Cairo university
Faculty of engineering
Computer engineering department
Machine Learning [**CMP4040**]
Project Report

Web page Phishing Detection

Team 9

Name	section	BN
احمد أسعد درويش محمد درويش	1	1
عمر فريد عبد العاطى لملوم	2	4
محمد نبيل عبد الفتاح فهمى	2	19
ممدوح احمد محمد محمد عطيه	2	27

Presented to:

Eng. Mohamed Shawky

Workload Division

Name	Workload
احمد أسعد درويش محمد درويش	Data Preprocessing
عمر فريد عبد العاطى لملوم	Data Preprocessing
محمد نبيل عبد الفتاح فهمى	Models
ممدوح احمد محمد محمد عطيه	Models

Problem definition & Motivation

Phishing continue s to prove one of the most successful and effective ways for cybercriminals *to defraud us and steal our personal and financial information*.

Our growing reliance on the internet to conduct much of our day-to-day business has provided fraudsters with the perfect environment to launch targeted phishing attacks. The phishing attacks taking place today are sophisticated and increasingly more difficult to spot. A study conducted by Intel found that 97% of security experts fail at identifying phishing emails from genuine emails.

So in our ML project we would like to address this problem by training 3 phishing detection models and apply our knowledge to evaluate these using the following metrics for example:

Evaluation metrics

Here are some of our proposed metrics (subject to add more of them – will be clarified in the final report إن شاء الله)

1. Accuracy
2. Confusion Matrix
    - which in turn include:
        i. TP : True positives
        ii. TN : True Negatives
        iii. FP : False positives
        iv. FN : False negatives
3. F1 – Score
4. Precision
5. Recall

Dataset Link

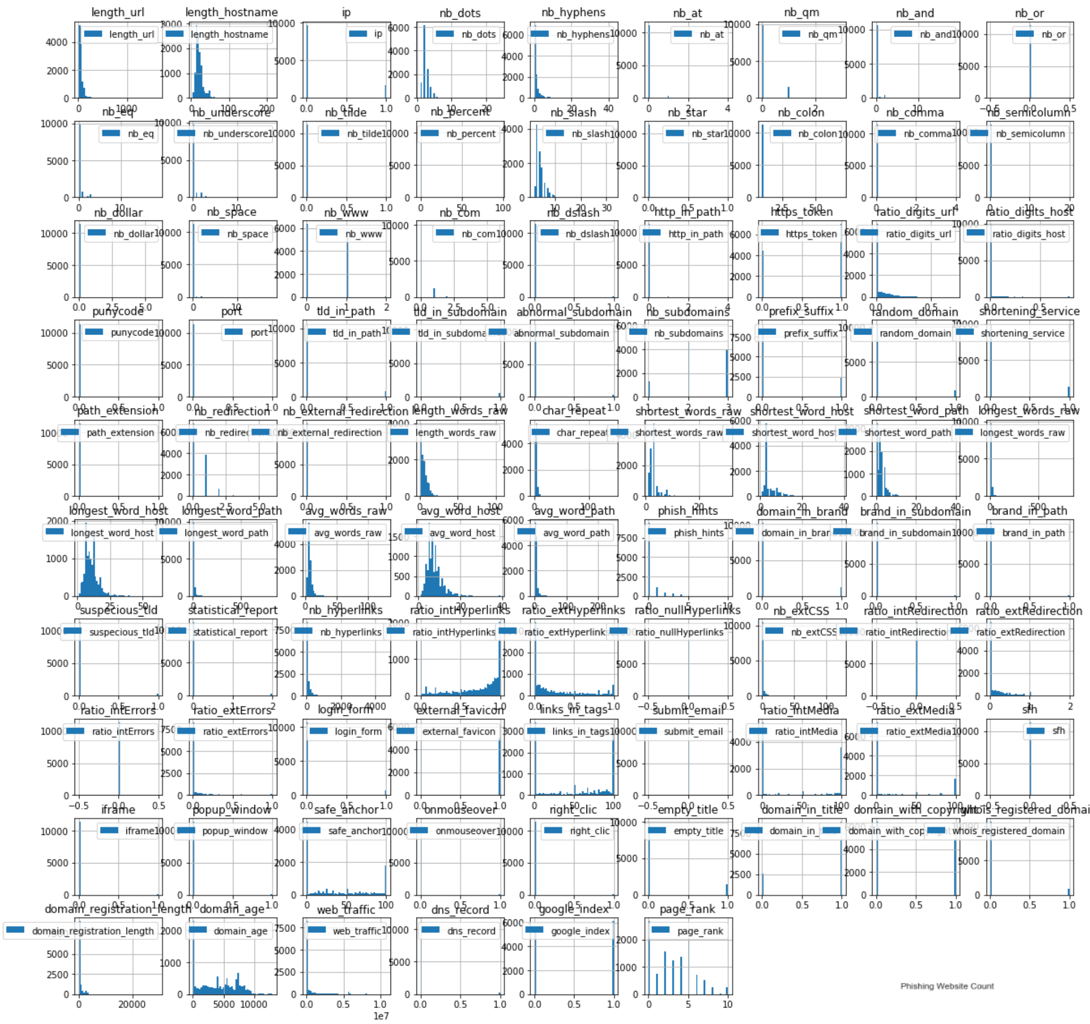
The dataset that we propose to use:

https://www.kaggle.com/datasets/shashwatwork/web-page-phishing-detection-dataset?resource=download

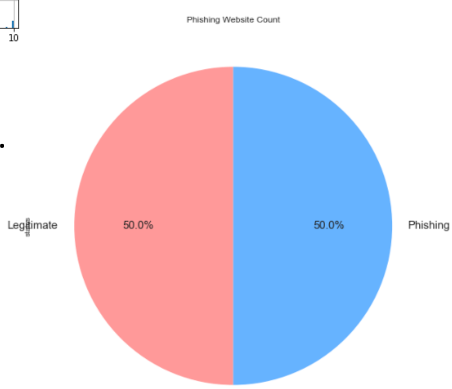
#1 : Dataset analysis

Let's talk about dataset analysis in the upcoming bullet-points

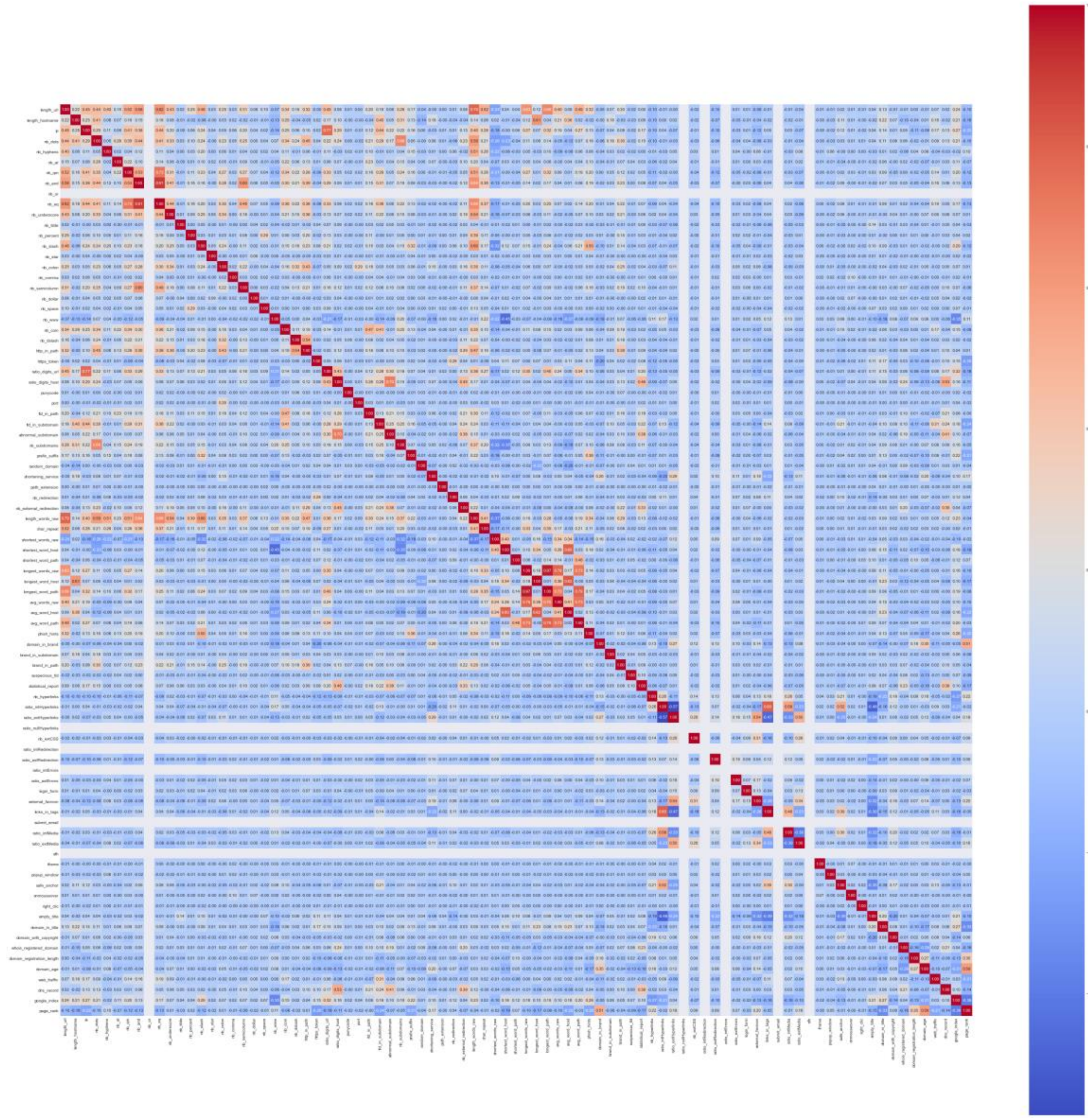
1. At first , we loaded the dataset from Kaggle site.
2. Explore the dataset : **info** – **description** – **shape.**
3. Data preprocessing : Drop duplicates – Drop nulls [There weren't any of these in our dataset]
4. Dataset visualization :
   a. **Histogram of features** : They gave me some insights about the feature values ranges and frequencies. Also you can notice that Many features are regex features ➔ The majority of values are zero , and they take that values 0 or 1. At first I thought about dropping them , but said that they may turn to have useful information even if small.



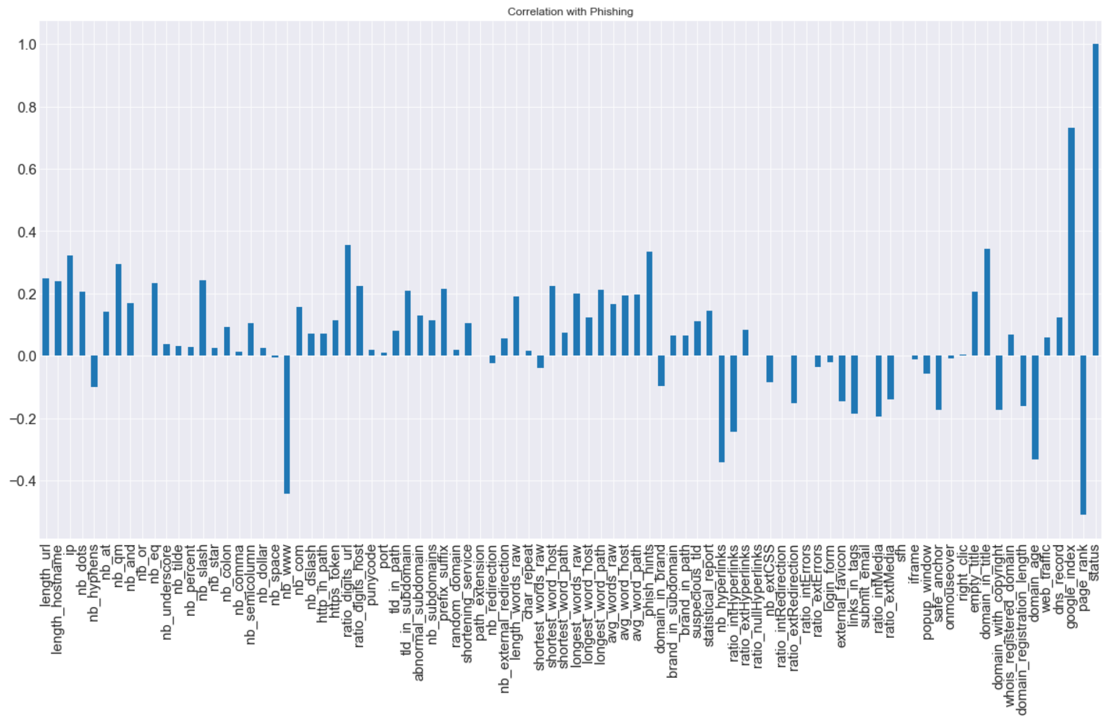
   b. Pie chart of the output variable
      i. Concluded that **the dataset is balanced.**



c. Correlation matrix

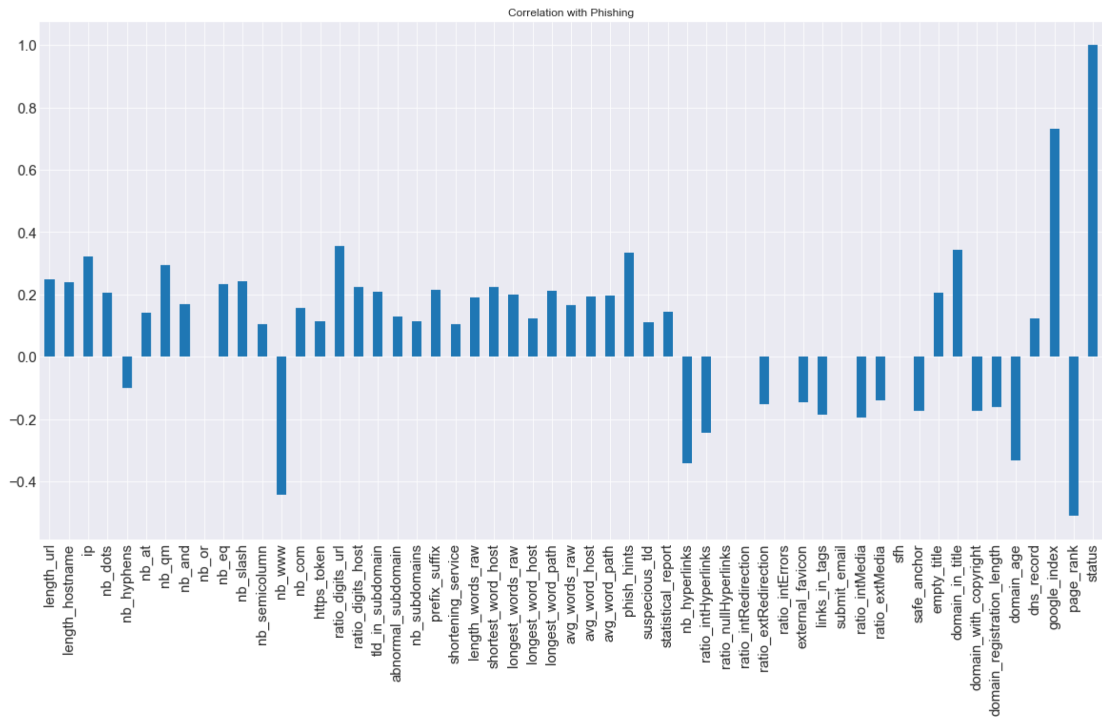


d. Correlation with the output variable yielded the following graph:



There were 2 experiments made , we will show the results before and after dropping the lowly correlated features (with target correlation < 0.1) in the experiments section below.

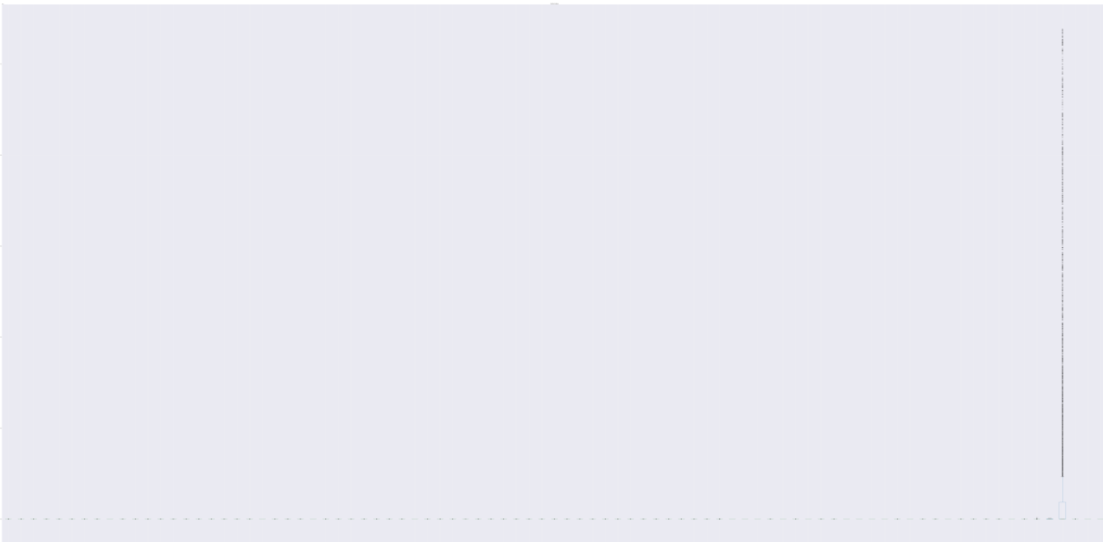
Anyway , after dropping the columns with correlation in range [-1 : 1], here are the rest of the features after dropping these columns:



Top 5 features with the highest correlation with the output variable

google_index	0.731171
page_rank	0.511137
nb_www	0.443468
ratio_digits_url	0.356395
domain_in_title	0.342807

e. Box Plot (To analyze outliers)



Woah ! umm well this is hard to view :)

```
Some important notes from the box plot are:
1. The feature : "web_traffic" has a lot of outliers.=> to solve this we can use log transformation.
2. Features ranges are different.=> to solve this we can use standardization.
Also , looks like the features needs scaling.   :)
```

f. Took random data sample to view (please refer to notebook for full row view:

	url	length_url	length_hostname	ip	nb_dots	nb_hyphens	nb_at	nb_qm	nb_and	nb_or	nb_eq	nb_underscore	nb_tilde	nb_percent	nb_slash	nb_star	nb_colon	nb_comma	nb_semicolumn
7655	http://lifehacker.com/what-is-tor-and-should-i...	64	14	1	1	8	0	0	0	0	0	0	0	0	3	0	1	0	0
5739	http://vksavesmusic.webservis.ru	32	25	0	2	0	0	0	0	0	0	0	0	0	2	0	1	0	0
5568	https://www.mk2.com/	20	11	0	2	0	0	0	0	0	0	0	0	0	3	0	1	0	0
3568	https://mic.com/articles/171764/nintendo-switc...	109	7	0	1	11	0	0	0	0	0	0	0	0	5	0	1	0	0
3963	https://www.breeze.pm/	22	13	0	2	0	0	0	0	0	0	0	0	0	3	0	1	0	0

5. Data preprocessing:

a. We have to convert the categorical data into numerical data

```
## the only categorical data are the target column and the url column
# we will convert the target column to numerical data
#by mapping the values : 1 for phishing and 0 for legitimate
```

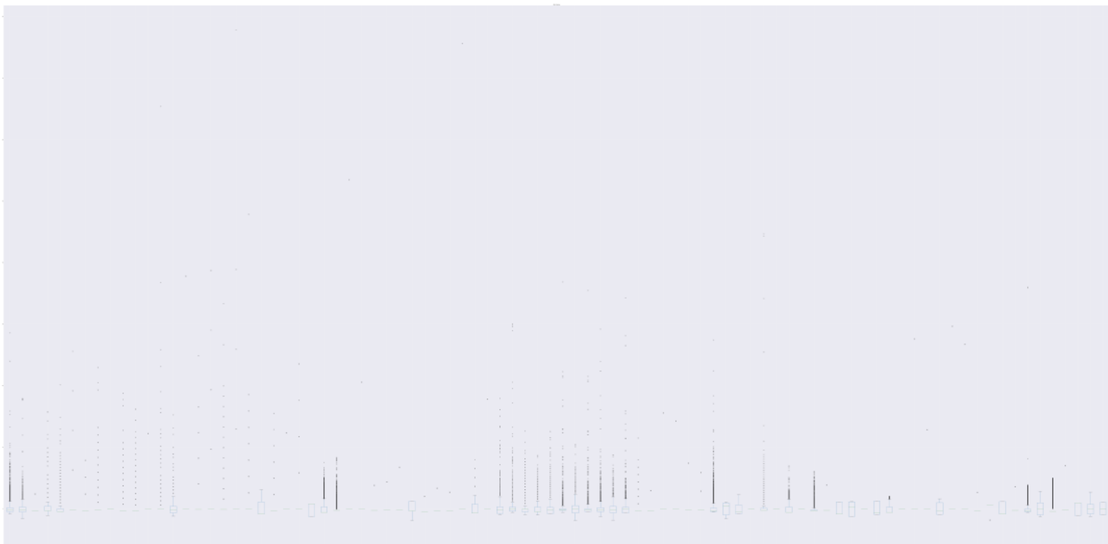
b. The url column is not useful for the model so we will drop it

c. Scaling the features using a StandardScaler.

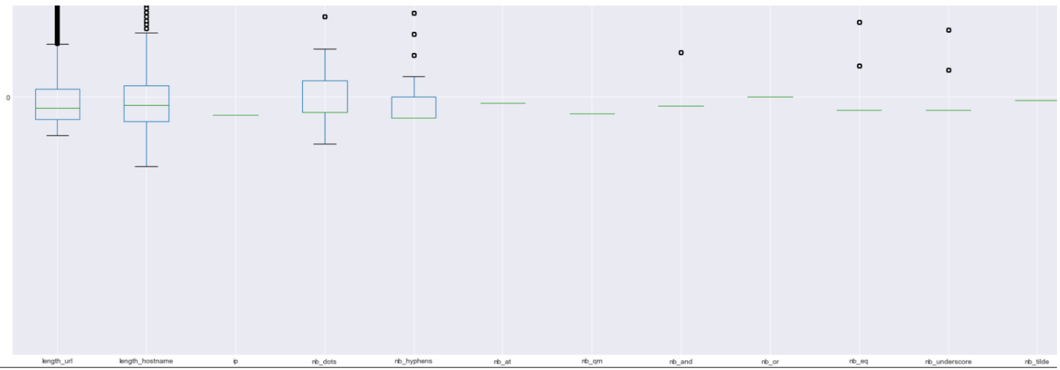
d. Fix the web_traffic column values
    i. we will use the median value to replace the negative values

now it looks like this:

e. Redrawing BoxPlot after the scaling and fixing



A bit better and the boxes are more apparent . For Better visualization kindly run the corresponding cell and open the plot from the cell and zoom in like this:



```
At first it came to my mind to remove the remaining outliers.
But after searching I decided to keep them because they are
important for the model to learn the patterns, and gain insights
from the data.
```

#2 Experiments & Results

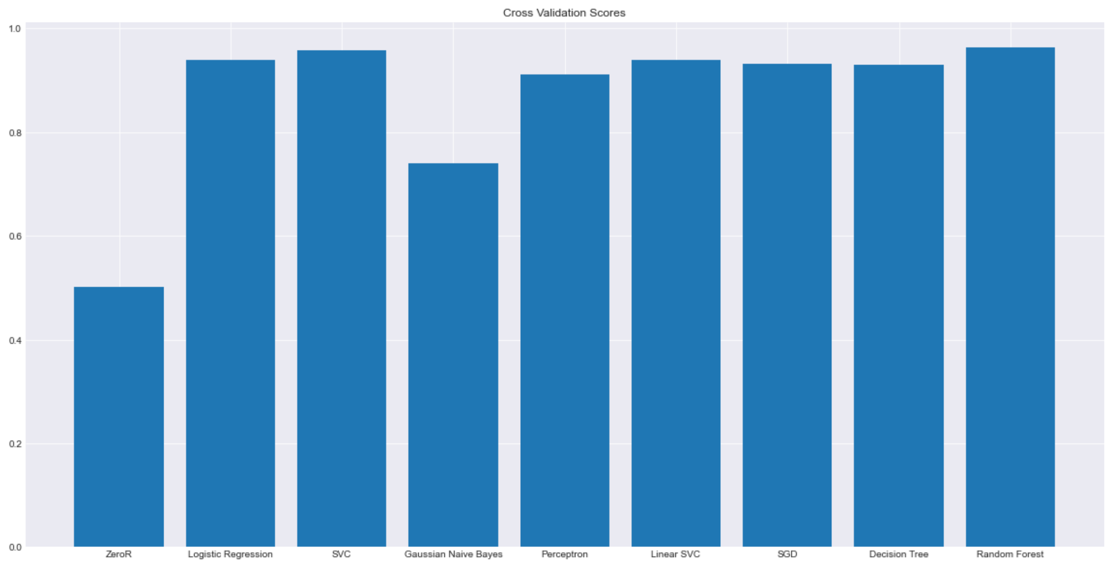
Experiment #1 : Without dropping low correlated features:

Experiment #2 : After dropping low correlated features (no hyperparameter tuning experiment):

Models accuracies:

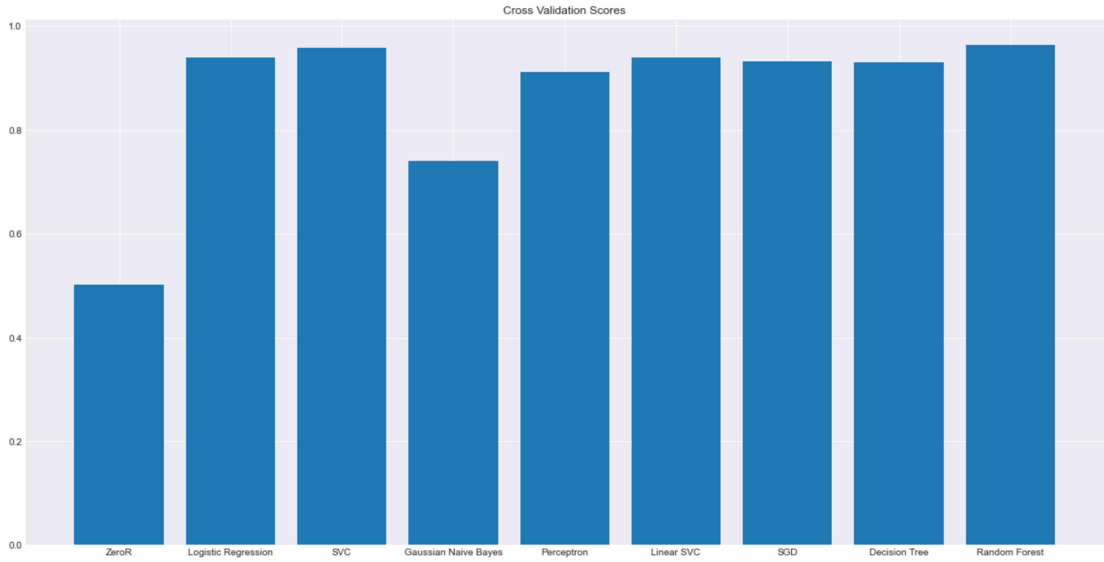
	Model	Score
8	Random Forest	0.968066
2	SVC	0.963255
5	Linear SVC	0.956693
1	Logistic Regression	0.955818
6	SGD	0.940507
7	Decision Tree	0.939633
4	Perceptron	0.910324
3	Gaussian Naive Bayes	0.680665
0	ZeroR	0.493876

Cross validation scores:



	Cross Validation Score
Random Forest	0.964350
SVC	0.958225
Logistic Regression	0.940182
Linear SVC	0.939635
SGD	0.932307
Decision Tree	0.930884
Perceptron	0.910873
Gaussian Naive Bayes	0.739718
ZeroR	0.501531

Cross validation scores [no hyperparameters tuning]:



Models

===

1. Ensemble Learning

a. Bagging

Using the following estimators:

estimators=[('zeroR',zero_r),('logreg', logreg), ('svc', svc), ('gaussian', gaussian), ('perceptron', perceptron), ('linear_svc', linear_svc), ('sgd', sgd), ('decision_tree', decision_tree), ('random_forest', random_forest)]

The accuracy is: **0.9597550306211724**

```
Another Experiment on the best 5 classsifier in the voting classifier
```

Using the following estimators:

estimators=[('logreg', logreg), ('svc', svc), ('linear_svc', linear_svc), ('sgd', sgd), ('random_forest', random_forest)]

The accuracy is: **0.9545056867891514**

So random forest accuracy is better than ensemble learning Boosting

Which is logical :) they are't weak learners , not

a. Boosting

Using AdaboostClassifier and RandomForest estimator:

adaboost = AdaBoostClassifier(RandomForestClassifier(), n_estimators=5)

The accuracy is: **0.9676290463692039**

—--------

2. ZeroR: as a baseline

zero_r = DummyClassifier(strategy='most_frequent', random_state=12)

F1 Score: 0.3331388564760793

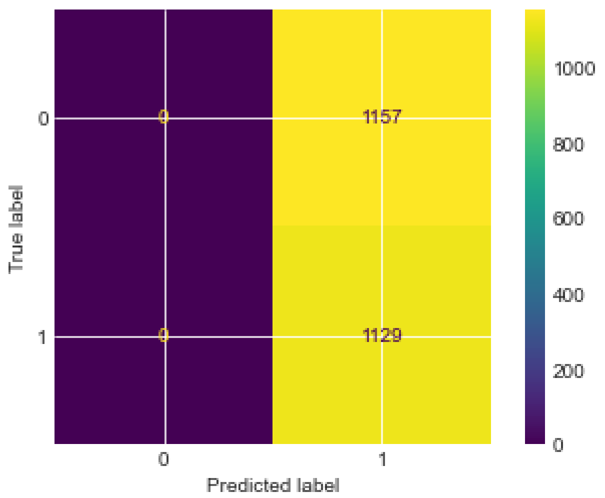
Confusion Matrix:

[[0 1157]

[0 1129]]

Classification Report:

	precision	recall	f1-score	support
-1.0	0.00	0.00	0.00	1157
1.0	0.49	1.00	0.66	1129
accuracy			0.49	2286
macro avg	0.25	0.50	0.33	2286
weighted avg	0.24	0.49	0.33	2286



logistic regression:

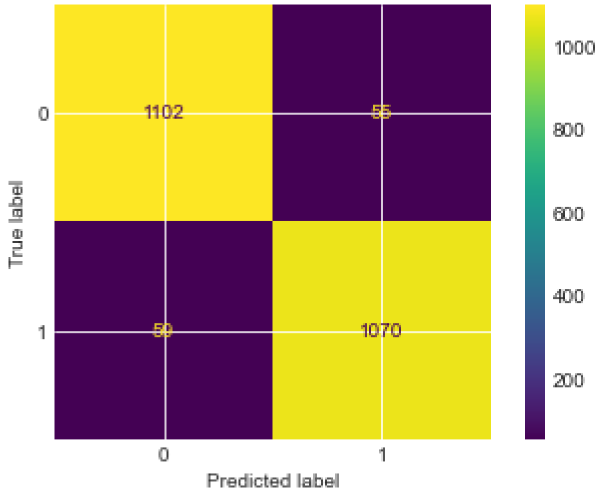
logreg = LogisticRegression()

Logistic Regression Accuracy: 0.9501312335958005

[[1102   55]

 [59 1070]]

	precision	recall	f1-score	support
-1.0	0.95	0.95	0.95	1157
1.0	0.95	0.95	0.95	1129
accuracy			0.95	2286
macro avg	0.95	0.95	0.95	2286
weighted avg	0.95	0.95	0.95	2286



Support Vector Machines+ Hyperparameter tuning:

svc = SVC()

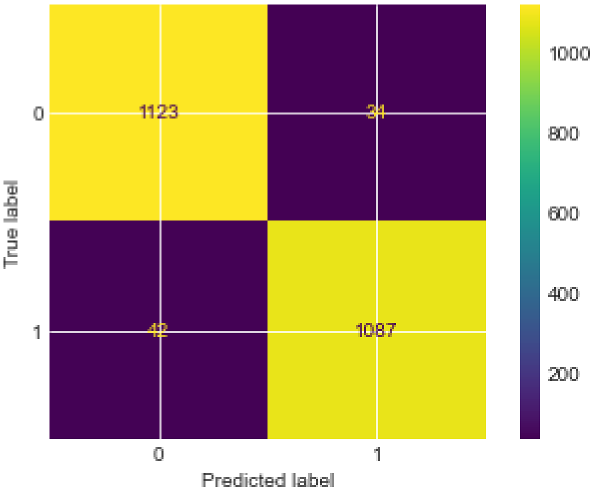
```
#hyperparameters for SVM are:
# C: regularization parameter
# kernel: specifies the kernel type to be used in the algorithm
# linear: linear kernel
# poly: polynomial kernel
# rbf: radial basis function kernel
# sigmoid: sigmoid kernel
# degree: degree of the polynomial kernel function
# gamma: kernel coefficient for rbf, poly and sigmoid
# random_state: seed for random number generator

C = [0.1, 1, 10, 100]
kernel = ['linear', 'poly', 'rbf', 'sigmoid']
degree = [3, 4, 5]
gamma = ['scale', 'auto']
```

SVM Accuracy: 0.9667541557305337

[[1123   34]

 [42 1087]]

	precision	recall	f1-score	support
-1.0	0.96	0.97	0.97	1157
1.0	0.97	0.96	0.97	1129
accuracy			0.97	2286
macro avg	0.97	0.97	0.97	2286
weighted avg	0.97	0.97	0.97	2286


1123
34
42
1087
0
1
0
1
True label
Predicted label
1000
800
600
400
200

<u>Gaussian Naive Bayes:</u>

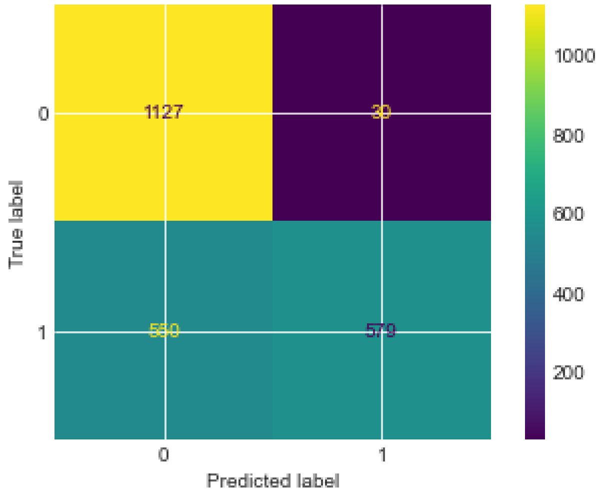
gaussian = GaussianNB()

Gaussian Naive Bayes Accuracy: 0.7462817147856518

[[1127   30]

 [550 579]]

	precision	recall	f1-score	support
-1.0	0.67	0.97	0.80	1157
1.0	0.95	0.51	0.67	1129
accuracy			0.75	2286
macro avg	0.81	0.74	0.73	2286
weighted avg	0.81	0.75	0.73	2286



Perceptron + Hyperparameter tuning :

Hyperparameter tuning :

1- Perceptron:

list of hyperparameters

penalty : l1 or l2 : The penalty (aka regularization term) to be used

alpha : float : Constant that multiplies the regularization term. The higher the value, the stronger the regularization

max_iter : int : The maximum number of passes over the training data (aka epochs)

tol : float : The stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol)

early_stopping : bool : Whether to use early stopping to terminate training when validation score is not improving

validation_fraction : float : The proportion of training data to set aside as validation set for early stopping

n_iter_no_change : int : Number of iterations with no improvement to wait before stopping

shuffle : bool : Whether to shuffle training data before each iteration

Tested the following values:

```
penalty = ['l1', 'l2']
alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
max_iter = [100, 1000, 10000]
tol = [1e-3, 1e-4, 1e-5]
early_stopping = [True, False]
validation_fraction = [0.1, 0.2, 0.3]
n_iter_no_change = [5, 10, 15]
shuffle = [True, False]
```

Used the **RandomizedSearchCV**

```
perceptron_random = RandomizedSearchCV(estimator = perceptron,
param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
random_state=42, n_jobs = -1)
```

perceptron = Perceptron()

perceptron_random = RandomizedSearchCV(estimator = perceptron, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)

Fitting 3 folds for each of 100 candidates, totalling 300 fits

{'validation_fraction': 0.1, 'tol': 0.001, 'shuffle': True, 'penalty': 'l1', 'n_iter_no_change': 10, 'max_iter': 1000, 'early_stopping': False, 'alpha': 0.0001}

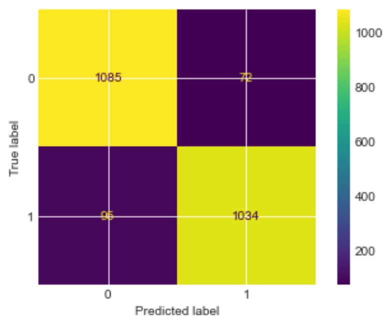
best params: {'validation_fraction': 0.1, 'tol': 0.001, 'shuffle': True, 'penalty': 'l1', 'n_iter_no_change': 10, 'max_iter': 1000, 'early_stopping': False, 'alpha': 0.0001}

Perceptron Accuracy: 0.9269466316710411

[[1085   72]

 [95 1034]]

	precision	recall	f1-score	support
-1.0	0.92	0.94	0.93	1157
1.0	0.93	0.92	0.93	1129
accuracy			0.93	2286
macro avg	0.93	0.93	0.93	2286
weighted avg	0.93	0.93	0.93	2286



Linear SVC:

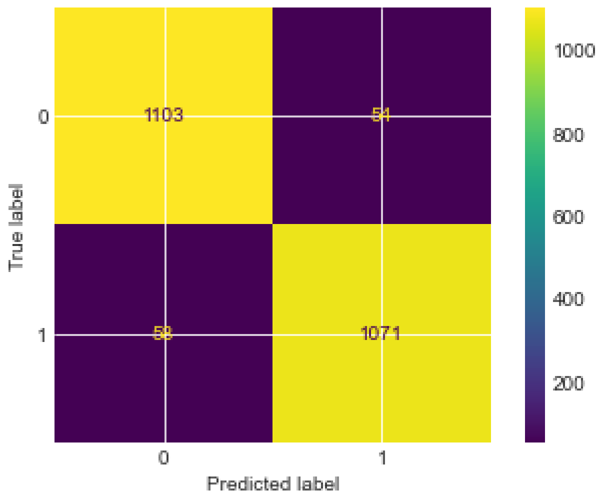
linear_svc = LinearSVC(max_iter=10000, dual=False)

Linear SVC Accuracy: 0.9510061242344707

[[1103   54]

 [58 1071]]

	precision	recall	f1-score	support
-1.0	0.95	0.95	0.95	1157
1.0	0.95	0.95	0.95	1129
accuracy			0.95	2286
macro avg	0.95	0.95	0.95	2286
weighted avg	0.95	0.95	0.95	2286



Stochastic Gradient Descent:

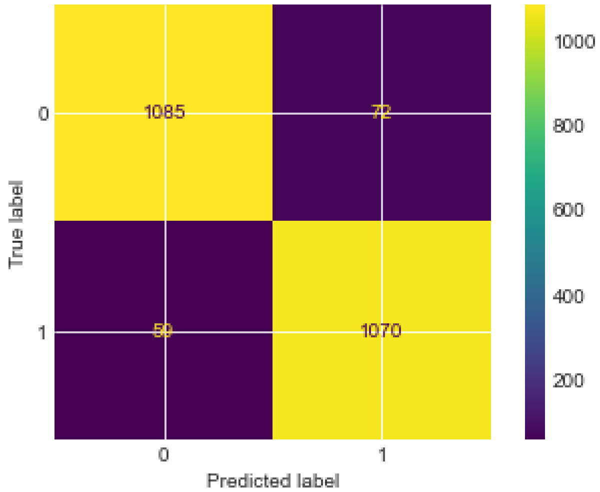
sgd = SGDClassifier()

SGD Accuracy: 0.9426946631671042

[[1085   72]

 [59 1070]]

	precision	recall	f1-score	support
-1.0	0.95	0.94	0.94	1157
1.0	0.94	0.95	0.94	1129
accuracy			0.94	2286
macro avg	0.94	0.94	0.94	2286
weighted avg	0.94	0.94	0.94	2286



Decision Tree:

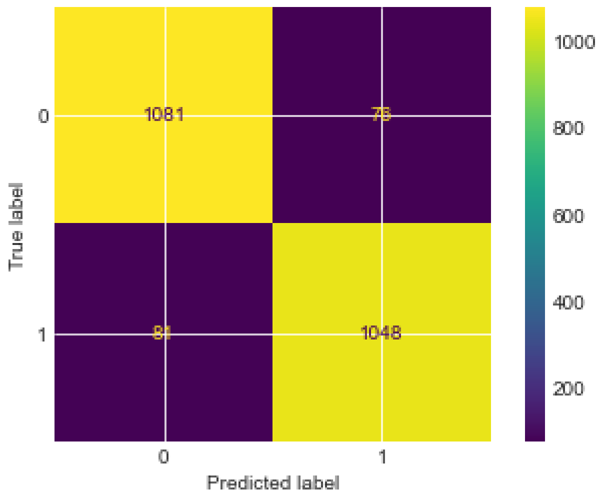
decision_tree = DecisionTreeClassifier()

Decision Tree Accuracy: 0.9313210848643919

[[1081   76]

 [81 1048]]

	precision	recall	f1-score	support
-1.0	0.93	0.93	0.93	1157
1.0	0.93	0.93	0.93	1129
accuracy			0.93	2286
macro avg	0.93	0.93	0.93	2286
weighted avg	0.93	0.93	0.93	2286



Random Forest:

random_forest = RandomForestClassifier(n_estimators=100)

Random Forest Accuracy: 0.968066491688539

[[1128   29]

 [44 1085]]

	precision	recall	f1-score	support
-1.0	0.96	0.97	0.97	1157
1.0	0.97	0.96	0.97	1129
accuracy			0.97	2286
macro avg	0.97	0.97	0.97	2286
weighted avg	0.97	0.97	0.97	2286