Dataset writing to csv file

In [130]:
```
# dictionary
# 1 = phishing
# -1 = non phishing
```

- Go to https://archive.ics.uci.edu/ml/machine-learning-databases/00327/Training%20Dataset.arff (https://archive.ics.uci.edu/ml/machine-learning-databases/00327/Training%20Dataset.arff)
- right click on the page and click 'Save As' and name something ending with .arff

In [104]:
```
# ^^ directions for commencement for creating a csv
#weka data set
```

In [266]:
```
# import pandas and arff for fast easy and exprssive data structure
import pandas as pd
import numpy as np
import arff
import urllib2
import matplotlib.pyplot as plt
import re
# importing library ^^ for plot production and interactive 2D data
visualizations.

from sklearn.ensemble import RandomForestClassifier
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (classification_report, confusion_matri
x, roc_curve, auc,
                             accuracy_score, roc_auc_score)
```

load the data with the path to the file and the give it a name

In [106]:
```
# data_arff = arff.load(open('Dataset.arff', 'rb'))
# ^^ direction to load the data, which is now stored in a dictionar
y
```

In [107]:
```
# column_names = [x[0] for x in data_arff['attributes']]
# ^^ directions to get the column names by calling the key 'attribu
tes' getting the first value in each tuple
```

In [108]:
```
# df = pd.DataFrame(data_arff['data'], columns = column_names)
# ^^ directions to load the data into a pandas data frame and set t
he column names
```

In [109]:
```
# df = df.astype(int)
# ^^ directions to change the column types from 'object' to 'int'
```

In [110]:
```
#df.Result = df.Result.map(lambda x: 0 if x <= -1 else 1)
```

csv data ready

In [114]:
```
#df.to_csv('phishingdata.csv')
```

In [ ]:
```
df = pd.read_csv('phishingdata.csv')
```

In [115]:
```
df.head()
```

Out[115]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	doubl
0	1	1	0	0	1
1	0	1	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	1	0	0

5 rows × 31 columns

In [125]:
```
df.age_of_domain.value_counts()
```

Out[125]:
```
-1    1088
 1    1080
 0     288
dtype: int64
```

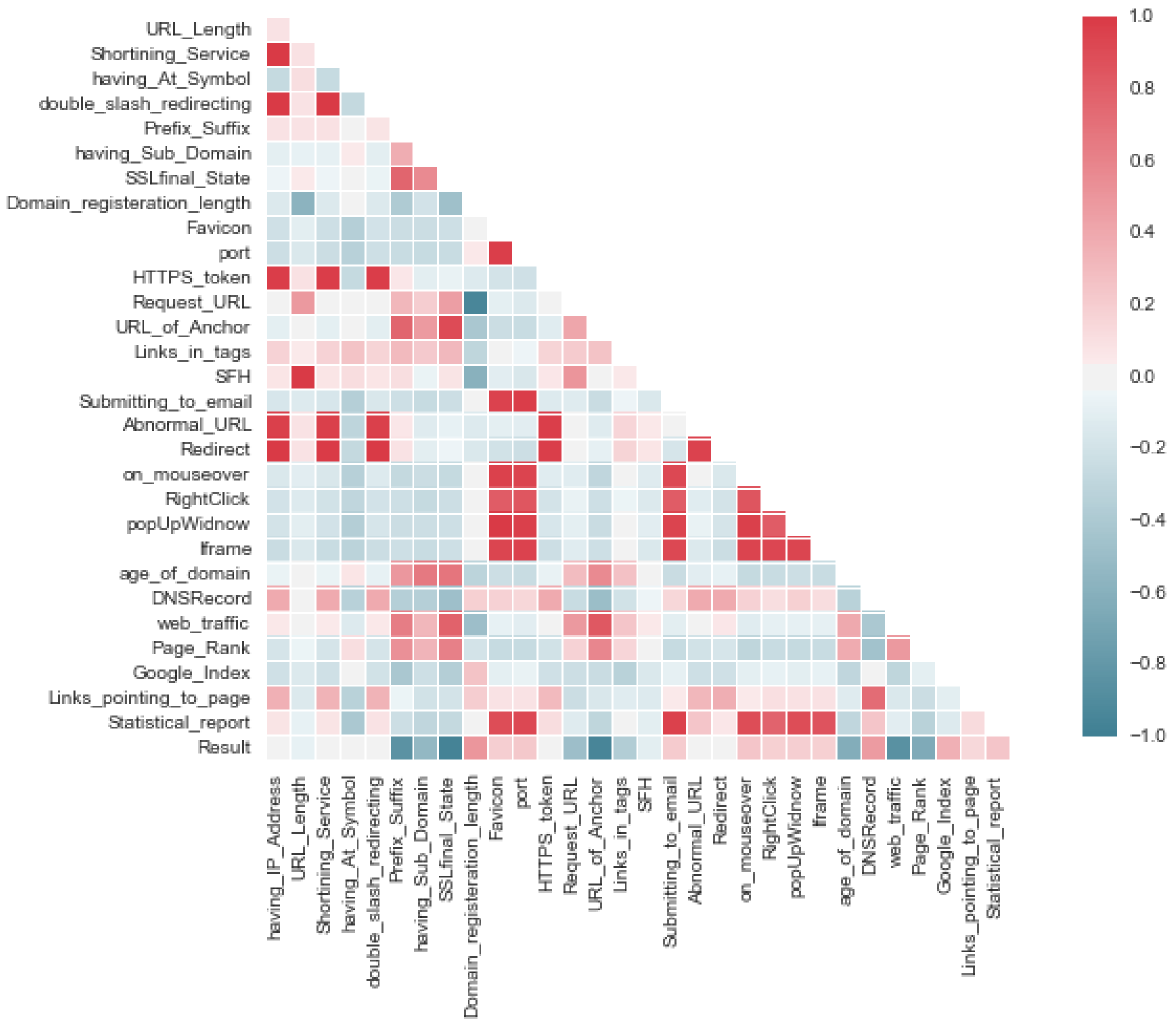
In [142]:
```
# exploring wiht a quick look at Age of Domain feature with a histo
gram visualization
# df.age_of_domain.hist();
```

In [143]:
```
# exploring with a quick look at Web Traffic feature with histogram

visualization
# df.web_traffic.hist();
```

In [17]:
```
df_corr = df.corr()
#correlation matrix varialb e= df.corr
```

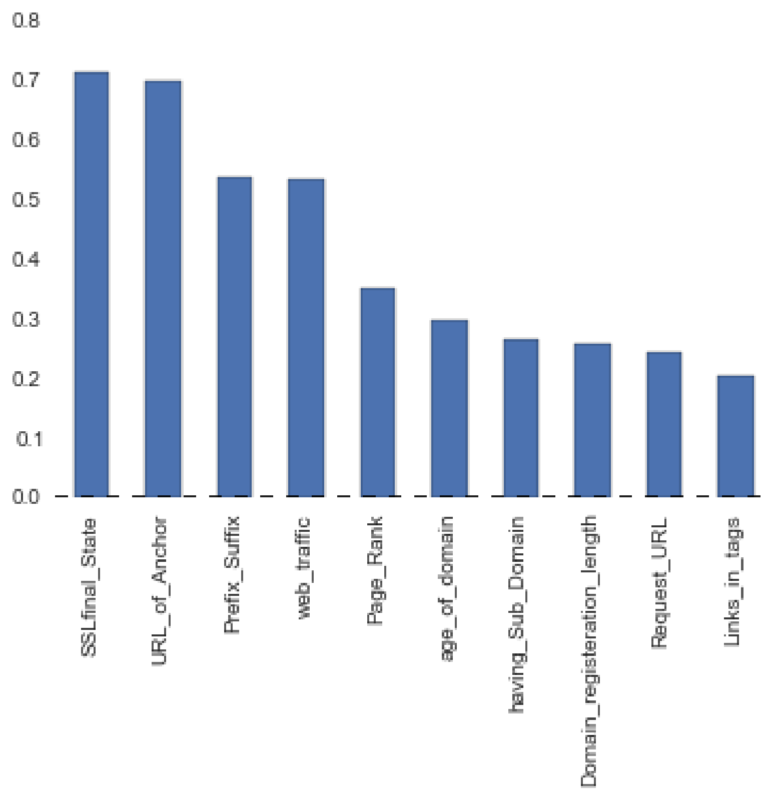
In [118]:
```
# plot each feature for visual understanding
# setting size of plot correlation matrix
f, ax = plt.subplots(figsize=(9, 9))

#color palate
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# plotting actual data
sns.corrplot(df_corr, annot=False, sig_stars=False,
             diag_names=False, cmap=cmap, ax=ax)

#plotting large correlational matrix
```

Out[118]: <matplotlib.axes._subplots.AxesSubplot at 0x10fde06d0>



In [19]:
```
corr_series = df_corr.Result.abs().order(ascending = False)[1:]
# ordering correlation + or - first, from highest to lowest correla
tion,
# getting all in series starting at first integer position [really
always 2nd as 0 is always 1st]
```

In [20]:
```
#plotting up to the 10th
corr_series[:10].plot(kind = 'bar');
```



In [21]:
```
corr_idx = corr_series [:10].index
```

In [22]:
```
corr_idx
```

Out[22]:
```
Index([u'SSLfinal_State', u'URL_of_Anchor', u'Prefix_Suffix', u'we
b_traffic', u'Page_Rank', u'age_of_domain', u'having_Sub_Domain',
u'Domain_registeration_length', u'Request_URL', u'Links_in_tags'],
dtype='object')
```

In [ ]:

In [23]:
```
#plt.rcParams['figure.figsize'] = (10.0, 50.0)

#def plot_notnull(df, col, idx):
#    return df.groupby(col).Result.mean().plot(kind = 'bar', title
= col, ax=axs[idx]);

#fig, axs = plt.subplots(10,1)
#for ix, col in enumerate(corr_idx):

#    print plot_notnull(df, col, ix);
```

for col in corr_idx:

```
#print df.groupby(col).Result.mean()
```

Predictive Model

In [ ]:

In [24]:
```
df.groupby('URL_of_Anchor').Result.mean()
#taking attribute of -1 and adding up all results
#98% if it is -1 it will be either a fishing or a non fishing email
```

Out[24]:
```
URL_of_Anchor
-1             0.988858
 0             0.301165
 1             0.041045
Name: Result, dtype: float64
```

In [ ]:

In [25]:
```
URL_df = df[['URL_of_Anchor', 'Result']]
#group by is taking attribute
```

In [26]:
```
URL_df.head(10)
```

Out[26]:

	URL_of_Anchor	Result
0	-1	1
1	0	1
2	0	1
3	0	1
4	0	0
5	0	0
6	0	0
7	1	0
8	0	0
9	-1	1

In [27]:
```
URL_df[URL_df.URL_of_Anchor == -1]
#Results is 1 when email with attribute of -1
```

Out[27]:

	URL_of_Anchor	Result
0	-1	1
9	-1	1
10	-1	1
11	-1	1
12	-1	1
17	-1	1
18	-1	1
19	-1	1
23	-1	1
29	-1	1
30	-1	1
34	-1	1
41	-1	1
44	-1	1
49	-1	1
53	-1	1

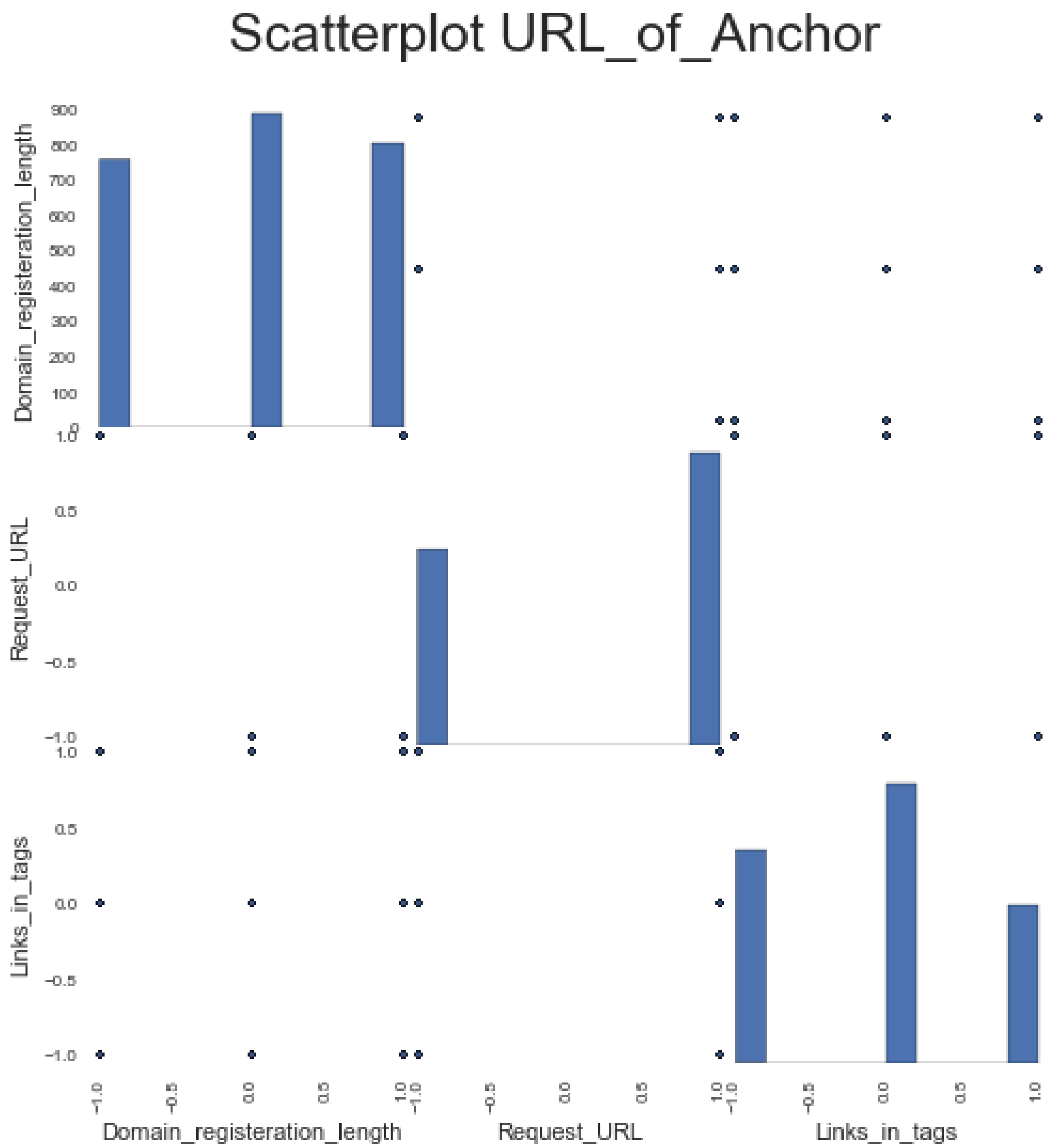
55	-1	1
60	-1	1
64	-1	1
68	-1	1
70	-1	1
72	-1	1
73	-1	1
76	-1	1
78	-1	1
82	-1	1
91	-1	1
92	-1	1
99	-1	1
109	-1	1
...	...	...
2380	-1	1
2381	-1	1
2382	-1	1

In [28]:
```
URL_df[URL_df.URL_of_Anchor == -1].Result.mean()
# interesting strong predictor , though 1s and -1s so we will chang
e to a percentage
```

Out[28]: 0.9888579387186629

In [29]:
```
#more plotting
pd.scatter_matrix(df[[u'Domain_registeration_length',
                      u'Request_URL',
                      u'Links_in_tags']],
                  figsize=(8,8))
plt.suptitle('Scatterplot URL_of_Anchor',size=25)
```

Out[29]:
```
<matplotlib.text.Text at 0x10d185150>
```



Scatter Plot

In [ ]:

In [ ]:

In [30]:
```
#grouping feature to view mean and count
grouped1 = df.groupby('having_IP_Address').Result.agg(('mean', 'cou
nt'))
#list comprehension appending collumn name to indeces
grouped1.index = ['having_IP_Address' + str(string) for string in g
rouped1.index]
grouped1
```

Out[30]:

	mean	count
having_IP_Address0	0.456382	2178
having_IP_Address1	0.359712	278

In [31]:
```
varsGrouped = pd.DataFrame()
# creating for loop for data frame
for col in corr_series.index:

    grouped = df.groupby(col).Result.agg(['mean', 'count'])
# grouping by a single column
    grouped.index = [col + str(string) for string in grouped.index]
    varsGrouped = pd.concat([varsGrouped, grouped])
```

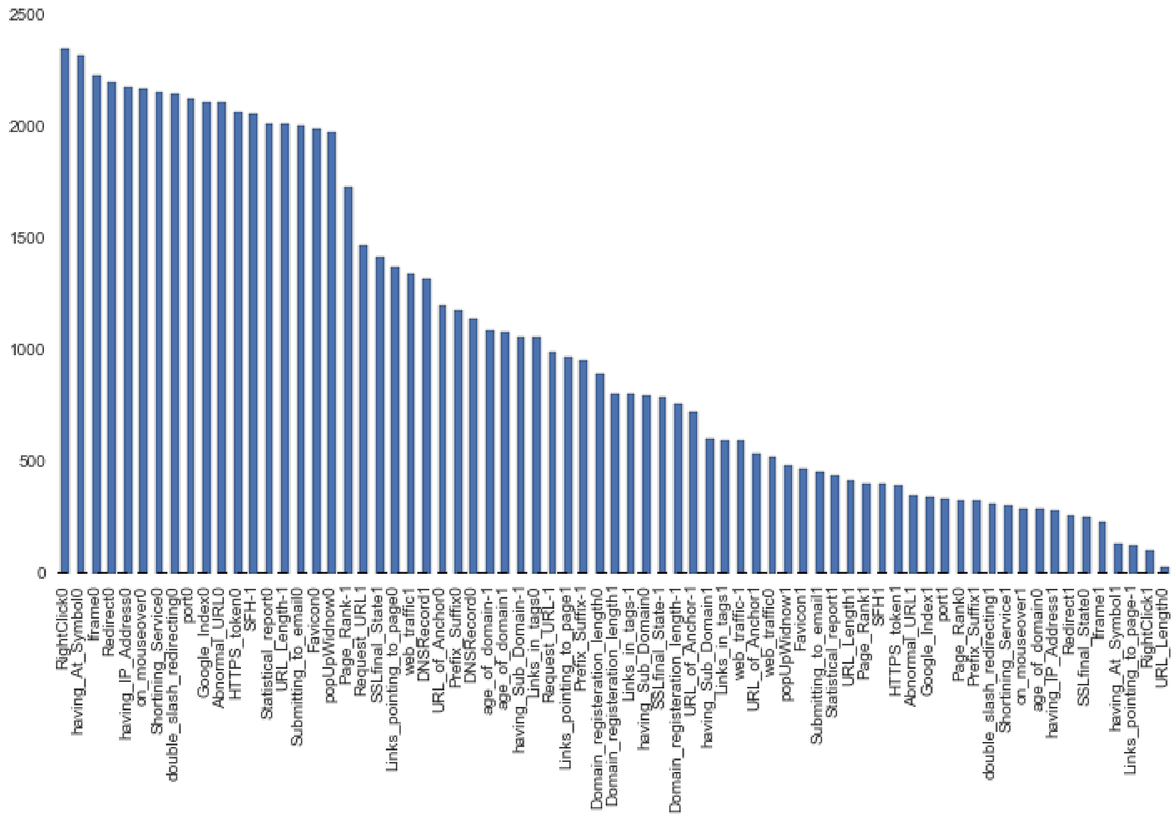
In [148]:
```
varsGrouped
```

Out[148]:

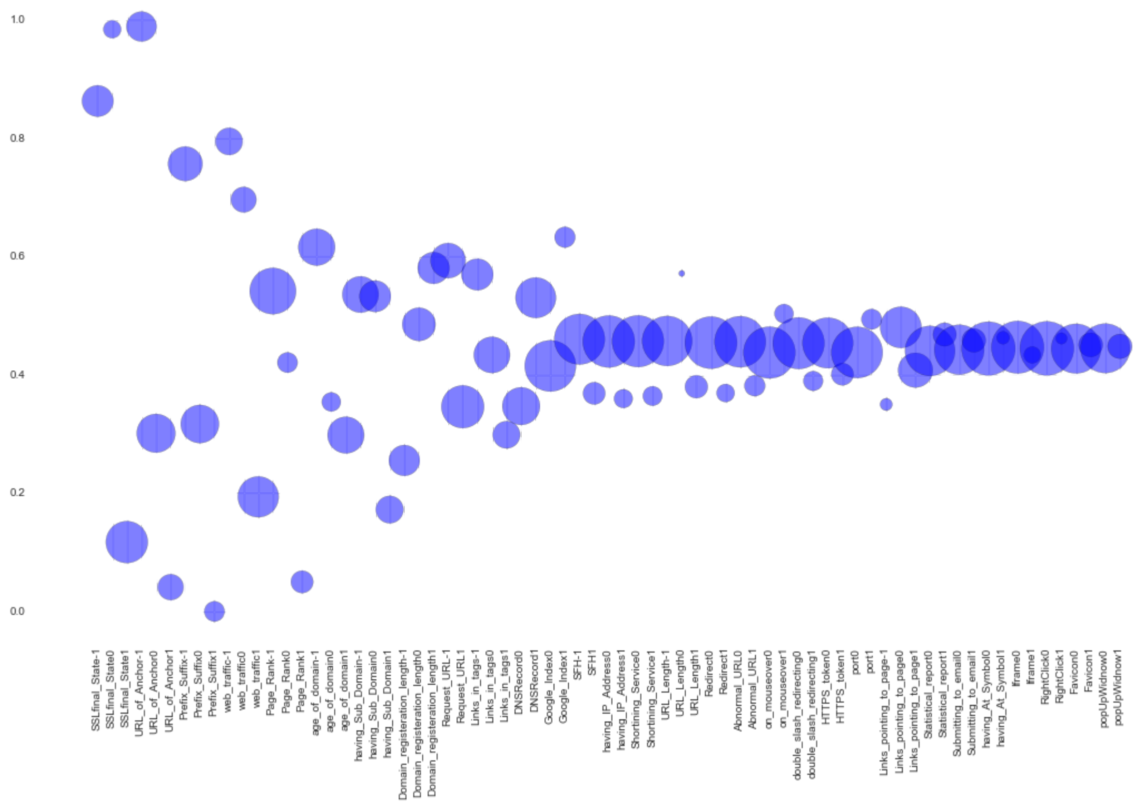
	mean	count
SSLfinal_State-1	0.862944	788
SSLfinal_State0	0.984127	252
SSLfinal_State1	0.117232	1416
URL_of_Anchor-1	0.988858	718
URL_of_Anchor0	0.301165	1202
URL_of_Anchor1	0.041045	536
Prefix_Suffix-1	0.756813	954
Prefix_Suffix0	0.316865	1174
Prefix_Suffix1	0.000000	328
web_traffic-1	0.794613	594
web_traffic0	0.696154	520
web_traffic1	0.193741	1342
Page_Rank-1	0.541667	1728
Page_Rank0	0.420732	328
Page_Rank1	0.050000	400

age_of_domain-1	0.615809	1088
age_of_domain0	0.354167	288
age_of_domain1	0.298148	1080
having_Sub_Domain-1	0.535849	1060
having_Sub_Domain0	0.532828	792
having_Sub_Domain1	0.172185	604
Domain_registeration_length-1	0.255263	760
Domain_registeration_length0	0.485393	890
Domain_registeration_length1	0.580645	806
Request_URL-1	0.593117	988
Request_URL1	0.346049	1468
Links_in_tags-1	0.569652	804
Links_in_tags0	0.433712	1056
Links_in_tags1	0.298658	596
DNSRecord0	0.347100	1138
...	...	...
URL_Length1	0.379808	416
Redirect0	0.454463	2196
Redirect1	0.369231	260

In [146]:
```
#varsGrouped['count']
varsGrouped['count'].order(ascending = False).plot(kind = 'bar', fi
gsize = (12,6));
```



In [32]:
```
#importing scatter plot
from matplotlib.artist import setp
fig = plt.gcf()
# setting axes and size for scatter plot
fig.set_size_inches(18.5, 10.5)
x = range(len(varsGrouped.index))
y = varsGrouped['mean']

# function for setting vertical labels to the x axis
my_xticks = varsGrouped.index

plt.xticks(x, my_xticks)
plt.scatter(x,
            y,
            s = varsGrouped['count'],
            alpha = .5)
plt.xticks(rotation=90)
plt.ylim(-0.05,1.05)
plt.show()
```



In [365]:
```
varsGrouped
# dictionary
# 1 = phishing
# 0 = non phishing

#Prefix_Suffix0 0.316865        1174
# 31% of ^^ = phishing
```

Out[365]:

	mean	count

SSLfinal_State-1	0.862944	788
SSLfinal_State0	0.984127	252
SSLfinal_State1	0.117232	1416
URL_of_Anchor-1	0.988858	718
URL_of_Anchor0	0.301165	1202
URL_of_Anchor1	0.041045	536
Prefix_Suffix-1	0.756813	954
Prefix_Suffix0	0.316865	1174
Prefix_Suffix1	0.000000	328
web_traffic-1	0.794613	594
web_traffic0	0.696154	520
web_traffic1	0.193741	1342
Page_Rank-1	0.541667	1728
Page_Rank0	0.420732	328
Page_Rank1	0.050000	400
age_of_domain-1	0.615809	1088
age_of_domain0	0.354167	288
age_of_domain1	0.298148	1080
having_Sub_Domain-1	0.535849	1060
having_Sub_Domain0	0.532828	792
having_Sub_Domain1	0.172185	604
Domain_registeration_length-1	0.255263	760
Domain_registeration_length0	0.485393	890
Domain_registeration_length1	0.580645	806
Request_URL-1	0.593117	988
Request_URL1	0.346049	1468
Links_in_tags-1	0.569652	804
Links_in_tags0	0.433712	1056
Links_in_tags1	0.298658	596
DNSRecord0	0.347100	1138
...	...	...
URL_Length1	0.379808	416

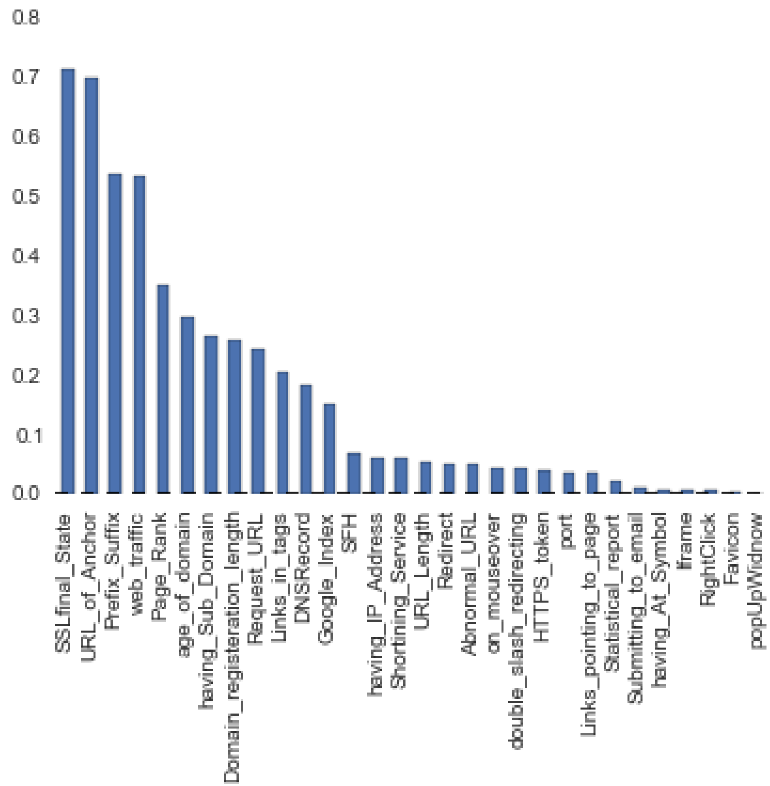
Redirect0	0.454463	2196

In [150]:

```
corr_series.plot(kind = 'bar')
```

Out[150]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11199fe90>
```



In [33]:

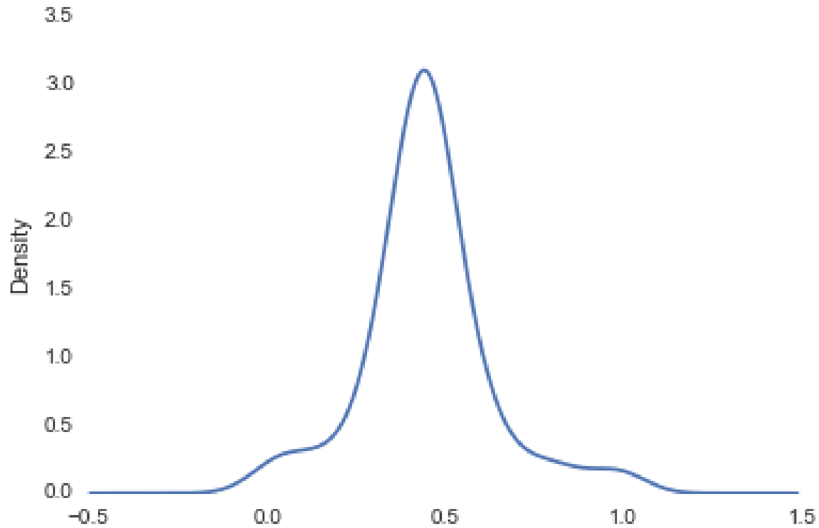
```
# grouping all features to view mean and count, correlation importa
nce
varsGrouped.head()
```

Out[33]:

	mean	count
SSLfinal_State-1	0.862944	788
SSLfinal_State0	0.984127	252
SSLfinal_State1	0.117232	1416
URL_of_Anchor-1	0.988858	718
URL_of_Anchor0	0.301165	1202

In [34]:
```
# should be weighted
varsGrouped['mean'].plot(kind = 'kde')
```

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x10e359850>



In [172]:
```
X
```

Out[172]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	dc
0	1	1	0	0	1
1	0	1	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	1	0	0
5	1	0	1	0	1
6	0	-1	0	0	0
7	0	-1	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0
11	0	0	0	0	0
12	0	0	0	0	0
13	0	1	0	0	0
14	0	-1	0	0	0
15	0	-1	0	0	0
16	1	-1	1	0	1

17	0	-1	0	0	0
18	0	-1	0	0	0
19	0	-1	0	1	0
20	0	1	0	0	0
21	1	1	1	0	1
22	0	-1	0	0	0
23	0	-1	0	0	0
24	0	-1	0	0	0
25	0	-1	0	0	0
26	0	1	0	0	0
27	1	-1	1	0	1
28	0	-1	0	0	0
29	0	-1	0	0	0
...	...	...	...	...	...
2426	1	1	1	0	1
2427	0	-1	0	0	0
2428	0	-1	0	0	0

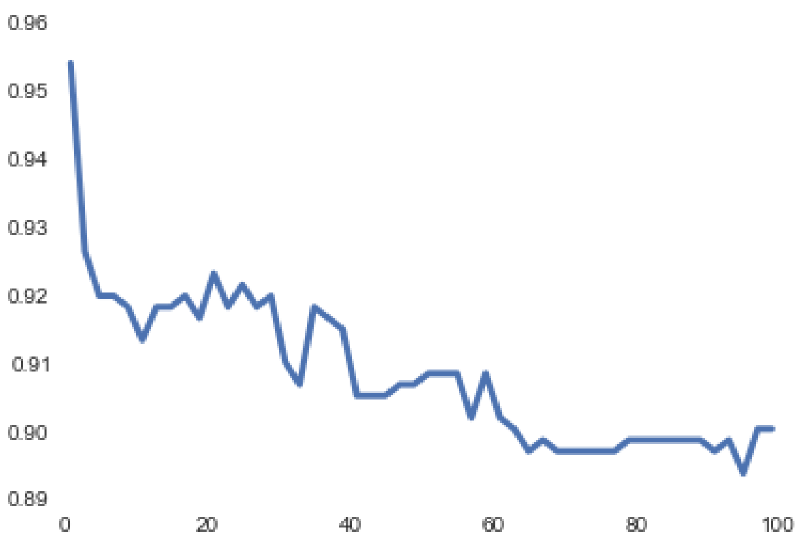
In [184]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(sparse=False)
```

In [224]:
```
# -1 is last collumn, all but Results
# setting = to var
X = df.ix[:,:-1]

# onehotlabelencoder can only use positive integers. adding 1 to en
tire df, therefore 2=1, 1=0, 0=-1
#X = X + 1
#X = enc.fit_transform(X)
# onehotelabel encoder mostly decreased accuracy in models below, t
herefore leaving it out

# Result column
y = df.Result
```

In [225]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_st
ate=1)

from sklearn import neighbors
n_neighbors=range(1, 101, 2)

scores = []
for n in n_neighbors:
    clf = neighbors.KNeighborsClassifier(n)
    clf.fit(X_train, y_train)
    scores.append(clf.score(X_test, y_test))
    # connect data to classification model and predictive ml alogri
thm using Knearest neighbors
# http://www.galvanize.com/blog/2015/05/28/classifying-and-visualiz
ing-musical-pitch-with-k-means-clustering/
```

In [226]:
```
# fiting model x-train, y-train
# accuracy score plotted over 100 values of K
plt.plot(n_neighbors, scores, linewidth=3.0)
```

Out[226]: [<matplotlib.lines.Line2D at 0x117ecc710>]



In [227]:
```
#Knearest neighbors above
# Accuracy score is good, though not very promising for use of pred
iction
```

Grouped features

In [228]:
```
#Using the Random Set (Section III-B), we tokenize each phishing UR
L by splitting it using
#non-alphanumeric characters
```

Graph Accuracy

Modularizing

In [229]:
```
knn = neighbors.KNeighborsClassifier(1)
svc = svm.SVC(kernel='linear', probability=True)
nb = GaussianNB()
lr = LogisticRegression()
rf = RandomForestClassifier(n_estimators=100)
```

In [264]:
```
knn
```

Out[264]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minko
wski',
           metric_params=None, n_neighbors=1, p=2, weights='unifor
m')
```

In [295]:
```
# for parameter names changing to a string splitting on open parent
hesis for purpose of ending as a list
k = str(rf).split('(')
# getting first item in the list
k[0]
```

Out[295]:
```
'RandomForestClassifier'
```

In [296]:
```
# to use in plt_Model
score_dict = {}
```

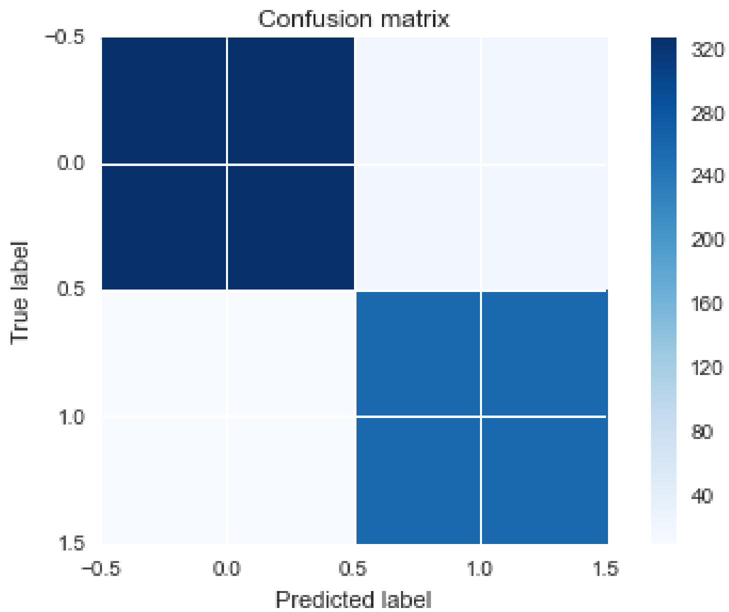
In [230]:
```
def plot_confusion_matrix(cm, cmap, title='Confusion matrix'):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    plt.tight_layout()
    plt.ylabel('True label')
    # True label = what it actually is
    plt.xlabel('Predicted label')
```

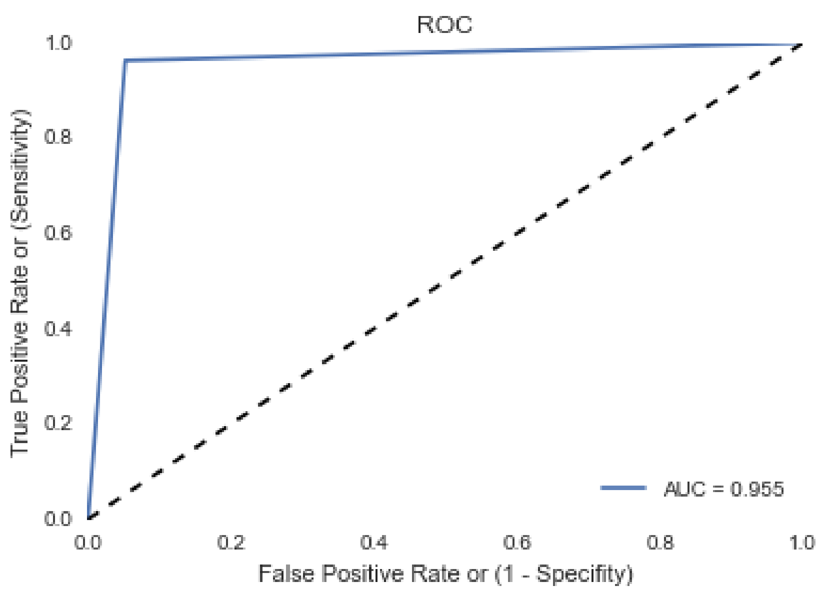
In [231]:
```
def plot_roc_curve(y_test, p_proba):
    # calculates: false positive rate, true positive rate,
    fpr, tpr, thresholds = roc_curve(y_test, p_proba[:, 1])

    roc_auc = auc(fpr, tpr)
    # Plot ROC curve
    plt.plot(fpr, tpr, label= 'AUC = %0.3f' % roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')  # random predictions curve
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('False Positive Rate or (1 - Specifity)')
    plt.ylabel('True Positive Rate or (Sensitivity)')
    plt.title('ROC')
    plt.legend(loc="lower right")
```

In [309]:
```
def plt_Model(X_train, y_train, X_test, y_test, clf, score_dict, cm
ap = plt.cm.Blues):
    fig = plt.figure()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print "Accuracy Score: ", accuracy_score(y_test, y_pred)

    # making score dict
    clf_list = str(clf).split('(')
    score_dict[clf_list[0]] = accuracy_score(y_test, y_pred)

    cm = confusion_matrix(y_test, y_pred)
    print "\nConfusion Matrix:\n", cm
    plot_confusion_matrix(cm, cmap, title='Confusion matrix')
    p_proba = clf.predict_proba(X_test)
    fig = plt.figure()
    plot_roc_curve(y_test, p_proba)
```

In [312]:
```
# Nearest Neighbors
plt_Model(X_train, y_train, X_test, y_test, knn, score_dict, cmap=p
lt.cm.Blues)
```

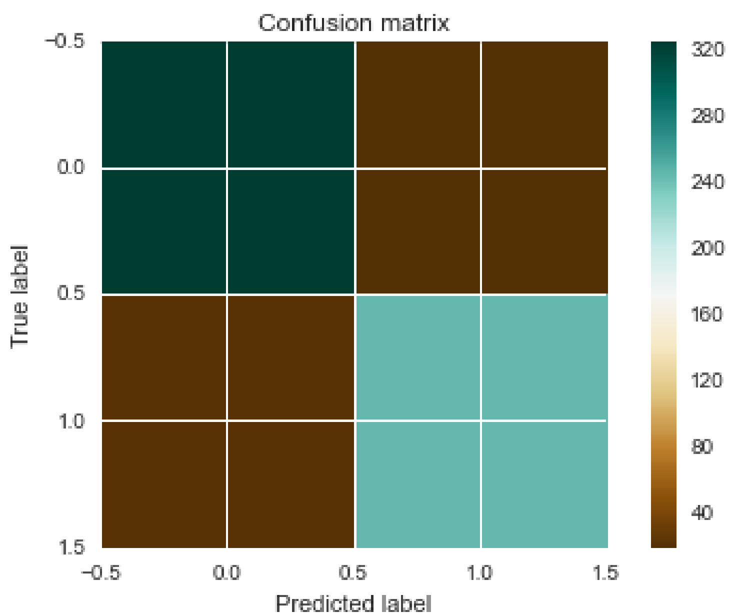
```
Accuracy Score:  0.954397394137

Confusion Matrix:
[[328  18]
 [ 10 258]]
```

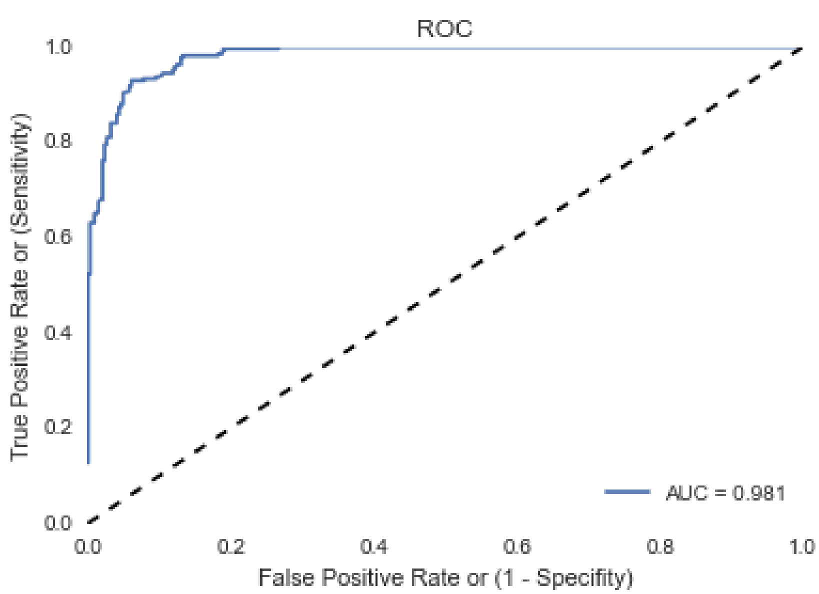




In [313]:
```
#Support Vector Machines (SVMs with rbf kernel)
#SVMs(?? with linear kernel??)
plt_Model(X_train, y_train, X_test, y_test, svc, score_dict, cmap=p
lt.cm.BrBG)
```

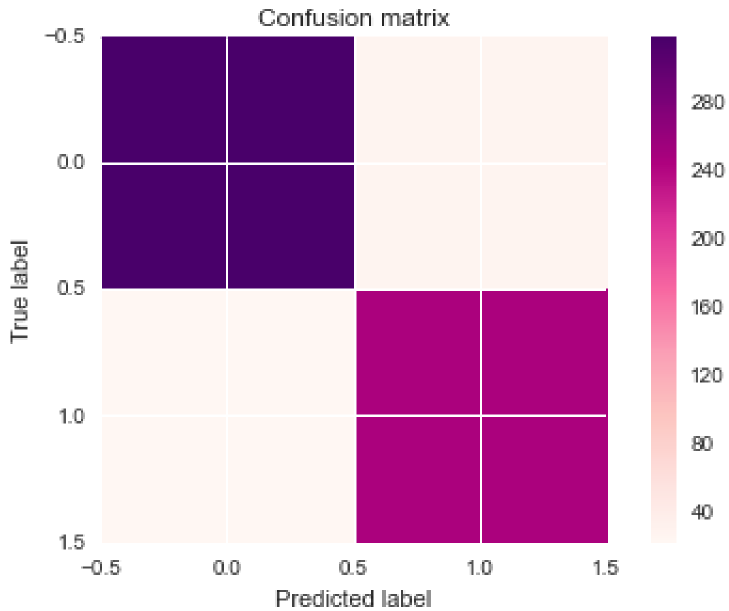
```
Accuracy Score:  0.92996742671

Confusion Matrix:
[[326  20]
 [ 23 245]]
```

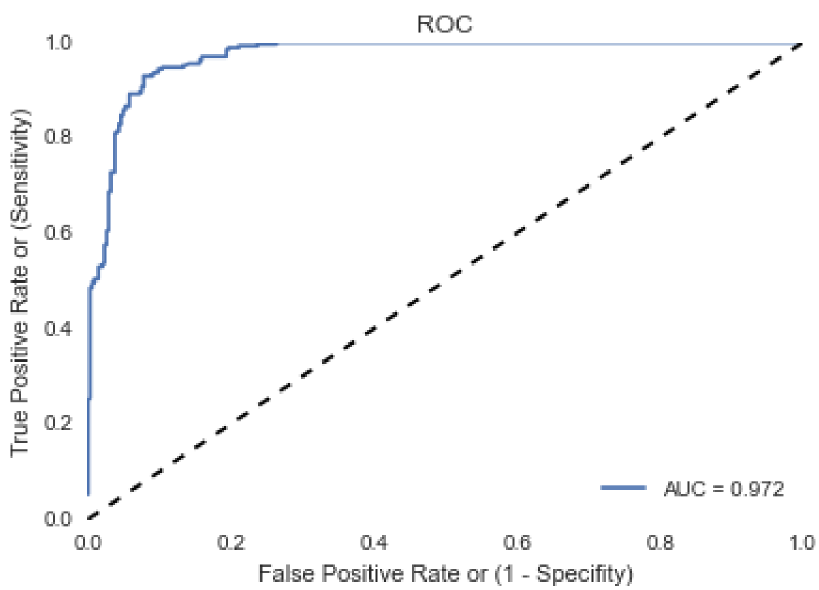




In [314]:
```
# Naïve Bayes (NB)
plt_Model(X_train, y_train, X_test, y_test, nb, score_dict, cmap=pl
t.cm.RdPu)
```

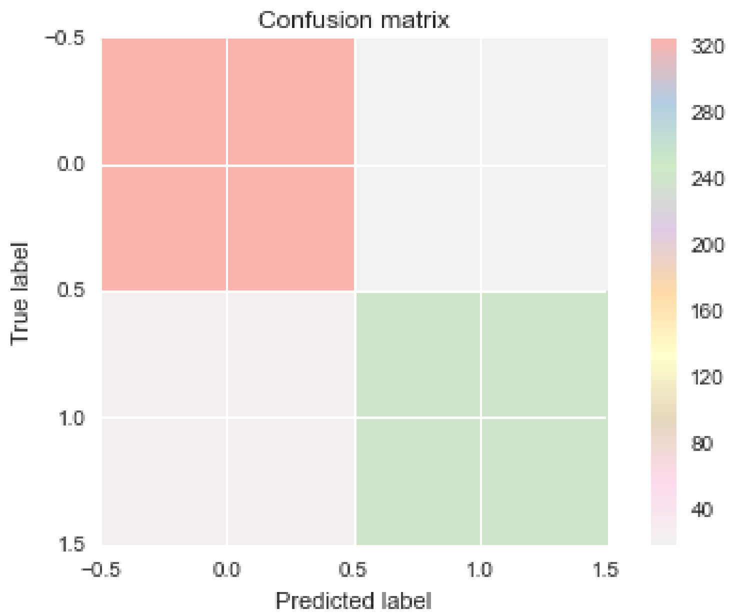
```
Accuracy Score:  0.920195439739

Confusion Matrix:
[[319  27]
 [ 22 246]]
```

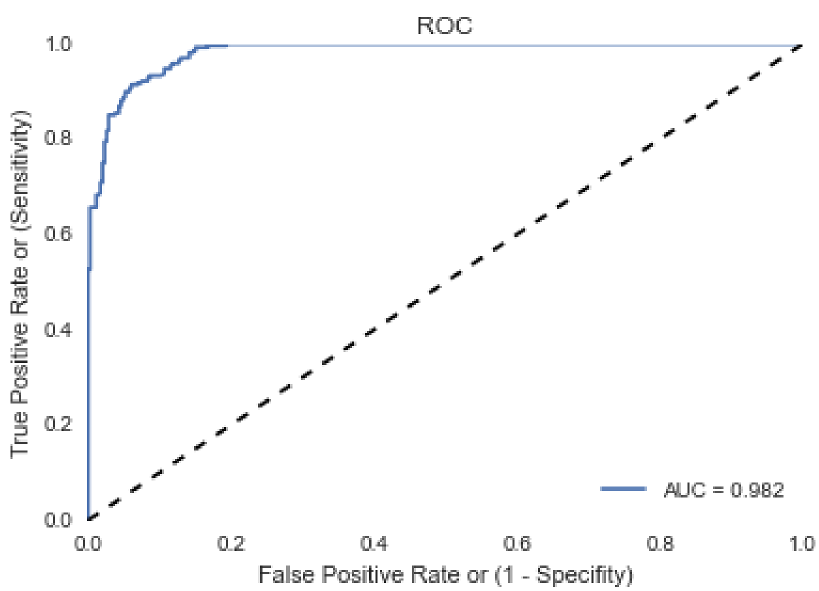




In [315]:
```
# Logistic Regression (LR)
plt_Model(X_train, y_train, X_test, y_test, lr, score_dict, cmap=pl
t.cm.Pastel1_r)
```

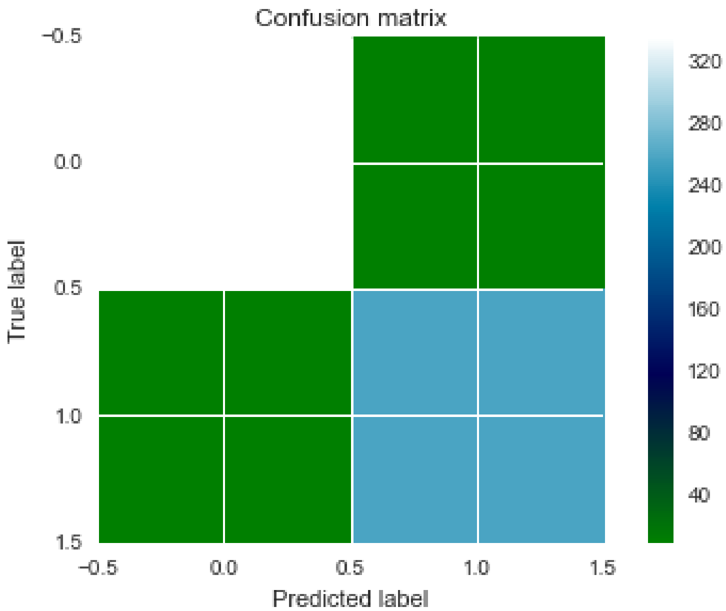
```
Accuracy Score:  0.92671009772

Confusion Matrix:
[[326  20]
 [ 25 243]]
```

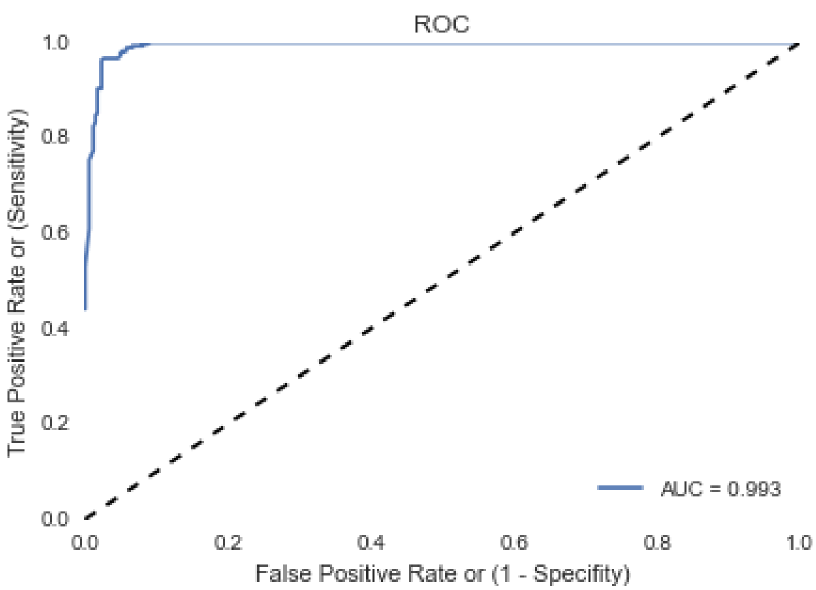




In [316]:
```
# Random Forest (RF)
plt_Model(X_train, y_train, X_test, y_test, rf, score_dict, cmap=pl
t.cm.ocean)
```

```
Accuracy Score:  0.970684039088

Confusion Matrix:
[[337   9]
 [  9 259]]
```





In [317]:
```
rf.fit(X_train, y_train)
```

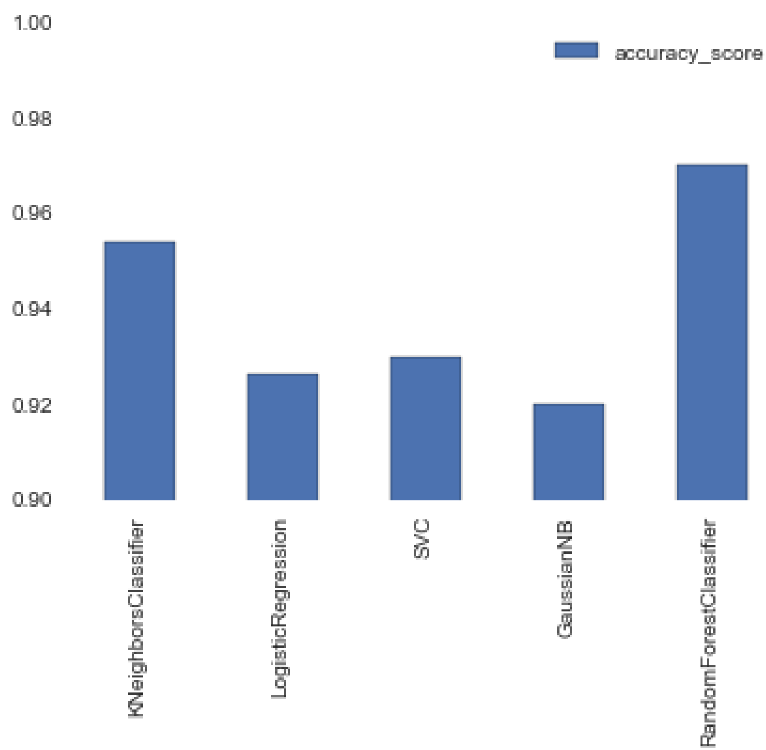
Out[317]:
```
RandomForestClassifier(bootstrap=True, compute_importances=None,
            criterion='gini', max_depth=None, max_features='auto',
            max_leaf_nodes=None, min_density=None, min_samples_lea
f=1,
            min_samples_split=2, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0)
```

In [325]:
```
score_dict.keys()
```

Out[325]:
```
['KNeighborsClassifier',
 'LogisticRegression',
 'SVC',
 'GaussianNB',
 'RandomForestClassifier']
```

In [329]:
```
score_df = pd.DataFrame(score_dict.values(), index = score_dict.key
s(), columns = ['accuracy_score'])
```

In [364]:
```
#plotting accuracy rate of models along same axis
score_df.plot(kind = 'bar', ylim= (.9,1));
```

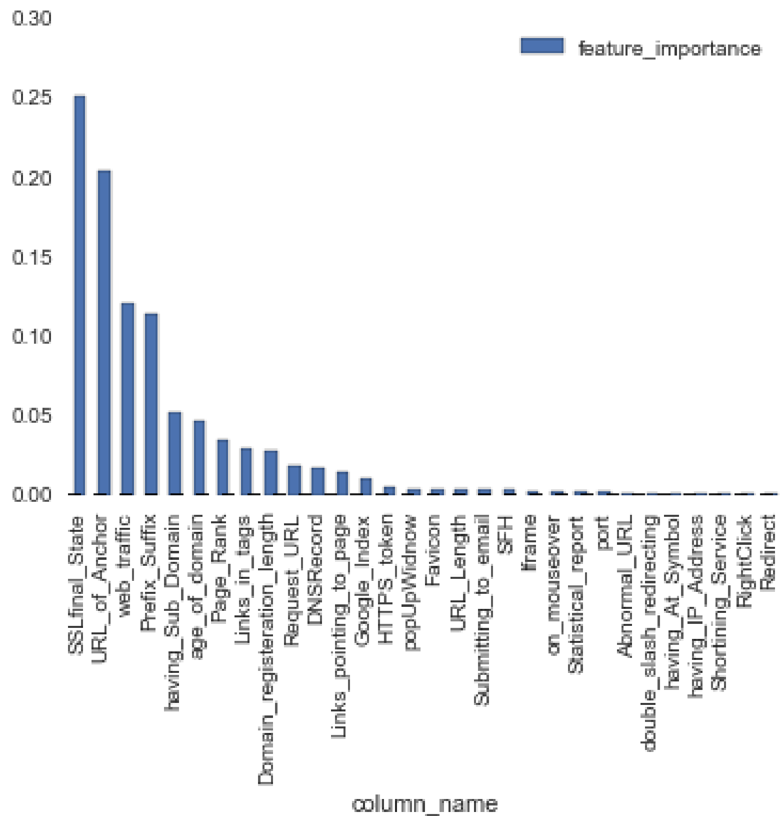


In [333]:
```
fi = sorted(zip(rf.feature_importances_, df.columns), reverse=True)
```

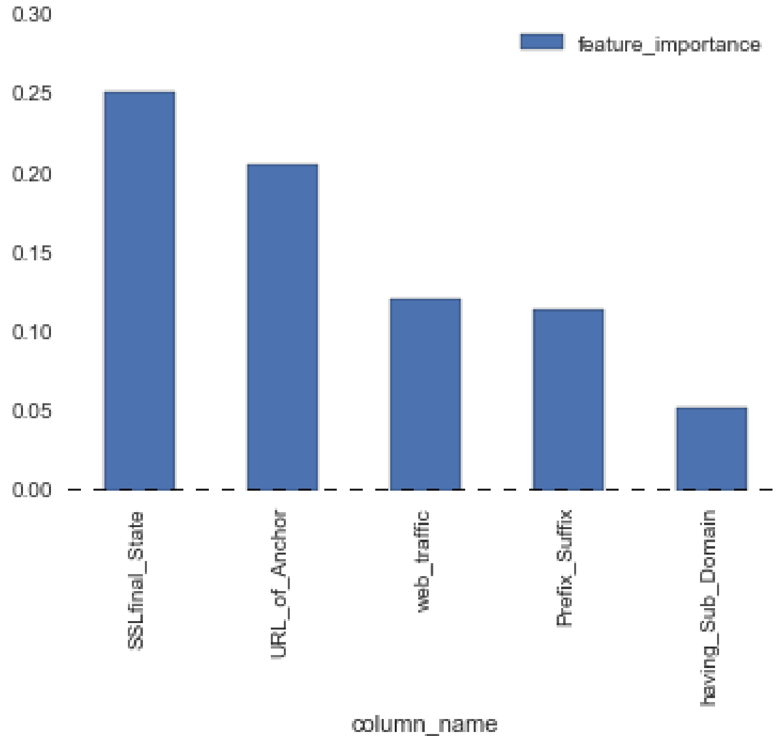
In [341]:
```
fi_df = pd.DataFrame(fi).rename(columns = {0: 'feature_importance',
1 : 'column_name'}).set_index(['column_name'])
```

In [ ]:

In [363]:
```
#plotting of five most important features indicative of most probab
ly to host phishing emails, according to data set.
fi_df.plot(kind = 'bar', ylim= (0,.3));
```



In [355]:
```
fi_df.ix[:5].plot(kind = 'bar', ylim= (0,.3));
```



In [360]:
```
fi_df.head()
#most predictive importance in accuracy for model
```

Out[360]:

	feature_importance
column_name	
SSLfinal_State	0.251322
URL_of_Anchor	0.204961
web_traffic	0.121434
Prefix_Suffix	0.114019
having_Sub_Domain	0.052425

In [240]:
```
# Nearest Neighbors
#plt_Model(X_train, y_train, X_test, y_test, knn,  cmap=plt.cm.Blue
s)
```

In [241]:
```
df.columns
```

Out[241]:
```
Index([u'having_IP_Address', u'URL_Length', u'Shortining_Service',
u'having_At_Symbol', u'double_slash_redirecting', u'Prefix_Suffi
x', u'having_Sub_Domain', u'SSLfinal_State', u'Domain_registeratio
n_length', u'Favicon', u'port', u'HTTPS_token', u'Request_URL',
u'URL_of_Anchor', u'Links_in_tags', u'SFH', u'Submitting_to_emai
l', u'Abnormal_URL', u'Redirect', u'on_mouseover', u'RightClick',
u'popUpWidnow', u'Iframe', u'age_of_domain', u'DNSRecord', u'web_t
raffic', u'Page_Rank', u'Google_Index', u'Links_pointing_to_page',
u'Statistical_report', u'Result'], dtype='object')
```

In [242]:
```
df.describe()
```

Out[242]:

	having_IP_Address	URL_Length	Shortining_Service	having_At_Symbol	d
count	2456.000000	2456.000000	2456.000000	2456.000000	2
mean	0.113192	-0.649837	0.122964	0.054560	0
std	0.316892	0.752690	0.328463	0.227166	0
min	0.000000	-1.000000	0.000000	0.000000	0
25%	0.000000	-1.000000	0.000000	0.000000	0
50%	0.000000	-1.000000	0.000000	0.000000	0
75%	0.000000	-1.000000	0.000000	0.000000	0
max	1.000000	1.000000	1.000000	1.000000	1

8 rows × 31 columns

In [247]:
```
```

In [ ]:
```
# dictionary
# 1 = phishing
# -1 = non phishing
```

In [248]:
```
#upload data to project on GitHub
# http://archive.ics.uci.edu/ml/machine-learning-databases/00327/
# http://archive.ics.uci.edu/ml/machine-learning-databases/00327/Tr
aining%20Dataset.arff
# https://archive.ics.uci.edu/ml/datasets/Phishing+Websites
```