CARLSON SCHOOL OF MANAGEMENT
UNIVERSITY OF MINNESOTA

PREDICTIVE ANALYTICS
MSBA 6420

Homework #3

Hao-Chun, Niu

(put your name above)

Total grade: _____ out of ___100___ points

There are 2 numbered questions. Please answer them all and submit your assignment as a single PDF file by uploading it to the course website.

Quation 1. (50 points) Use numeric prediction techniques to build a predictive model for the HW3.xlsx dataset. This dataset is provided on the course website and contains data about whether or not different consumers made a purchase in response to a test mailing of a certain catalog and, in case of a purchase, how much money each consumer spent. The data file has a brief description of all the attributes in a separate worksheet. Note that this dataset has two possible outcome variables: Purchase (0/1 value: whether or not the purchase was made) and Spending (numeric value: amount spent).

Your tasks:

(a) (20 points) Build numeric prediction models that predict Spending based on the other available customer information (obviously, not including the Purchase attribute among the inputs!). Use linear regression, k-NN, regression tree, SVM regreesion and Neural Network and ensembling models. Briefly discuss your explorations and present the best result (best predictive model) for each of these techniques. Compare the techniques; which of them provides the best predictive performance? Please make sure you use best practices for predictive modeling. (I.e., do you need to set which hyper-parameter? Normalize?)

(b) (20 points) As a variation on this exercise, create a separate "restricted" dataset (i.e., a subset of the original dataset), which includes only purchase records (i.e., where Purchase = 1). Build numeric prediction models to predict Spending for this restricted dataset. All the same requirements as for task (a) apply.

(c) (10 points) For each predictive modeling technique, discuss the predictive performance differences between the models built for task (a) vs. task (b): which models exhibit better predictive performance? Why do you think that is?

In [59]:
```python
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

import numpy as np
np.random.seed(13)

import tensorflow as tf
import pandas as pd
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

import matplotlib.pyplot as plt
import scikitplot as skplt
from sklearn import neighbors, datasets, tree, linear_model, metrics,svm
from sklearn.model_selection import cross_val_score, train_test_split,KFold
import itertools
from itertools import permutations
from sklearn.metrics import recall_score, confusion_matrix, mean_squared_error
from sklearn.model_selection import GridSearchCV
```

1. Import data

In [60]:
```python
df=pd.read_excel('HW3.xlsx')
df=df.drop(columns=['sequence_number','Purchase'])
```

2. Formating the data

In [61]:
```python
c=['US', 'source_a', 'source_c', 'source_b', 'source_d', 'source_e','source_m', 'source_o', 'source_h',
 'source_r',
   'source_s', 'source_t','source_u', 'source_p', 'source_x', 'source_w','Web order','Gender=male', 'Ad
dress_is_res']
df[c] = df[c].astype(str)
```

In [62]:
```python
df.dtypes
```

Out[62]:
```
US                        object
source_a                  object
source_c                  object
source_b                  object
source_d                  object
source_e                  object
source_m                  object
source_o                  object
source_h                  object
source_r                  object
source_s                  object
source_t                  object
source_u                  object
source_p                  object
source_x                  object
source_w                  object
Freq                       int64
last_update_days_ago       int64
1st_update_days_ago        int64
Web order                 object
Gender=male               object
Address_is_res            object
Spending                 float64
dtype: object
```

(a)

In [63]:
```python
#Split Train and Test Data
x_train,x_test,y_train,y_test = train_test_split(df.drop(columns=['Spending']),df['Spending'],test_size
=0.2,random_state=9)

#Standardization
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
mms.fit(x_train[['Freq','last_update_days_ago','1st_update_days_ago']])
x_train[['Freq','last_update_days_ago','1st_update_days_ago']]=mms.transform(x_train[['Freq','last_upda
te_days_ago','1st_update_days_ago']])
x_test[['Freq','last_update_days_ago','1st_update_days_ago']]=mms.transform(x_test[['Freq','last_update
_days_ago','1st_update_days_ago']])
```

In [65]:
```python
#Elastic Net Regression
from sklearn.linear_model import ElasticNet
grid={'alpha':[1,2,3,4,5,6,7,8,9,10],
      'l1_ratio':[0.3,0.5,0.7]}
EN_linear=ElasticNet(random_state=99)
model = GridSearchCV(estimator=EN_linear, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is alpha={} and L1 ratio={}.".format(model.
best_params_['alpha'],model.best_params_['l1_ratio']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is alpha=1 and L1 ratio=0.7.
MSE on Test Data: 31795.78
```

In [66]:
```python
#KNN regression
from sklearn.neighbors import KNeighborsRegressor
grid={'n_neighbors':[5,10,15,20,50,100,500],
      'weights':['uniform','distance']}
KNN=KNeighborsRegressor()
model = GridSearchCV(estimator=KNN, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is n_neighbors={} and weights={}.".format(m
odel.best_params_['n_neighbors'],model.best_params_['weights']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is n_neighbors=5 and weights=distance.
MSE on Test Data: 33890.18
```

In [67]:
```python
#Regression Tree
from sklearn.tree import DecisionTreeRegressor
grid={'max_depth':[2,3,4,5,6,7,8,9,10,20],
      'min_samples_split':[2,5,10,15,20]}
Tree=DecisionTreeRegressor(random_state=99)
model = GridSearchCV(estimator=Tree, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is max_depth={} and min_sample_split={}.".f
ormat(model.best_params_['max_depth'],model.best_params_['min_samples_split']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is max_depth=7 and min_sample_split=15.
MSE on Test Data: 23458.39
```

In [68]:
```python
#SVM Regression
from sklearn.svm import SVR
grid={'kernel':['linear','rbf'],
      'C':[1,2,3,4,5,6,7,8,9,10]}
s=SVR()
model = GridSearchCV(estimator=s, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is kernel={} and C={}.".format(model.best_p
arams_['kernel'],model.best_params_['C']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is kernel=linear and C=10.
MSE on Test Data: 27265.73
```

In [69]:
```python
# Random Search With Neural Network
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
```

In [70]:
```python
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu',input_dim=22))
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu'))
    model.add(layers.Dense(1,activation='linear'))
    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',
                                                            values=[0.01,0.001,0.0001])),
                  loss='mse',
                  metrics=['mse'])
    return model

tuner=RandomSearch(build_model,
                   objective='mse',
                   max_trials=3,
                   overwrite=True,
                   executions_per_trial=3)

#Keras cannot input object data type, so no matter the column is boolean or numeric we need to transfor
m them to float32
x_train_float = np.asarray(x_train).astype(np.float32)
y_train_float = np.asarray(y_train).astype(np.float32)
x_test_float = np.asarray(x_test).astype(np.float32)
y_test_float = np.asarray(y_test).astype(np.float32)

tuner.search(x=x_train_float,y=y_train_float,epochs=200,batch_size=32,validation_data=(x_test_float,y_t
est_float))
```

```
Trial 3 Complete [00h 00m 40s]
mse: 18517.412109375

Best mse So Far: 5621.048177083333
Total elapsed time: 00h 01m 58s
INFO:tensorflow:Oracle triggered exit
```

In [72]:
```python
result=tuner.get_best_hyperparameters()[0].values
print('The best 3 layers NN parameters would be {} neurons and {} learning rate.'.format(result['units'
],result['learning_rate']))
print('------------------------------------------')
print("The best model's mse on test data = 5621")
print('------------------------------------------')
print(tuner.results_summary())
```

```
The best 3 layers NN parameters would be 80 neurons and 0.01 learning rate.
------------------------------------------
The best model's mse on test data = 5621
------------------------------------------
Results summary
Results in ./untitled_project
Showing 10 best trials
Objective(name='mse', direction='min')
Trial summary
Hyperparameters:
units: 80
learning_rate: 0.01
Score: 5621.048177083333
Trial summary
Hyperparameters:
units: 70
learning_rate: 0.0001
Score: 18517.412109375
Trial summary
Hyperparameters:
units: 60
learning_rate: 0.0001
Score: 20536.832682291668
None
```

In [73]:
```python
#Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
grid={'n_estimators':[100,200,300,400,500],
      'max_depth':[1,2,3,4,5,6,7,8,9,10]}
rf=RandomForestRegressor()
model = GridSearchCV(estimator=rf, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is n_estimators={} and max_depth={}.".forma
t(model.best_params_['n_estimators'],model.best_params_['max_depth']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is n_estimators=100 and max_depth=8.
MSE on Test Data: 23006.77
```

Conclusion:

The **3 layers Neural Network with 80 Neurons** in each layer and **learning rate = 0.01** has the best predictive ability. **The MSE on test data is around 5484**.

(b)

In [76]:
```python
df=pd.read_excel('HW3.xlsx')

#Include only the purchase data
df=df[df['Purchase']==1]
df=df.drop(columns=['sequence_number','Purchase'])

#Formating
c=['US', 'source_a', 'source_c', 'source_b', 'source_d', 'source_e','source_m', 'source_o', 'source_h',
 'source_r',
   'source_s', 'source_t','source_u', 'source_p', 'source_x', 'source_w','Web order','Gender=male', 'Ad
dress_is_res']
df[c] = df[c].astype(str)

#Split Train and Test Data
x_train,x_test,y_train,y_test = train_test_split(df.drop(columns=['Spending']),df['Spending'],test_size
=0.2,random_state=9)

#Standardization
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler()
mms.fit(x_train[['Freq','last_update_days_ago','1st_update_days_ago']])
x_train[['Freq','last_update_days_ago','1st_update_days_ago']]=mms.transform(x_train[['Freq','last_upda
te_days_ago','1st_update_days_ago']])
x_test[['Freq','last_update_days_ago','1st_update_days_ago']]=mms.transform(x_test[['Freq','last_update
_days_ago','1st_update_days_ago']])
```

In [77]:
```python
#Elastic Net Regression
from sklearn.linear_model import ElasticNet
grid={'alpha':[1,2,3,4,5,6,7,8,9,10],
      'l1_ratio':[0.3,0.5,0.7]}
EN_linear=ElasticNet(random_state=99)
model = GridSearchCV(estimator=EN_linear, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is alpha={} and L1 ratio={}.".format(model.
best_params_['alpha'],model.best_params_['l1_ratio']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is alpha=1 and L1 ratio=0.7.
MSE on Test Data: 58009.48
```

In [78]:
```python
#KNN regression
from sklearn.neighbors import KNeighborsRegressor
grid={'n_neighbors':[5,10,15,20,50,100,500],
      'weights':['uniform','distance']}
KNN=KNeighborsRegressor()
model = GridSearchCV(estimator=KNN, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is n_neighbors={} and weights={}.".format(m
odel.best_params_['n_neighbors'],model.best_params_['weights']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is n_neighbors=5 and weights=distance.
MSE on Test Data: 62928.14
```

In [79]:
```python
#Regression Tree
from sklearn.tree import DecisionTreeRegressor
grid={'max_depth':[2,3,4,5,6,7,8,9,10,20],
      'min_samples_split':[2,5,10,15,20]}
Tree=DecisionTreeRegressor(random_state=99)
model = GridSearchCV(estimator=Tree, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is max_depth={} and min_sample_split={}.".f
ormat(model.best_params_['max_depth'],model.best_params_['min_samples_split']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is max_depth=3 and min_sample_split=2.
MSE on Test Data: 38713.35
```

In [80]:
```python
#SVM Regression
from sklearn.svm import SVR
grid={'kernel':['linear','rbf'],
      'C':[1,2,3,4,5,6,7,8,9,10]}
s=SVR()
model = GridSearchCV(estimator=s, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is kernel={} and C={}.".format(model.best_p
arams_['kernel'],model.best_params_['C']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is kernel=linear and C=10.
MSE on Test Data: 51984.36
```

In [81]:
```python
# Random Search With Neural Network
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
```

In [82]:
```python
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu',input_dim=22))
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu'))
    model.add(layers.Dense(1,activation='linear'))
    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',
                                                            values=[0.01,0.001,0.0001])),
                  loss='mse',
                  metrics=['mse'])
    return model

tuner=RandomSearch(build_model,
                   objective='mse',
                   max_trials=3,
                   overwrite=True,
                   executions_per_trial=3)

#Keras cannot input object data type, so no matter the column is boolean or numeric we need to transfor
m them to float32
x_train_float = np.asarray(x_train).astype(np.float32)
y_train_float = np.asarray(y_train).astype(np.float32)
x_test_float = np.asarray(x_test).astype(np.float32)
y_test_float = np.asarray(y_test).astype(np.float32)

tuner.search(x=x_train_float,y=y_train_float,epochs=200,batch_size=32,validation_data=(x_test_float,y_t
est_float))
```

```
Trial 3 Complete [00h 00m 27s]
mse: 37634.412760416664

Best mse So Far: 9512.133463541666
Total elapsed time: 00h 01m 20s
INFO:tensorflow:Oracle triggered exit
```

In [83]:
```python
result=tuner.get_best_hyperparameters()[0].values
print('The best 3 layers NN parameters would be {} neurons and {} learning rate.'.format(result['units'
],result['learning_rate']))
print('------------------------------------------')
print("The best model's mse on test data = 9512")
print('------------------------------------------')
print(tuner.results_summary())
```

```
The best 3 layers NN parameters would be 80 neurons and 0.01 learning rate.
------------------------------------------
The best model's mse on test data = 9512
------------------------------------------
Results summary
Results in ./untitled_project
Showing 10 best trials
Objective(name='mse', direction='min')
Trial summary
Hyperparameters:
units: 80
learning_rate: 0.01
Score: 9512.133463541666
Trial summary
Hyperparameters:
units: 60
learning_rate: 0.001
Score: 19139.3359375
Trial summary
Hyperparameters:
units: 90
learning_rate: 0.0001
Score: 37634.412760416664
None
```

In [84]:
```python
#Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
grid={'n_estimators':[100,200,300,400,500],
      'max_depth':[1,2,3,4,5,6,7,8,9,10]}
rf=RandomForestRegressor()
model = GridSearchCV(estimator=rf, param_grid=grid, cv=5,scoring='neg_mean_squared_error')
model.fit(x_train,y_train)
print("With CV grid search, I found the best hyperparameter is n_estimators={} and max_depth={}.".forma
t(model.best_params_['n_estimators'],model.best_params_['max_depth']))
print("MSE on Test Data: {}".format(round((metrics.mean_squared_error(y_test, model.predict(x_test))),2
)))
```

```
With CV grid search, I found the best hyperparameter is n_estimators=400 and max_depth=4.
MSE on Test Data: 39281.85
```

(c)

For full data, including **both purchased and non-purchased data**, **tree based models (Regression Tree, Ramdom Forest Regression) and Neural Network have the better performances**. On the other hand, for **purchased data only**, **tree based models (Regression Tree, Random Forest Regression) and Neural Network also have the better performances**. In general, **after the purchased data is excluded, all models' performances droppped dramatically**. I believe the reason is **because those who do not purchase and spend 0 dollars have very similar traits. On the other hand, those who do purchase and spend more than 0 dollar have very diverse traits**. Therefore, out models' performances drop dramatically.

Quation 2. (50 points) Download the dataset on spam vs. non-spam emails from the following URL: http://archive.ics.uci.edu/ml/datasets/Spambase. Specifically, (i) file "spambase.data" contains the actual data, and (ii) files "spambase.names" and "spambase.DOCUMENTATION" contain the description of the data. This dataset has 4601 records, each record representing a different email message. Each record is described with 58 attributes (indicated in the aforementioned .names file): attributes 1-57 represent various content-based characteristics already extracted from each email message (related to the frequency of certain words or certain punctuation symbols in a message as well as to the usage of capital letters in a message), and the last attribute represents the class label for each message (spam or non-spam).

Task: The general task for this assignment is to build two different models for detecting spam messages (based on the email characteristics that are given): (i) the best possible model that you can build in terms of the overall predictive accuracy (i.e., not taking any cost information into account), and (ii) the best cost-sensitive classification model that you can build in terms of the average misclassification cost.

Some specific instructions for your assignment/write-up:

- **Start working on the assignment early.**
- **Make sure to explore multiple classification techniques (we have learned quite a few of them in the class by now).**
    - **Also, make sure to explore different hyper-param for each technique (for example, try several different values of k for k-NN) to find which configurations work best for this application.**

- **Make sure to explore the impact of various data pre-processing techniques (e.g., normalization).**
- **When building cost-sensitive prediction models, use 10:1 cost ratio for different misclassification errors. (It should be pretty clear which of the two errors – false positive or false negative – is the costlier one in this scenario.)**
- **In general, use best practices when evaluating the models: nested CV, discuss the confusion matrix and some relevant performance metrics (accuracy, precision, recall, f-measure, AUC, average misclassification cost…), show some visual indications of model performance (ROC curves, lift charts).**
- **As a deliverable, produce a write-up describing your aforementioned explorations. Report the performances of different models that you tried. Discuss the best models in two different tasks in detail (which parameters worked best, what was the performance), provide some comparisons. (upload your code as well) Draw some conclusions from the assignment.**

- **Evaluation: 50 points:**
    - **Performance: 30 points (based on the performance achieved by your best reported models).**
    - **Exploration/write-up: 20 points (based on the comprehensiveness of your exploration, i.e., when searching for the best performing model, did you evaluate and report just one or two techniques, or did you try a number of different variations, based on what you know from the class?).**

In [1]:

```python
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

import numpy as np
np.random.seed(13)

import tensorflow as tf
import pandas as pd
from tensorflow.keras import backend as K
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch

import matplotlib.pyplot as plt
import scikitplot as skplt
from sklearn import neighbors, datasets, tree, linear_model, metrics,svm
from sklearn.model_selection import cross_val_score, train_test_split,KFold
import itertools
from itertools import permutations
from sklearn.metrics import recall_score, confusion_matrix
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:19: DeprecationWarning: `import kera
stuner` is deprecated, please use `import keras_tuner`.
```

1. Import Data

In [70]:

```python
df=pd.read_csv('spambase.data',header=None)
df.columns=['word_freq_make','word_freq_address','word_freq_all','word_freq_3d','word_freq_our','word_f
req_over','word_freq_remove','word_freq_internet',
            'word_freq_order','word_freq_mail','word_freq_receive','word_freq_will','word_freq_people',
'word_freq_report','word_freq_addresses',
            'word_freq_free','word_freq_business','word_freq_email','word_freq_you','word_freq_credit',
'word_freq_your','word_freq_font','word_freq_000',
            'word_freq_money','word_freq_hp','word_freq_hpl','word_freq_george','word_freq_650','word_fr
eq_lab','word_freq_labs','word_freq_telnet','word_freq_857',
            'word_freq_data','word_freq_415','word_freq_85','word_freq_technology','word_freq_1999','wor
d_freq_parts','word_freq_pm','word_freq_direct','word_freq_cs',
            'word_freq_meeting','word_freq_original','word_freq_project','word_freq_re','word_freq_edu',
'word_freq_table','word_freq_conference','char_freq_;',
            'char_freq_(','char_freq_[','char_freq_!','char_freq_$','char_freq_#','capital_run_length_av
erage','capital_run_length_longest',
            'capital_run_length_total','spam']
df['spam']=df['spam'].astype(str)
```

2. Train-Test Data Split

In [71]:

```python
x_train,x_test,y_train,y_test = train_test_split(df.drop(columns=['spam']),df['spam'],test_size=0.2,ran
dom_state=9)
```

3. SMOTE to fix imbalance problem

In [72]:

```python
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
x_train, y_train = oversample.fit_resample(x_train, y_train)
```

In [73]:

```python
y_train=y_train.astype(int)
y_test=y_test.astype(int)
```

4. Data Normalization

In [74]:

```python
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler=scaler.fit(x_train)
x_train[x_train.columns] = scaler.transform(x_train[x_train.columns])
x_test[x_test.columns] = scaler.transform(x_test[x_test.columns])
```

5. Nested Grid Search CV to find the best model, using accuracy

In [75]:

```python
# Create the CV
inner_cv = KFold(n_splits=5, shuffle=True, random_state=9)
outer_cv = KFold(n_splits=5, shuffle=True, random_state=9)

# Create the Classifier
def create_model(activation, nb_hidden):
    model = Sequential()
    model.add(Dense(nb_hidden, input_dim=57, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
NN = KerasClassifier(build_fn=create_model, epochs=30, batch_size=256, verbose=0)
t=tree.DecisionTreeClassifier(random_state=9)
knn=neighbors.KNeighborsClassifier()
s=svm.SVC(random_state=9)

# Set up the parameter grid
NN_grid = {'activation':['relu', 'sigmoid'], 'nb_hidden':[10,20,30,40,50,60,70,80,90,100]}
tree_grid={'criterion':['gini','entropy'],
           'max_depth':list(range(30))}
knn_grid={'weights':['uniform','distance'],
          'n_neighbors':list(range(5,31))}
svm_grid={'C':[0.1,1,5,10,50,100],
          'gamma':[1,5,10,15,20,25,30,50,100],
          'kernel':['linear','rbf']}

#Nested CV for Neural Network
clf = GridSearchCV(estimator=NN, param_grid=NN_grid, cv=inner_cv,scoring='accuracy')
nested_score = cross_val_score(clf, X=x_train, y=y_train, cv=outer_cv,scoring='accuracy')
NN_result=nested_score.mean()

#Nested CV for SVM
clf = GridSearchCV(estimator=s, param_grid=svm_grid, cv=inner_cv,scoring='accuracy')
nested_score = cross_val_score(clf, X=x_train, y=y_train, cv=outer_cv,scoring='accuracy')
svm_result=nested_score.mean()

#Nested CV for Decision Tree
clf = GridSearchCV(estimator=t, param_grid=tree_grid, cv=inner_cv,scoring='accuracy')
nested_score = cross_val_score(clf, X=x_train, y=y_train, cv=outer_cv,scoring='accuracy')
tree_result=nested_score.mean()

#Nested CV for KNN
clf = GridSearchCV(estimator=knn, param_grid=knn_grid, cv=inner_cv,scoring='accuracy')
nested_score = cross_val_score(clf, X=x_train, y=y_train, cv=outer_cv,scoring='accuracy')
knn_result=nested_score.mean()

print('Average Performance of Neural Network Classifier: {}%'.format(round(NN_result*100,2)))
print('Average Performance of SVM Classifier: {}%'.format(round(svm_result*100,2)))
print('Average Performance of Decision Tree Classifier: {}%'.format(round(tree_result*100,2)))
print('Average Performance of KNN Classifier: {}%'.format(round(knn_result*100,2)))
```

```
Average Performance of Neural Network Classifier: 94.64%
Average Performance of SVM Classifier: 94.08%
Average Performance of Decision Tree Classifier: 93.14%
Average Performance of KNN Classifier: 91.94%
```

6-1. Use GridSearch to find best hyper-parameters for 4-layers NN model

In [77]:

```python
#Grid Search Verison
NN_grid = {'activation':['relu', 'sigmoid'], 'nb_hidden':[10,20,30,40,50,60,70,80,90,100]}
# Create the Classifier
def create_model(activation, nb_hidden):
    model = Sequential()
    model.add(Dense(nb_hidden, input_dim=57, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
NN = KerasClassifier(build_fn=create_model, epochs=30, batch_size=256, verbose=0)

#Grid Search CV for Neural Network
NNmodel = GridSearchCV(estimator=NN, param_grid=NN_grid, cv=5,scoring='accuracy')
NNmodel.fit(x_train, y_train)

#Result
print('The best parameters activation={} & nb_hidden={}'.format(NNmodel.best_params_['activation'],NNmo
del.best_params_['nb_hidden']))
print("Prediction Accuracy Score on Test Data: {}%".format(round(metrics.accuracy_score(y_test, NNmodel
.predict(x_test))*100,2)))
print("Recall positive on Test Data: {}%".format(round(metrics.recall_score(y_test, NNmodel.predict(x_t
est),pos_label=1)*100,2)))
```

```
The best parameters activation=relu & nb_hidden=100
Prediction Accuracy Score on Test Data: 94.46%
Recall positive on Test Data: 93.8%
```

6-2. Use RandomSearch to find best hyper-parameters for 4-layers NN model

In [78]:

```python
#Random Search Version
def build_model(hp):
    model = keras.Sequential()
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu',input_dim=57))
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu'))
    model.add(layers.Dense(units=hp.Int('units',
                                        min_value=10,
                                        max_value=100,
                                        step=10),
                           activation='relu'))
    model.add(layers.Dense(1,activation='sigmoid'))
    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',
                                                            values=[0.01,0.001,0.0001])),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

tuner=RandomSearch(build_model,
                   objective='accuracy',
                   max_trials=5,
                   overwrite=True,
                   seed=99,
                   executions_per_trial=5)

#Keras cannot input object data type, so no matter the column is boolean or numeric we need to transfor
m them to float32
x_train_float = np.asarray(x_train).astype(np.float32)
y_train_float = np.asarray(y_train).astype(np.float32)
x_test_float = np.asarray(x_test).astype(np.float32)
y_test_float = np.asarray(y_test).astype(np.float32)

tuner.search(x=x_train_float,y=y_train_float,epochs=30,batch_size=512,validation_data=(x_test_float,y_t
est_float))
```

```
Trial 5 Complete [00h 00m 08s]
accuracy: 0.9723672270774841

Best accuracy So Far: 0.9880287885665894
Total elapsed time: 00h 00m 42s
INFO:tensorflow:Oracle triggered exit
```

In [79]:

```python
#Get the best model
NNmodel2=tuner.get_best_models()[0]
```

In [80]:

```python
#result
result=tuner.get_best_hyperparameters()[0].values
print('The best 4-layers NN parameters would be {} neurons and {} learning rate.'.format(result['units'
],result['learning_rate']))
print('------------------------------------------')
print("The best model's accuracy on test data = 98%")
print('------------------------------------------')
print(tuner.results_summary())
```

```
The best 4-layers NN parameters would be 100 neurons and 0.01 learning rate.
------------------------------------------
The best model's accuracy on test data = 98%
------------------------------------------
Results summary
Results in ./untitled_project
Showing 10 best trials
Objective(name='accuracy', direction='max')
Trial summary
Hyperparameters:
units: 100
learning_rate: 0.01
Score: 0.9880287885665894
Trial summary
Hyperparameters:
units: 60
learning_rate: 0.01
Score: 0.9828082680702209
Trial summary
Hyperparameters:
units: 50
learning_rate: 0.01
Score: 0.9807830810546875
Trial summary
Hyperparameters:
units: 30
learning_rate: 0.01
Score: 0.9723672270774841
Trial summary
Hyperparameters:
units: 90
learning_rate: 0.0001
Score: 0.8956345677375793
None
```

7. Plot the ROC Curve & Confusion Matrix

In [81]:

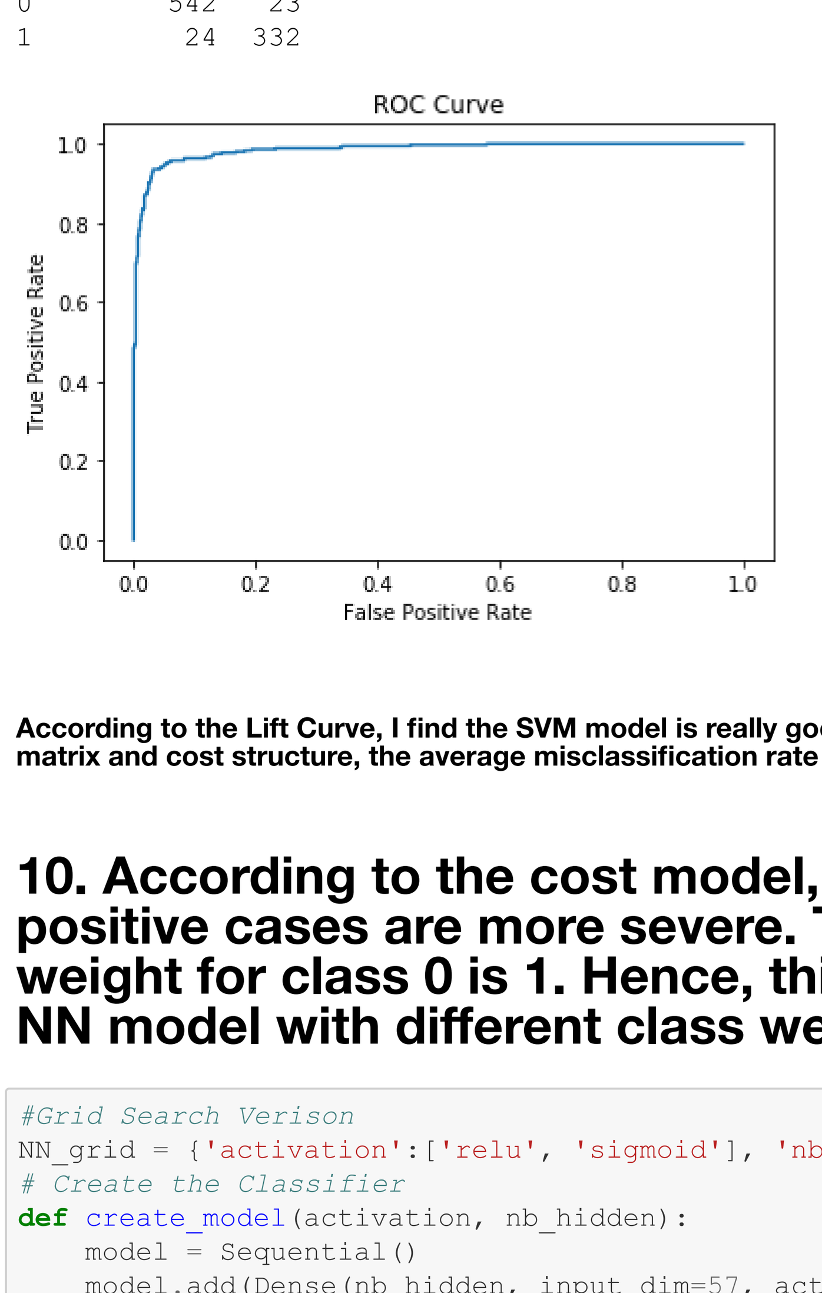
```python
prob=NNmodel2.predict(x_test)
tem=[]
for i in range(len(prob)):
    tem.append(prob[i][0])
prob=tem
y_pred=[]
for i in range(len(prob)):
    if prob[i]>0.5:
        y_pred.append('1')
    else:
        y_pred.append('0')
```

In [83]:

```python
#create the confucion matrix
tem=pd.DataFrame({'True':y_test,'Predict':y_pred})
print(tem.groupby(['Predict','True']).size().unstack())

#create ROC curve
fpr, tpr, _ = metrics.roc_curve(y_test, prob,pos_label=1)
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title('ROC Curve')
plt.show()
```

```
True       0    1
Predict
0        542   23
1         24  332
```



According to the Lift Curve, I find the SVM model is really good at capturing spam cases. In addition according to the confustion matrix and cost structure, the average misclassification rate = (230+24) / 921 = 0.2757

10. According to the cost model, the consequence of mis-classifying positive cases are more severe. The weight for class 1 is 10, and the weight for class 0 is 1. Hence, this time I try to find the best 4-layers NN model with different class weights.

In [84]:

```python
#Grid Search Verison
NN_grid = {'activation':['relu', 'sigmoid'], 'nb_hidden':[10,20,30,40,50,60,70,80,90,100]}
# Create the Classifier
def create_model(activation, nb_hidden):
    model = Sequential()
    model.add(Dense(nb_hidden, input_dim=57, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(nb_hidden, activation=activation))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
NN = KerasClassifier(build_fn=create_model, epochs=30, batch_size=256, verbose=0)

#Grid Search CV for Neural Network
NNmodel3 = GridSearchCV(estimator=NN, param_grid=NN_grid, cv=5,scoring='accuracy')
class_weight = {0: 1.,1:10.}
NNmodel3.fit(x_train, y_train,class_weight=class_weight)

#Result
print('The best parameters activation={} & nb_hidden={}'.format(NNmodel3.best_params_['activation'],NNm
odel3.best_params_['nb_hidden']))
print("Prediction Accuracy Score on Test Data: {}%".format(round(metrics.accuracy_score(y_test, NNmodel
3.predict(x_test))*100,2)))
print("Recall positive on Test Data: {}%".format(round(metrics.recall_score(y_test, NNmodel3.predict(x_
test),pos_label=1)*100,2)))
```

```
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.iter
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_1
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.beta_2
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.decay
WARNING:tensorflow:Unresolved object in checkpoint: (root).optimizer.learning_rate
WARNING:tensorflow:A checkpoint was restored (e.g. tf.train.Checkpoint.restore or tf.keras.Model.load
_weights) but not all checkpointed values were used. See above for specific issues. Use expect_partia
l() on the load status object, e.g. tf.train.Checkpoint.restore(...).expect_partial(), to silence the
se warnings, or use assert_consumed() to make the check explicit. See https://www.tensorflow.org/guid
e/checkpoint#loading_mechanics for details.
The best parameters activation=relu & nb_hidden=100
Prediction Accuracy Score on Test Data: 92.94%
Recall positive on Test Data: 95.77%
```

In [85]:

```python
#create the confucion matrix
y_pred=NNmodel3.predict(x_test)
y_pred=[ y_pred[i][0] for i in range(len(x_test))]
tem=pd.DataFrame({'True':y_test,'Predict':y_pred})
print(tem.groupby(['Predict','True']).size().unstack())
```

```
True       0    1
Predict
0        516   15
1         50  340
```

According to the confustion matrix, after taking class wights into account, the average misclassification rate drop dramatically to (150+50) / 921 = 0.217.