manxilu_telecom_user_churn_analysis

October 2, 2018

1 User Churn Prediction

In this project, we use supervised learning models to identify customers who are likely to stop using service in the future. Furthermore, we will analyze top factors that influence user retention.

1.1 Contents

2 Part 1: Data Exploration

2.0.1 Part 1.1: Understand the Raw Dataset

Data Source: https://www.sgi.com/tech/mlc/db/churn.all
Data info: https://www.sgi.com/tech/mlc/db/churn.names

```
In [1]: import warnings
        warnings.filterwarnings('ignore')

        import pandas as pd
        import numpy as np
        pd.set_option('display.max_columns', None)

        churn_df = pd.read_csv('churn.all')
        print churn_df.head()
        print churn_df.info()
```

```
  state  account_length  area_code phone_number intl_plan voice_mail_plan  \
0    KS             128        415     382-4657        no             yes
1    OH             107        415     371-7191        no             yes
2    NJ             137        415     358-1921        no              no
3    OH              84        408     375-9999       yes              no
4    OK              75        415     330-6626       yes              no
```

```
   number_vmail_messages  total_day_minutes  total_day_calls  \
0                     25              265.1              110
1                     26              161.6              123
2                      0              243.4              114
3                      0              299.4               71
4                      0              166.7              113

   total_day_charge  total_eve_minutes  total_eve_calls  total_eve_charge  \
0             45.07              197.4               99             16.78
1             27.47              195.5              103             16.62
2             41.38              121.2              110             10.30
3             50.90               61.9               88              5.26
4             28.34              148.3              122             12.61

   total_night_minutes  total_night_calls  total_night_charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total_intl_minutes  total_intl_calls  total_intl_charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   number_customer_service_calls  churned
0                              1   False.
1                              1   False.
2                              0   False.
3                              2   False.
4                              3   False.
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
state                            5000 non-null object
account_length                   5000 non-null int64
area_code                        5000 non-null int64
phone_number                     5000 non-null object
intl_plan                        5000 non-null object
voice_mail_plan                  5000 non-null object
number_vmail_messages            5000 non-null int64
total_day_minutes                5000 non-null float64
total_day_calls                  5000 non-null int64
total_day_charge                 5000 non-null float64
```

```
total_eve_minutes                5000 non-null float64
total_eve_calls                  5000 non-null int64
total_eve_charge                 5000 non-null float64
total_night_minutes              5000 non-null float64
total_night_calls                5000 non-null int64
total_night_charge               5000 non-null float64
total_intl_minutes               5000 non-null float64
total_intl_calls                 5000 non-null int64
total_intl_charge                5000 non-null float64
number_customer_service_calls    5000 non-null int64
churned                          5000 non-null object
dtypes: float64(8), int64(8), object(5)
memory usage: 820.4+ KB
None
```

In [2]:
```
# check if data is inbalanced for classification problem
churn_df.churned.value_counts()
```

Out[2]:
```
 False.    4293
 True.      707
Name: churned, dtype: int64
```

In [3]:
```
# check two related features
print churn_df.state.value_counts()
print churn_df.area_code.value_counts()
```

```
WV    158
MN    125
AL    124
ID    119
VA    118
OH    116
TX    116
WY    115
NY    114
OR    114
NJ    112
UT    112
WI    106
ME    103
MA    103
MI    103
MD    102
VT    101
KY     99
KS     99
CT     99
MT     99
```

```
MS      99
RI      99
WA      98
IN      98
CO      96
NH      95
DE      94
MO      93
AR      92
NM      91
SC      91
NC      91
FL      90
NV      90
OK      90
AZ      89
TN      89
IL      88
DC      88
NE      88
ND      88
HI      86
SD      85
GA      83
LA      82
PA      77
AK      72
IA      69
CA      52
Name: state, dtype: int64
415    2495
408    1259
510    1246
Name: area_code, dtype: int64
```

- In quick scan, there is no missing value but data is inbalanced;
- area code is associated with state, but it might be different. This factor only contains three areas probably because data is only collected from the three local stores, which may cause biased sampling;
- phone_number is like userID, except for the case to use phone number to search for speacific person information, in this case it should be ignored
- account length are assumed to the total lenght of the user usage
- people who don't have voice_mail_plan will not have voice mail message
  % check by churn_df.loc[(churn_df["voice_mail_plan"]=="no") &(churn_df["number_vmail_messages"]!=0)]
- number_customer_service_calls may be a good feature that affect user experience

2.0.2 Part 1.2: Data cleaning

In [4]:
```
# plan choice converted to 1,0
yes_no_cols = ["intl_plan","voice_mail_plan"]
churn_df[yes_no_cols] = churn_df[yes_no_cols] == ' yes'
churn_df[yes_no_cols] = churn_df[yes_no_cols]*1

# churned converted to 1,0
churn_df["churned"]=churn_df["churned"].map(lambda x: x.strip())

# "phone_number" and "state" are removed
churn_area=churn_df.drop(["phone_number","state"],axis=1)
churn_state=churn_df.drop(["phone_number","area_code"],axis=1)
churn_area.describe().round(2)
```

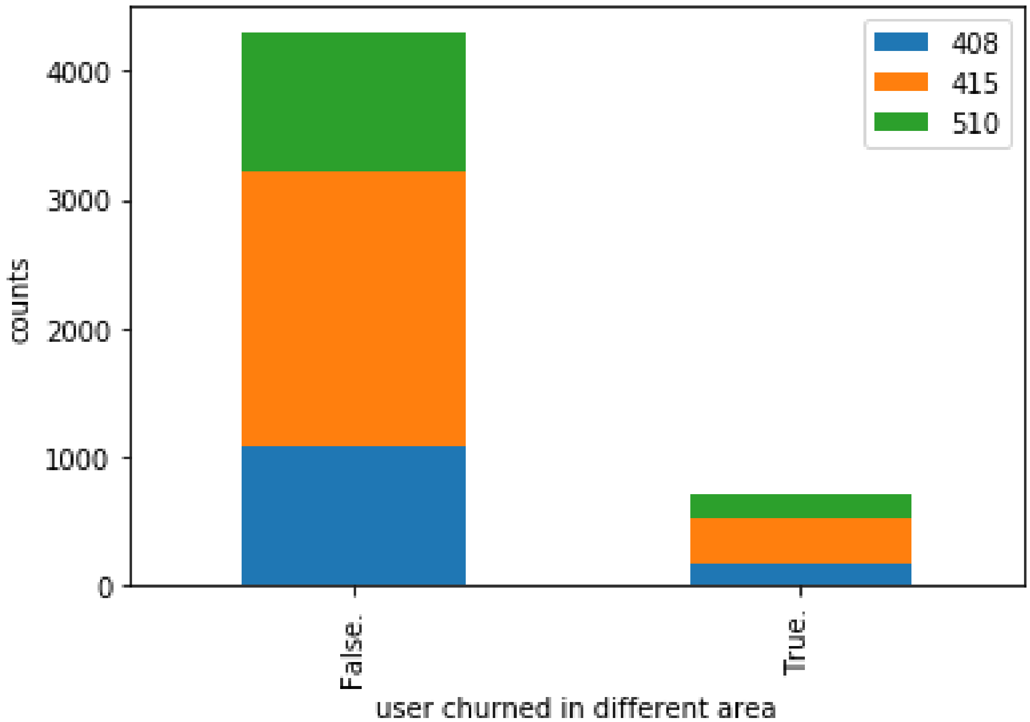
Out[4]:
```
       account_length  area_code  intl_plan  voice_mail_plan  \
count         5000.00    5000.00    5000.00          5000.00
mean           100.26     436.91       0.09             0.26
std             39.69      42.21       0.29             0.44
min              1.00     408.00       0.00             0.00
25%             73.00     408.00       0.00             0.00
50%            100.00     415.00       0.00             0.00
75%            127.00     415.00       0.00             1.00
max            243.00     510.00       1.00             1.00

       number_vmail_messages  total_day_minutes  total_day_calls  \
count                5000.00            5000.00          5000.00
mean                    7.76             180.29           100.03
std                    13.55              53.89            19.83
min                     0.00               0.00             0.00
25%                     0.00             143.70            87.00
50%                     0.00             180.10           100.00
75%                    17.00             216.20           113.00
max                    52.00             351.50           165.00

       total_day_charge  total_eve_minutes  total_eve_calls  total_eve_charge  \
count           5000.00            5000.00          5000.00           5000.00
mean              30.65             200.64           100.19             17.05
std                9.16              50.55            19.83              4.30
min                0.00               0.00             0.00              0.00
25%               24.43             166.38            87.00             14.14
50%               30.62             201.00           100.00             17.09
75%               36.75             234.10           114.00             19.90
max               59.76             363.70           170.00             30.91

       total_night_minutes  total_night_calls  total_night_charge  \
count              5000.00            5000.00             5000.00
mean                200.39              99.92                9.02
std                  50.53              19.96                2.27
```

```
min                  0.00            0.00              0.00
25%                166.90           87.00              7.51
50%                200.40          100.00              9.02
75%                234.70          113.00             10.56
max                395.00          175.00             17.77

       total_intl_minutes  total_intl_calls  total_intl_charge  \
count             5000.00           5000.00            5000.00
mean                10.26              4.44               2.77
std                  2.76              2.46               0.75
min                  0.00              0.00               0.00
25%                  8.50              3.00               2.30
50%                 10.30              4.00               2.78
75%                 12.00              6.00               3.24
max                 20.00             20.00               5.40

       number_customer_service_calls
count                        5000.00
mean                            1.57
std                             1.31
min                             0.00
25%                             1.00
50%                             1.00
75%                             2.00
max                             9.00
```

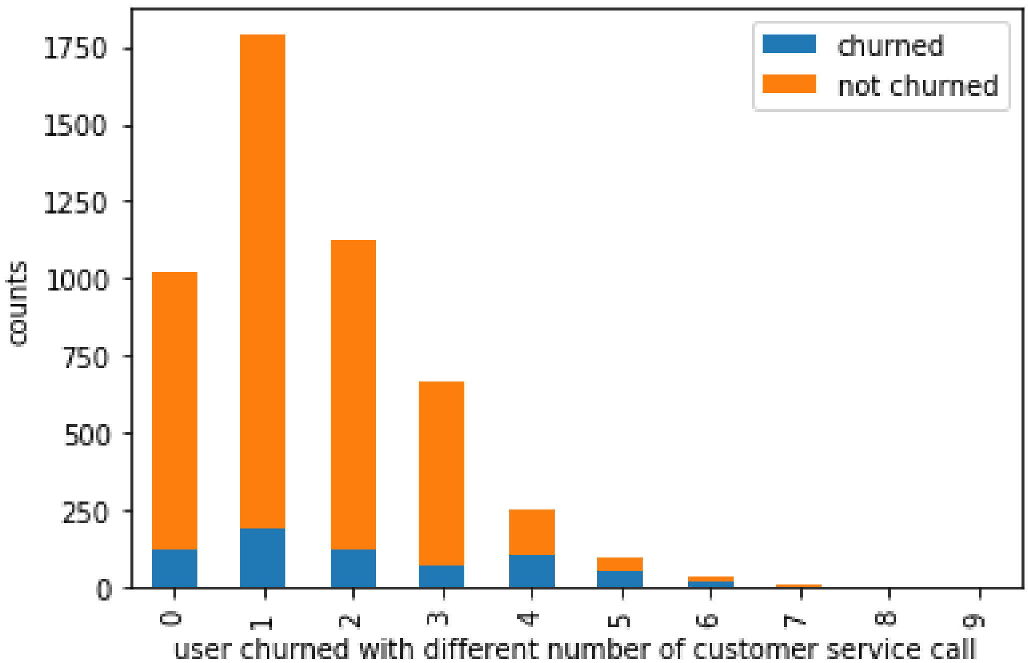
2.0.3 Part 1.3: Understand the features

In [5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sb

churn_415 = churn_area.churned[churn_area.area_code == 415].value_counts()
churn_408 = churn_area.churned[churn_area.area_code == 408].value_counts()
churn_510 = churn_area.churned[churn_area.area_code == 510].value_counts()

df1=pd.DataFrame({u'415':churn_415, u'408':churn_408,
                  u'510':churn_510})
df1.plot.bar(stacked=True)
plt.xlabel(u"user churned in different area")
plt.ylabel(u"counts")
```

Out[5]:
```
Text(0,0.5,u'counts')
```



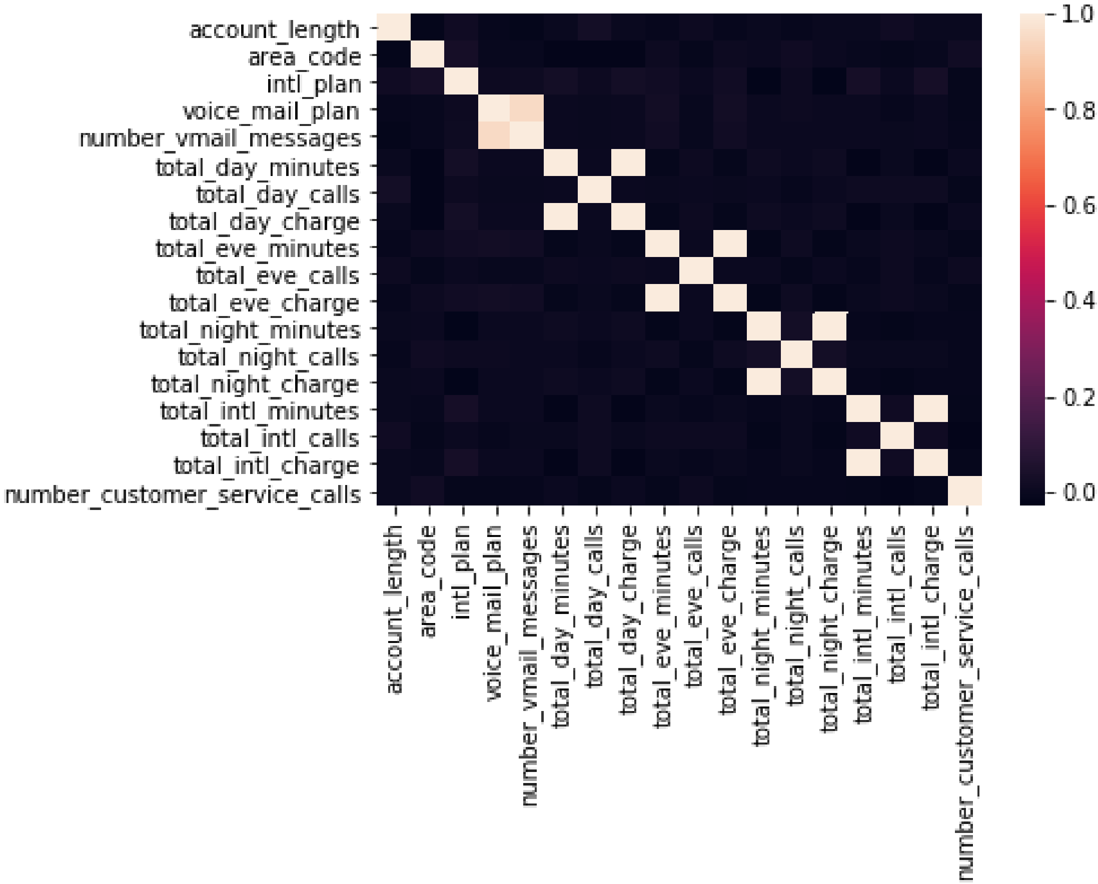

```
In [6]: churn_0_cs = churn_area.number_customer_service_calls[churn_area.churned=="False."].val
        churn_1_cs = churn_area.number_customer_service_calls[churn_area.churned=="True."].valu

        df2=pd.DataFrame({u'not churned':churn_0_cs, u'churned':churn_1_cs})
        df2.plot.bar(stacked=True)
        plt.xlabel(u"user churned with different number of customer service call")
        plt.ylabel(u"counts")

Out[6]: Text(0,0.5,u'counts')
```




```
In [7]: plt.scatter(churn_area.churned, churn_area.account_length)

Out[7]: <matplotlib.collections.PathCollection at 0x1a15435890>
```

```
In [8]: # find the correlation between feature
        corr = churn_area.corr()
        sb.heatmap(corr)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a154491d0>



high correlation: * voice_mail_plan & number_vmail_messages * total_day_minutes & total_day_charge * total_eve_minutes & total_eve_charge * total_night_minutes & total_night_charge * total_intl_minutes & total_intl_charge

3 Part 2: Feature Preprocessing

```
In [9]: # use one-hot-encoding to convert area_code
        churn_area["area_code"]=churn_area["area_code"].map(lambda x: str(x))
        churn_feat = churn_area.drop(["churned"], axis=1)
        churn_feat=pd.get_dummies(churn_feat)

        # convert data into np-array format
        y = np.where(churn_df['churned'] == 'True.',1,0)
        X = churn_feat.as_matrix().astype(np.float)
```

```
In [10]: # Scale the data
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X = scaler.fit_transform(X)

         print "Feature space holds %d observations and %d features" % X.shape
         print "Unique target labels:", np.unique(y)
```

```
Feature space holds 5000 observations and 20 features
Unique target labels: [0 1]
```

4 Part 3: Model Training and Result Evaluation

4.0.1 Part 3.1: K-fold Cross-Validation and Learning Curve

```
In [11]: from sklearn.model_selection import KFold
         from sklearn.model_selection import learning_curve

         #This program does 5-fold. It saves the result at each time as different parts of y_p
         #In the end, it returns the y_pred as the result of all the five 5-fold.
         def run_cv(X,y,clf_class,**kwargs):
             # Construct a kfolds object
             kf = KFold(n_splits=5,shuffle=True)
             y_pred = y.copy()
             clf = clf_class(**kwargs)
             # Iterate through folds
             for train_index, test_index in kf.split(X):
                 X_train, X_test = X[train_index], X[test_index]
                 y_train = y[train_index]

                 clf.fit(X_train,y_train)
                 y_pred[test_index] = clf.predict(X_test)
             return y_pred, clf

         # NumPy interpretes True and False as 1. and 0.
         def accuracy(y_true,y_pred):
             return np.mean(y_true == y_pred)

         # plot learning curve
         def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                                 n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
             plt.figure()
             plt.title(title)
             if ylim is not None:
                 plt.ylim(*ylim)
             plt.xlabel("Training examples")
             plt.ylabel("Score")
```

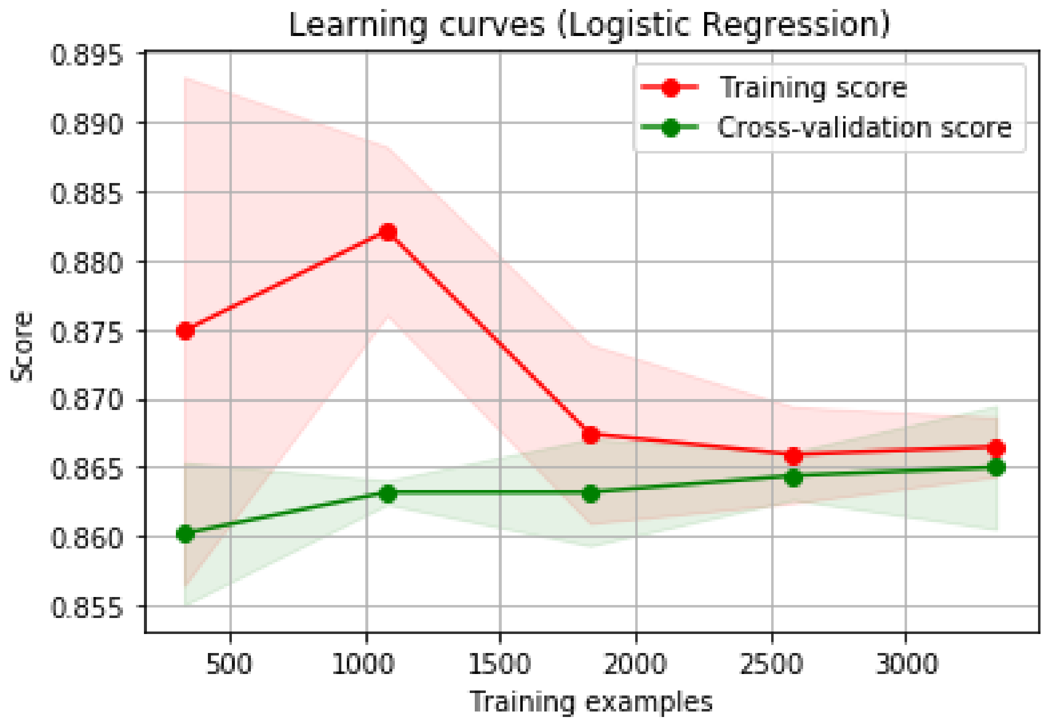
```
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

4.0.2 Part 3.2: Run Supervised Learning Models and Calculate Accuracy

```
In [12]: from sklearn.linear_model import LogisticRegression
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.ensemble import RandomForestClassifier

         LR_CV_result = run_cv(X,y,LogisticRegression)

         print "Logistic Regression (L2 is default): " + str(accuracy(y, LR_CV_result[0]))
         plot_learning_curve(LR_CV_result[1], "Learning curves (Logistic Regression)", X,y)
```

```
Logistic Regression (L2 is default): 0.8634
```

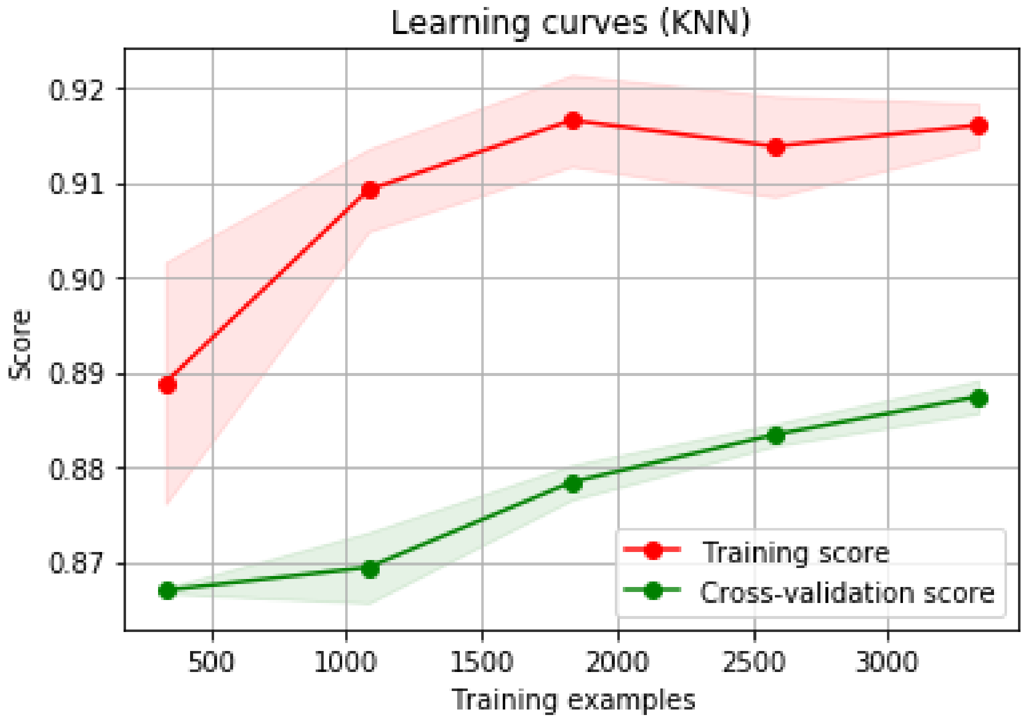
```
Out[12]: <module 'matplotlib.pyplot' from '/Users/manxilu/Applications/anaconda2/lib/python2.7/
```



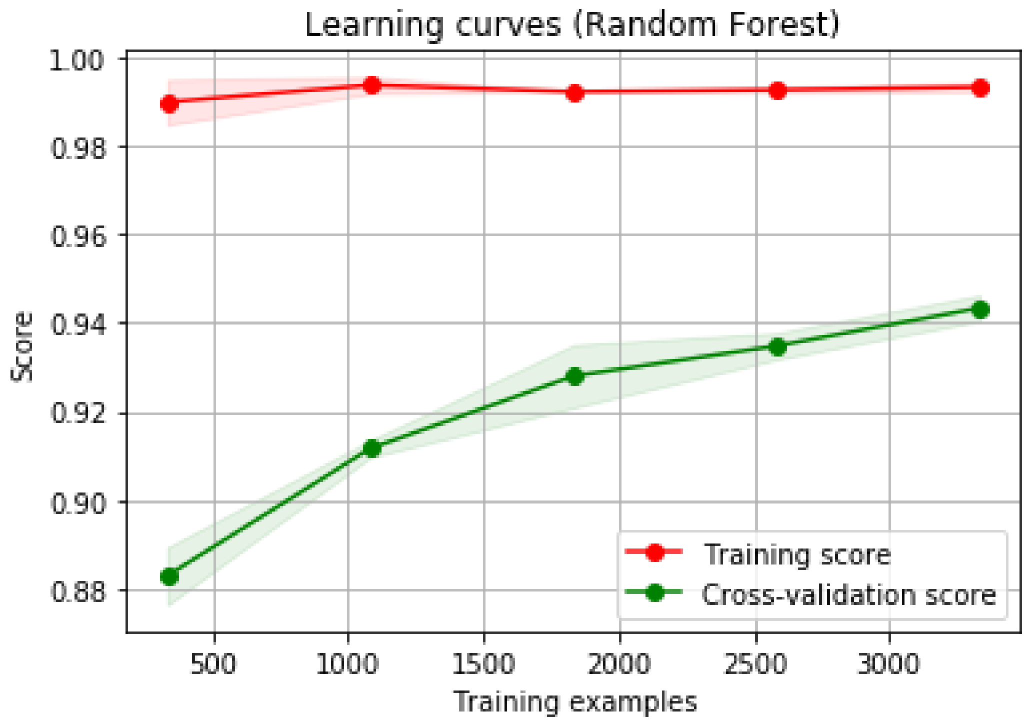

```
In [13]: KNN_CV_result = run_cv(X,y,KNeighborsClassifier) #Default: n_neighbors=5
         print "K-nearest-neighbors: " + str(accuracy(y, KNN_CV_result[0]))
         plot_learning_curve(KNN_CV_result[1], "Learning curves (KNN)", X,y)
```

```
K-nearest-neighbors: 0.8916
```

```
Out[13]: <module 'matplotlib.pyplot' from '/Users/manxilu/Applications/anaconda2/lib/python2.7/
```




```
In [14]: RF_CV_result = run_cv(X,y,RandomForestClassifier)
         print "Random forest: " + str(accuracy(y, RF_CV_result[0]))
         plot_learning_curve(RF_CV_result[1], "Learning curves (Random Forest)", X,y)

Random forest: 0.9488


Out[14]: <module 'matplotlib.pyplot' from '/Users/manxilu/Applications/anaconda2/lib/python2.7/
```



From the learning curve plots and accuracy scores for respective model: * The model is underfitted in Logistic Regression * The model is not well-behaved in KNN * The model is overfitted in Random Forest, we could continuously use this one to furthur tune the hyperprameters

4.0.3 Part 3.3: Use Grid Search to Find Optimal Parameters

Part 3.3.1: Find Optimal Parameters - RandomForest

```
In [15]: from sklearn.grid_search import GridSearchCV

         # Create the parameter grid based on the results of random search
         param_grid = {
             'bootstrap': [True],
             'n_estimators': [100, 200, 300, 1000]
         }

         Grid_RF = GridSearchCV(RandomForestClassifier(),param_grid, cv=5, verbose=1, refit=Fal
         Grid_RF.fit(X, y)
         print Grid_RF.best_params_
```

```
/Users/manxilu/Applications/anaconda2/lib/python2.7/site-packages/sklearn/cross_validation.py:4
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/manxilu/Applications/anaconda2/lib/python2.7/site-packages/sklearn/grid_search.py:42: De
  DeprecationWarning)
```

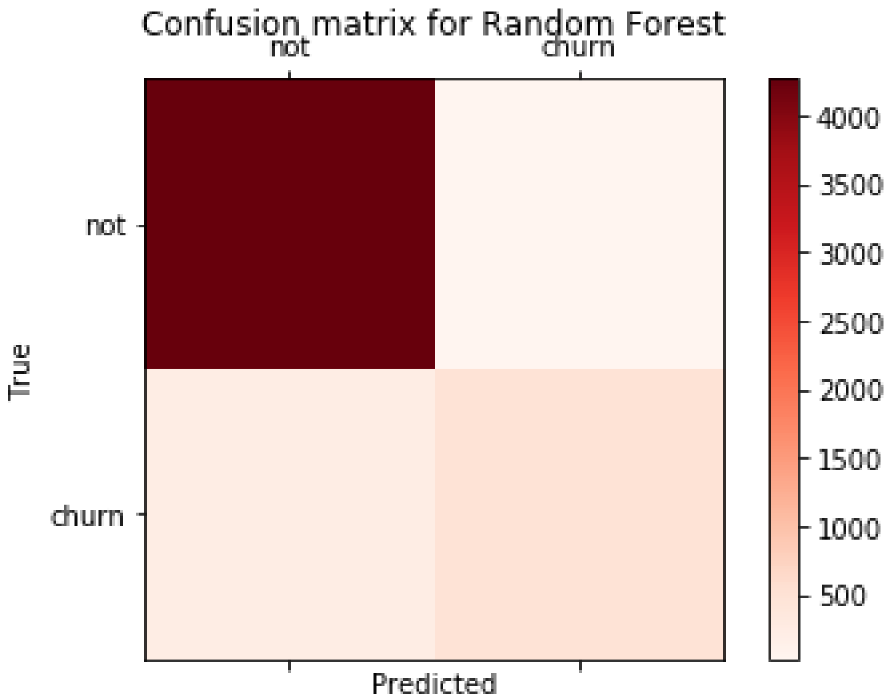
```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
{'n_estimators': 300, 'bootstrap': True}


[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:  1.1min finished
```

4.0.4 Part 3.4: Calculate Confusion Matrix (Precision, Recall, Accuracy)

```
In [16]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import precision_score
         from sklearn.metrics import recall_score

         def cal_evaluation(classifier, cm):
             tn = cm[0][0]
             fp = cm[0][1]
             fn = cm[1][0]
             tp = cm[1][1]
             accuracy  = (tp + tn) / (tp + fp + fn + tn + 0.0)
             precision = tp / (tp + fp + 0.0)
             recall = tp / (tp + fn + 0.0)
             print classifier
             print "Accuracy is " + str(accuracy)
             print "Precision is " + str(precision)
             print "Recall is " + str(recall)

         def draw_confusion_matrices(confusion_matricies,class_names=["not","churn"]):
             for cm in confusion_matricies:
                 classifier, cm = cm[0], cm[1]
                 cal_evaluation(classifier, cm)
                 fig = plt.figure()
                 ax = fig.add_subplot(111)
                 cax = ax.matshow(cm, interpolation='nearest',cmap=plt.get_cmap('Reds'))
                 plt.title('Confusion matrix for %s' % classifier)
                 fig.colorbar(cax)
                 ax.set_xticklabels([''] + class_names)
                 ax.set_yticklabels([''] + class_names)
                 plt.xlabel('Predicted')
                 plt.ylabel('True')
                 plt.show()

         cm=confusion_matrix(y, RF_CV_result[0])
         draw_confusion_matrices([("Random Forest", cm)])
         print "(tn,fp,fn,tp)= ", (cm[0][0],cm[0][1],cm[1][0],cm[1][1])
```

```
Random Forest
Accuracy is 0.9488
Precision is 0.9447731755424064
```

Recall is 0.6775106082036775



(tn,fp,fn,tp)= (4265, 28, 228, 479)

5 Part 4: Feature Selection

5.0.1 Random Forest Model - Feature Importance Discussion

In [17]:
```
importances = RF_CV_result[1].feature_importances_

# Print the feature ranking
print("Feature importance ranking by Random Forest Model:")
for k,v in sorted(zip(map(lambda x: round(x, 4), importances), churn_feat.columns), re
    print v + ": " + str(k)
```

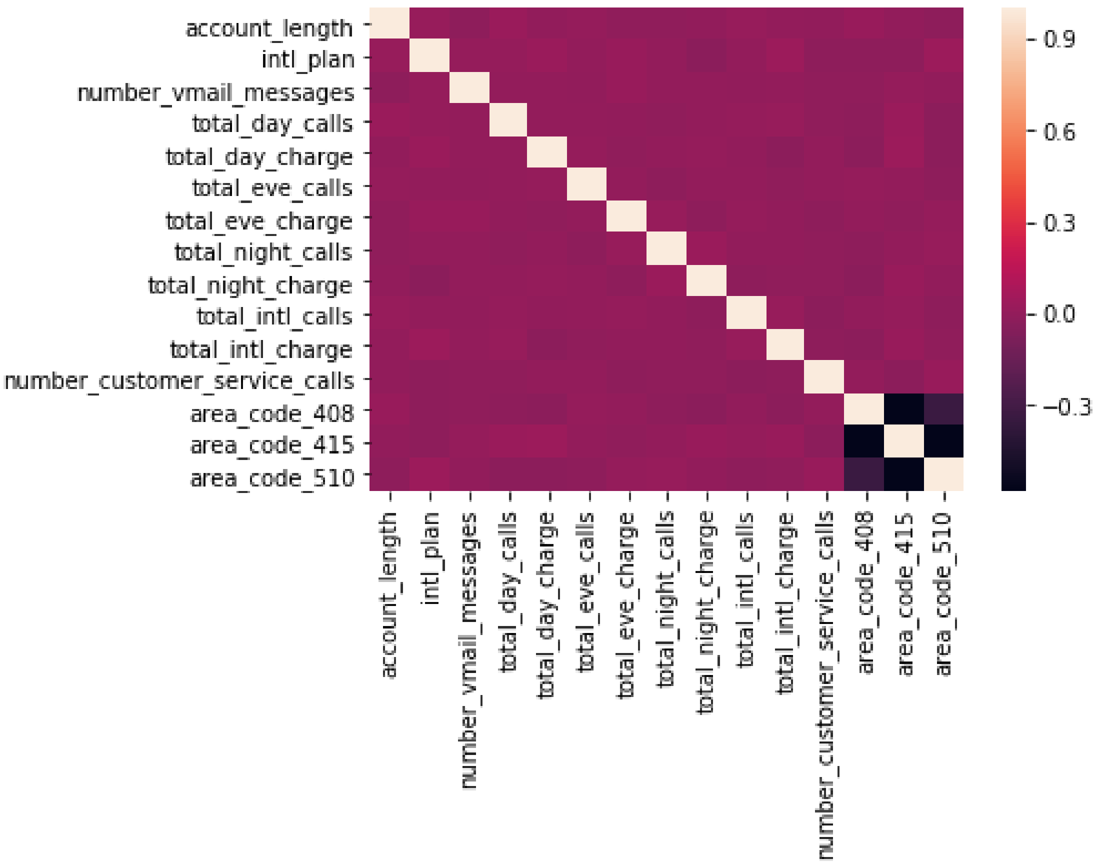
```
Feature importance ranking by Random Forest Model:
total_day_charge: 0.144
total_day_minutes: 0.1397
number_customer_service_calls: 0.1059
intl_plan: 0.0853
total_eve_charge: 0.0631
total_eve_minutes: 0.058
```

```
total_night_minutes: 0.0487
total_intl_calls: 0.0485
total_intl_charge: 0.042
total_night_charge: 0.0419
total_intl_minutes: 0.0405
account_length: 0.0356
voice_mail_plan: 0.0308
total_eve_calls: 0.0303
total_night_calls: 0.0299
total_day_calls: 0.0275
number_vmail_messages: 0.0142
area_code_415: 0.005
area_code_510: 0.0047
area_code_408: 0.0043
```

The corelated features that we are interested in previous heat plot: * voice_mail_plan & number_vmail_messages * total_day_minutes & total_day_charge * total_eve_minutes & total_eve_charge * total_night_minutes & total_night_charge * total_intl_minutes & total_intl_charge

From the above feature importance , the minutes & charge pairs have relative equaly weighting, and number_vmail_messages has slightly higher weighting than voice_mail_plan. So we decide to only drop the minutes in the pair and voice_mail_plan in feature data.

```
In [18]: select_df=churn_feat.drop(["total_day_minutes",
                                    "total_eve_minutes","total_night_minutes",
                                    "total_intl_minutes","voice_mail_plan"],axis=1)
         corr_select = select_df.corr()
         sb.heatmap(corr_select)

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1ba23dd0>
```




```
In [19]: # convert data into np-array format
         X_select = select_df.as_matrix().astype(np.float)

         # Scale the data
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         X_select = scaler.fit_transform(X_select)

         RF_select_result = run_cv(X_select,y,RandomForestClassifier)
         print "Random forest: " + str(accuracy(y, RF_select_result[0]))

         importances_select = RF_select_result[1].feature_importances_

         # Print the feature ranking
         print("Feature importance ranking by Random Forest Model:")
         for k,v in sorted(zip(map(lambda x: round(x, 4), importances), select_df.columns), rev
             print v + ": " + str(k)
```

```
Random forest: 0.9378
Feature importance ranking by Random Forest Model:
```

```
total_eve_charge: 0.144
total_day_charge: 0.1397
intl_plan: 0.0853
total_intl_calls: 0.0631
total_night_calls: 0.058
total_intl_charge: 0.0487
area_code_510: 0.0485
area_code_408: 0.0419
area_code_415: 0.0405
account_length: 0.0356
number_vmail_messages: 0.0308
total_night_charge: 0.0303
number_customer_service_calls: 0.0299
total_eve_calls: 0.0275
total_day_calls: 0.0142
```

- Total_eve_charge has principle weighting and total_day_charge also rank 2rd
  - Total charge is reasonable feature to affect user churn
  - Contact marketing group to design better plan with affordable charge
- International plan usage has high weighted for user churn
  - Conduct more advertisement on international plan service
  - Improve international telecommunication service (better communication quality, fast internet surffing etc.)
- The people will become more churned from area coded in 510 than that in 415, 408
  - Contact the customer service department from those areas especially in 510 to see if there exists terrible customer service that affect user experience
  - If the service only covered in these three areas, it could a good chance to expand service to other places
-

6 Part 5: Use Probabilities as Prediction Results

RandomForestClassifier, KNeighborsClassifier and LogisticRegression have predict_prob() function

```
In [20]: def run_prob_cv(X, y, clf_class, roc=False, **kwargs):
             kf = KFold(n_splits=5, shuffle=True)
             y_prob = np.zeros((len(y),2))
             for train_index, test_index in kf.split(X):
                 X_train, X_test = X[train_index], X[test_index]
                 y_train = y[train_index]
```

```
            clf = clf_class(**kwargs)
            clf.fit(X_train,y_train)
            # Predict probabilities, not classes
            y_prob[test_index] = clf.predict_proba(X_test)
        return y_prob
```

Result Evaluation: Use the ground truth probability to compare with our probability prediction results.

```
In [21]: from collections import defaultdict
         true_prob = defaultdict(float)

         pred_prob = run_prob_cv(X, y, RandomForestClassifier, n_estimators=200)
         pred_churn = pred_prob[:,1]
         is_churn = (y == 1)

         counts = pd.value_counts(pred_churn)
         for prob in counts.index:
             true_prob[prob] = np.mean(is_churn[pred_churn == prob])
         true_prob = pd.Series(true_prob)
```

```
In [22]: EvaResults = pd.concat([counts,true_prob], axis=1).reset_index()
         EvaResults.columns = ['pred_prob', 'count', 'true_prob']
         EvaResults
```

Out[22]:

	pred_prob	count	true_prob
0	0.000	112	0.053571
1	0.005	244	0.016393
2	0.010	284	0.031690
3	0.015	338	0.029586
4	0.020	317	0.031546
5	0.025	291	0.013746
6	0.030	266	0.026316
7	0.035	251	0.023904
8	0.040	239	0.016736
9	0.045	204	0.019608
10	0.050	144	0.027778
11	0.055	149	0.013423
12	0.060	138	0.028986
13	0.065	113	0.008850
14	0.070	100	0.010000
15	0.075	91	0.032967
16	0.080	82	0.036585
17	0.085	75	0.026667
18	0.090	73	0.041096
19	0.095	68	0.014706
20	0.100	50	0.060000
21	0.105	48	0.000000
22	0.110	49	0.000000

23	0.115	45	0.000000
24	0.120	36	0.027778
25	0.125	41	0.048780
26	0.130	37	0.027027
27	0.135	25	0.040000
28	0.140	27	0.000000
29	0.145	24	0.000000
..	...	...	...
166	0.845	8	1.000000
167	0.850	9	1.000000
168	0.855	12	1.000000
169	0.860	8	1.000000
170	0.865	7	1.000000
171	0.870	5	1.000000
172	0.875	14	1.000000
173	0.880	5	1.000000
174	0.885	8	1.000000
175	0.890	11	1.000000
176	0.895	7	1.000000
177	0.900	5	1.000000
178	0.905	7	1.000000
179	0.910	1	1.000000
180	0.915	13	1.000000
181	0.920	2	1.000000
182	0.925	7	1.000000
183	0.930	4	1.000000
184	0.935	3	1.000000
185	0.940	3	1.000000
186	0.945	4	1.000000
187	0.950	5	1.000000
188	0.955	7	1.000000
189	0.960	5	1.000000
190	0.965	4	1.000000
191	0.970	5	1.000000
192	0.975	2	1.000000
193	0.980	5	1.000000
194	0.990	2	1.000000
195	0.995	1	1.000000

[196 rows x 3 columns]