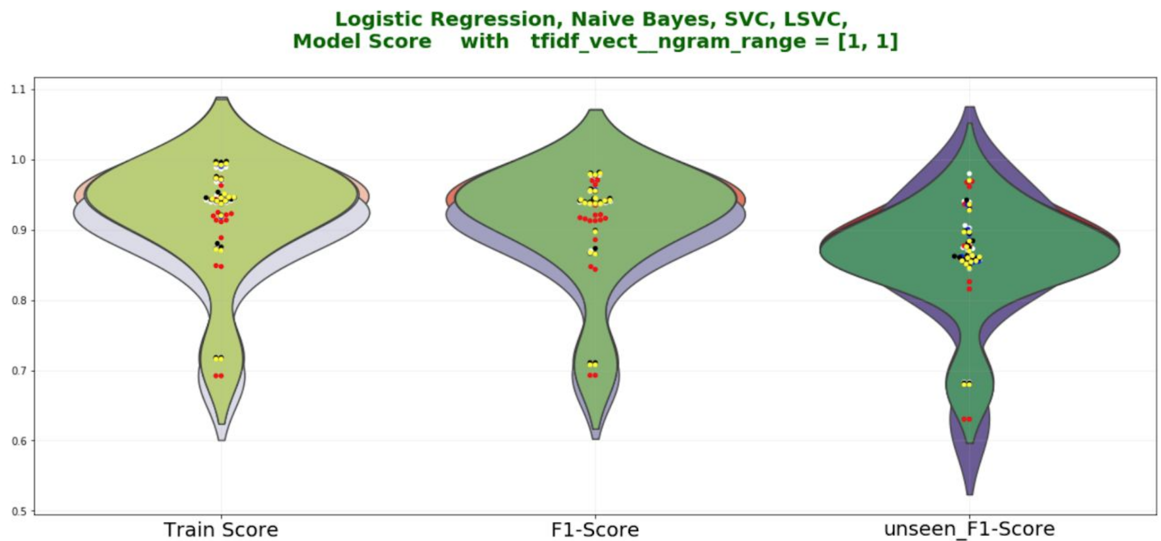
Model Performance Table

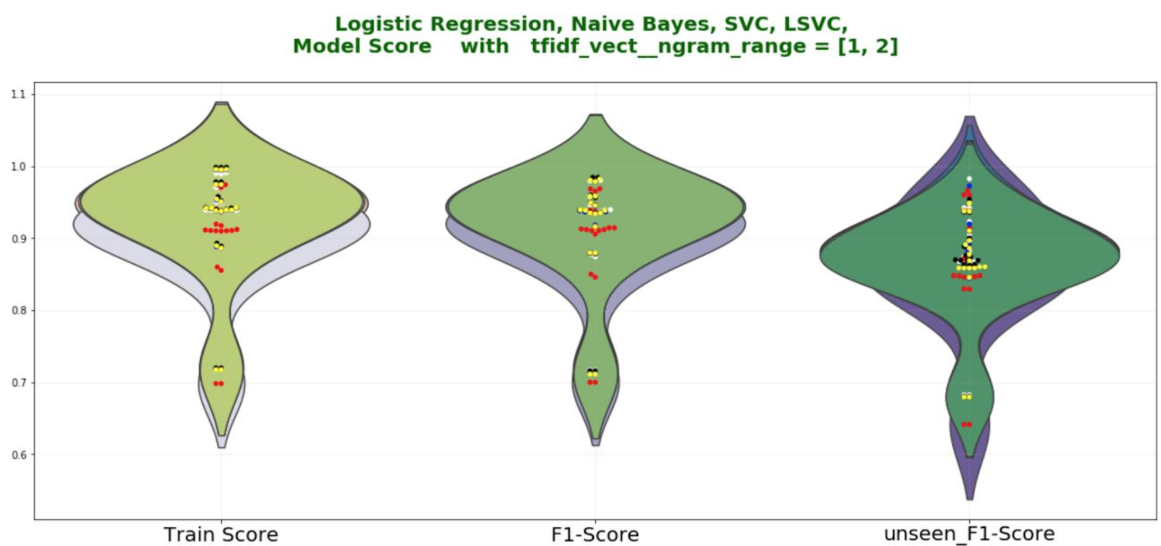
	Model	Train Score	F1-Score	unseen_F1-Score	tfidf_vect__max_df	tfidf_vect__max_features	tfidf_vect__min_df	tfidf_vect__ngram_range
19	Logistic Regression	0.9907	0.9801	0.9824	0.11	NaN	0.01	[1, 2]
1	Logistic Regression	0.9863	0.9744	0.9797	0.11	NaN	0.01	[1, 1]
19	SVC	0.9946	0.9782	0.9724	0.11	NaN	0.01	[1, 2]
1	SGD	0.992	0.9778	0.9698	0.11	NaN	0.01	[1, 1]
1	SVC	0.9899	0.9774	0.969	0.11	NaN	0.01	[1, 1]
3	Naive Bayes	0.9731	0.9698	0.9686	0.5	NaN	0.01	[1, 1]
5	Naive Bayes	0.9733	0.9705	0.9675	0.8	NaN	0.01	[1, 1]
19	Naive Bayes	0.9701	0.9652	0.9659	0.11	NaN	0.01	[1, 2]
1	Naive Bayes	0.963	0.964	0.9613	0.11	NaN	0.01	[1, 1]

Finally, for n-gram variation the scores are distributed between 65% and upper 90%. Although there are slight distinguishing differences in model scores, both n_gram = [1,1] and [1,2] lead to similar performance.

Notable observation here is that generalized models, such as the Logistic Regression, perform better when including all features and/or extending n-grams to include 2 or 3 token words. Models that are not generalized, such as Naive Bayes, tend to overfit the training set and usually do not perform as optimized.

	Violine Plot Palette	Swarmplot color
Logistic Regression	Reds	White
Naive Bayes	Purples	Red
SVC (Support Vector Classifier)	Blues	Black
Linear SVC	Grays	Blue
SGD	Greens	Yellow





Dataset Source:

https://www.edrm.net/resources/data-sets/edrm-enron-email-data-set/