In [2]:

```
import pandas as pd
import numpy as np

from datetime import datetime
from sklearn.externals import joblib

import itertools
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression, LogisticRegression, RidgeCV
from sklearn.model_selection import train_test_split, learning_curve, cross_val_sco
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, MultinomialNB

from sklearn import svm
from sklearn.svm import SVC

from sklearn.cross_validation import KFold
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, Gradient

from sklearn.preprocessing import label_binarize, LabelEncoder
from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import make_pipeline

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,
from sklearn.metrics import confusion_matrix, precision_recall_fscore_support, class

%matplotlib inline
pd.set_option('display.max_columns', 400)
```

```
/Users/Sonal/anaconda/envs/py35/lib/python3.5/site-packages/sklearn/cr
oss_validation.py:44: DeprecationWarning: This module was deprecated i
n version 0.18 in favor of the model_selection module into which all t
he refactored classes and functions are moved. Also note that the inte
rface of the new CV iterators are different from that of this module.
This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
ic_df = pd.read_csv("ic_3mill.csv")
ic_df.head()
```

...

In [ ]:

```

```

In [3]:

```
#changing the order of the columns
ic_df = ic_df[['order_id','user_id','department_id','department','aisle_id','aisle',
ic_df.head()
```

...

In [4]:

```
#removed duplicate columns
ic_df.head()
```

...

In [5]:

```
len(ic_df['department_id'].unique())
```

...

In [6]:

```
ic_df.shape
```

...

In [7]:

```
#checking which column have null values
ic_df.isnull().any()
```

...

In [8]:

```
ic_df['days_since_prior'].isnull().sum().sum()
```

...

In [9]:

```
ic_df[pd.isnull(ic_df['days_since_prior'])]
```

...

In [10]:

```
ic_df['days_since_prior'].value_counts()
```

...

Orders per user

In [11]:

```
#orders per user
ic_df.groupby('user_id',as_index=False)['order_number'].max()
```

```
...
```

In [12]:

```
ic_df.groupby('user_id',as_index=False)['order_number'].max().describe()
```

```
...
```

In [20]:

```
#Understanding the distributions - mean, min, max, median for both the variables to
#orders_per_user_df.order_id.describe()
```

In [14]:

```
#Avg Items in Cart per Order Per User
avg_items_per_order_per_user_df = ic_df.groupby(['user_id','order_id'],as_index=Fals
avg_items_per_order_per_user_df
```

```
...
```

items_per_order_per_user

In [15]:

```
#6.36 items per order per user
items_per_order_per_user_df = avg_items_per_order_per_user_df.groupby(['user_id'],as
items_per_order_per_user_df.loc[:, 'add_to_cart_order'] = np.ceil(items_per_order_pe
items_per_order_per_user_df
```

```
...
```

In [18]:

```
items_per_order_per_user_df.add_to_cart_order.describe()
```

```
...
```

In [21]:

```
items_per_order_per_user_df[items_per_order_per_user_df['add_to_cart_order'] >10]
```

```
...
```

In [22]:

```
#Understanding the distributions - mean, min, max, median for both the variables to
#lapply( avg_items_per_order_per_user_df['order_id'].describe(), round, digits=2)
avg_items_per_order_per_user_df['order_id'].describe()
```

```
...
```

In [23]:

```
#number of reordered items per order
#(sum for all orders) / total number of orders (avg number of reordered items per o
```

In [24]:

```
#num_reordered_items_per_order_df = ic_df.groupby(['user_id','order_id'],as_index=F
reorder_sum_of_all_orders_df = ic_df.groupby(['user_id'],as_index=False)['reordered
reorder_sum_of_all_orders_df
```

```
...
```

In [25]:

```
#total orders per user
total_orders_per_user_df = ic_df.groupby(['user_id']).order_id.nunique().reset_index
#total_orders_per_user_df.to_csv("total_order_per_user.csv")
total_orders_per_user_df
```

```
...
```

In [26]:

```
#avg number of reordered items per order per user
avg_reordered_items_per_order_per_user =  np.ceil(reorder_sum_of_all_orders_df['reor

avg_reordered_items_per_order_per_user = pd.DataFrame(avg_reordered_items_per_order_
avg_reordered_items_per_order_per_user
```

```
...
```

In [27]:

```
#determining mean and max of 'days_since_prior_order' for a user

mean_days_since_prior_df = ic_df.groupby(['user_id'],as_index=False)['days_since_pri
mean_days_since_prior_df.loc[:, 'days_since_prior'] = np.ceil(mean_days_since_prior_

max_days_since_prior_df = ic_df.groupby(['user_id'],as_index=False)['days_since_pric
max_days_since_prior_df.loc[:, 'days_since_prior'] = np.ceil(max_days_since_prior_df

max_days_since_prior_df
```

```
...
```

In [28]:

```
#determining mode of 'order_dow' for a user

mode_order_dow = ic_df.groupby(['user_id']).agg(lambda x:x.value_counts().index[0])
mode_order_dow
```

```
...
```

In [31]:

```
mode_order_dow1 = mode_order_dow.reset_index()
mode_order_dow2 = pd.DataFrame(mode_order_dow1['order_dow'])

mode_order_hour_of_day = pd.DataFrame(mode_order_dow1['order_hour_of_day'])
mode_order_hour_of_day
```

```
...
```

In [32]:

```
ic_df.groupby('user_id').first().reset_index()
```

```
...
```

In [33]:

```
ic_train_orders_df = ic_df[ic_df['eval_set'] == 'train']
ic_train_orders_df
```

```
...
```

In [34]:

```
#ic_train_orders_df.groupby(['user_id'],as_index=False)['add_to_cart_order'].max()

ic_inter_df = ic_train_orders_df.groupby(['user_id'])[["order_id", "department_id",
```

In [35]:

```
ic_inter_df.shape
```

```
...
```

In [36]:

```
ic_inter_df.head()
```

```
...
```

In [37]:

```
#concat avg_orders_per_user to the main dataframe ic_inter_df

ic_inter_df1 = pd.concat([ic_inter_df, total_orders_per_user_df['order_id']], axis=
ic_inter_df1
```

```
...
```

In [38]:

```
#concat items_per_order_per_user_df to the dataframe - ic_inter_df1

ic_inter_df2 = pd.concat([ic_inter_df1, items_per_order_per_user_df['add_to_cart_or
ic_inter_df2
```

```
...
```

In [39]:

```
#concat avg_reordered_items_per_order_per_user to the dataframe - ic_inter_df2

ic_inter_df3 = pd.concat([ic_inter_df2, avg_reordered_items_per_order_per_user], ax
ic_inter_df3
```

```
...
```

In [40]:

```
#concat mean_days_since_prior_df to the dataframe - ic_inter_df3

ic_inter_df4 = pd.concat([ic_inter_df3, mean_days_since_prior_df], axis=1)
ic_inter_df4
```

```
...
```

In [41]:

```
#concat max_days_since_prior_df to the dataframe - ic_inter_df4

ic_inter_df5 = pd.concat([ic_inter_df4, max_days_since_prior_df], axis=1)
ic_inter_df5
```

```
...
```

In [42]:

```
#concat mode_order_dow2 to the dataframe - ic_inter_df5

ic_inter_df6 = pd.concat([ic_inter_df5, mode_order_dow2], axis=1)
ic_inter_df6
```

```
...
```

In [43]:

```
#concat mode_order_hour_of_day to the dataframe - ic_inter_df6

ic_df_final = pd.concat([ic_inter_df6, mode_order_hour_of_day], axis=1)
ic_df_final
```

```
...
```

In [72]:

```


ic_df_final.head()
```

```
...
```

In [64]:

```
ic_df_final.columns.values
```

```
...
```

In [66]:

```
#ic_df_final.drop(ic_df_final.columns[[18,20]], axis =1, inplace = True)
ic_df_final.columns.values
```

```
...
```

In [67]:

```
ic_df_final.shape
```

```
...
```

In [68]:

```
new_columns = ic_df_final.columns.values

new_columns[14] = 'avg_orders'
new_columns[15] = 'avg_items_per_order'
new_columns[16] = 'avg_reordered_items_per_order'
new_columns[17] = 'avg_days_since_prior'
new_columns[18] = 'max_days_since_prior'
new_columns[19] = 'mode_order_dow'
new_columns[20] = 'mode_order_hour_of_day'


ic_df_final.columns = new_columns
ic_df_final.head()
```

```
...
```

In [69]:

```
 ic_df_final.columns.values
```

...

In [75]:

```
user_id_df = ic_inter_df6.iloc[:,0]
user_id_df
```

...

In [76]:

```
ic_df_final1 = pd.concat([ic_df_final, user_id_df], axis=1)
ic_df_final1
```

...

In [77]:

```
 ic_df_final1.columns.values
```

...

In [78]:

```
#changing the order of the columns
instacart_df = ic_df_final1[['user_id','order_id', 'department_id', 'department', 'a
       'product_id', 'product_name', 'eval_set', 'order_number',
       'order_dow', 'order_hour_of_day', 'days_since_prior',
       'add_to_cart_order', 'reordered', 'avg_orders',
       'avg_items_per_order', 'avg_reordered_items_per_order',
       'avg_days_since_prior', 'max_days_since_prior', 'mode_order_dow',
       'mode_order_hour_of_day']]
instacart_df.head()
```

...

In [256]:

```
instacart_df.to_csv("instacart_snapshot.csv")
```

In [79]:

```
#Creating target variable power_user
#(orders per user > 15 and items per order per user > 10) -> count how many labels 

def f(row):
    if ((row['avg_orders'] > 10) & (row['avg_items_per_order'] > 9)):
        val = 1
    else:
        val = 0
    return val

instacart_df['power_user'] = instacart_df.apply(f, axis=1)
```

...

In [117]:

```
len(instacart_df[instacart_df['power_user'] ==1])
```

...

In [121]:

```
instacart_df['avg_orders'].hist()
```

...

Functions

In [222]:

```
def plotting_roc(fpr_val,tpr_val,roc_auc_val):
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr_val, tpr_val, 'b', label = 'AUC = %0.2f' % roc_auc_val)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
    plt.savefig('roc1.png')

def roc_for_thresholds(y,scores):
     # Compute ROC curve and ROC area for each class

    # Use roc_curve to return the TPR and FPR rates at various thresholds
     fpr, tpr, thresholds = roc_curve(y, scores, pos_label=1)

     fpr_df = pd.DataFrame(fpr)
     tpr_df = pd.DataFrame(tpr)
     threshold_df = pd.DataFrame(thresholds)
     tpr_fpr_df = pd.concat([fpr_df,tpr_df],axis =1)
     metrics_df = pd.concat([tpr_fpr_df,threshold_df],axis =1)
     #print(metrics_df)

     print('FPR:' + str(fpr))
     print('TPR:' + str(tpr))
     print('Thresholds:' + str(thresholds))
     # Plot our ROC curve!
     plt.plot(fpr, tpr)
     plt.xlabel('FPR')
     plt.ylabel('TPR')

     return(metrics_df)


def plot_response(k,knn_accuracy):
    #print(len(k))
    plt.plot(k,knn_accuracy,lw=2)

    plt.legend(['knn accuracy'])

    plt.xlabel('k')
    plt.ylabel('accuracy')
    plt.title('Accuracy response to k')
    plt.show()

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)
    test_scores_mean = np.mean(test_scores, axis=1)
    plt.grid()

    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


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
```

```
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


def logistic_regression(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    logr = LogisticRegression()
    logr.fit(ind_var_train,dep_var_train)
    y_pred = logr.predict(ind_var_test)
    y_pred_prob = logr.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]


    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    metrics_df = roc_for_thresholds(dep_var_test,y_pred_prob)
    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    # Check trained model intercept
    print("Intercept :",logr.intercept_)

    # Check trained model coefficients
    print("Coefficients :",logr.coef_)

    joblib.dump(logr, 'instcart_model.pkl', protocol=2)
    joblib.dump(logr, "instcart_model.pkl",protocol=2)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    #print(f1_score)
    print("Recall :", recall)
    print("Precision :", precision)
    #print(fpr)
    #print(tpr)
    print("ROC_AUC :", roc_auc)

    # View summary of common classification metrics
    print(classification_report(dep_var_test, y_pred))


    # Compute confusion matrix
    class_names =[1,0]
    cnf_matrix = confusion_matrix(dep_var_test, y_pred)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix using matplotlib
    plt.figure()
    plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix')

    # Plot normalized confusion matrix
    #plt.figure()
    #plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
    #                      title='Normalized confusion matrix')

    plt.show()

    #Plot confusion matrix using Seaborn
    cm = confusion_matrix(dep_var_test,y_pred)

    df_cm = pd.DataFrame(cm, index = ['True (positive)', 'True (negative)'])
    df_cm.columns = ['Predicted (positive)', 'Predicted (negative)']

    sns.heatmap(df_cm, annot=True, fmt="d")

    return accuracy, recall, precision, metrics_df


def logistic_regression_cv(ind_var,dep_var,cv):
    logr_cv = LogisticRegression()
    accuracy = cross_val_score(logr_cv, ind_var, dep_var, cv=cv, scoring='r2')

    precision = cross_val_score(logr_cv, ind_var, dep_var, cv=cv, scoring='precisio
    recall = cross_val_score(logr_cv, ind_var, dep_var, cv=cv, scoring='recall')

    print("Accuracy :", accuracy.mean())
    print("Precision :", precision.mean())
    print("Recall :", recall.mean())
    return accuracy, recall, precision

def logistic_regression_holdout(ind_var_train,dep_var_train,ind_var_test,dep_var_tes
    # Create the hyperparameter grid
    c_space = np.logspace(-5, 8, 15)
    param_grid = {'C': c_space, 'penalty': ['l1', 'l2']}

    # Instantiate the logistic regression classifier: logreg
    logr = LogisticRegression()

    # Instantiate the GridSearchCV object: logreg_cv
    logreg_cv = GridSearchCV(logr, param_grid, cv=cv)

    # Fit it to the training data
    logreg_cv.fit(ind_var_train, dep_var_train)

    # Print the optimal parameters and best score
    print("Tuned Logistic Regression Parameter: {}".format(logreg_cv.best_params_))
    print("Tuned Logistic Regression Accuracy: {}".format(logreg_cv.best_score_))


def logistic_regression_poly(ind_var_train,dep_var_train,ind_var_test,dep_var_test,
    #degree = 3
    # Generate the model type with make_pipeline
    # This tells it the first step is to generate 3rd degree polynomial features in
    # a linear regression on the resulting features
    est = make_pipeline(PolynomialFeatures(degree), LogisticRegression())
    # Fit our model to the training data
    est.fit(ind_var_train, dep_var_train)
    #est.score(X_test,y_test)
    y_pred = est.predict(ind_var_test)
    y_pred_prob = est.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)

    return accuracy, recall, precision


def gaussian_nb(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    gnb = GaussianNB()
    gnb.fit(ind_var_train,dep_var_train)
    y_pred = gnb.predict(ind_var_test)
    y_pred_prob = gnb.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)

    return accuracy, recall, precision


def support_vector_machine(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    model_svm = svm.SVC(kernel='rbf',probability=True)
    model_svm.fit(ind_var_train,dep_var_train)
    y_pred = model_svm.predict(ind_var_test)
    y_pred_prob = model_svm.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)

    return accuracy, recall, precision

def decision_tree(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    dt = DecisionTreeClassifier()
    dt.fit(ind_var_train,dep_var_train)
    y_pred = dt.predict(ind_var_test)
    y_pred_prob = dt.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)

    return accuracy, recall, precision

def random_forest(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    rf = RandomForestClassifier()
    rf.fit(ind_var_train,dep_var_train)
    y_pred = rf.predict(ind_var_test)

    y_pred_prob = rf.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)

    return accuracy, recall, precision

def gradient_boost(ind_var_train,dep_var_train,ind_var_test,dep_var_test):
    gb = GradientBoostingRegressor()
    gb.fit(ind_var_train,dep_var_train)
    y_pred = gb.predict(ind_var_test)

    y_pred_prob = gb.predict_proba(ind_var_test)
    y_pred_prob = y_pred_prob[:,1]

    accuracy = accuracy_score(dep_var_test,y_pred)
    #f1_score = f1_score(dep_var_test, y_pred,average='weighted')
    recall =  recall_score(dep_var_test, y_pred,average='weighted')
    precision = precision_score(dep_var_test, y_pred,average='weighted')

    fpr, tpr, threshold = roc_curve(dep_var_test,y_pred_prob)
    roc_auc = auc(fpr, tpr)

    plotting_roc(fpr,tpr,roc_auc)

    print("Accuracy :", accuracy)
    print("Recall :", recall)
    print("Precision :", precision)
    print("ROC_AUC :", roc_auc)
```

In [30]:

```
final_cols = ['avg_reordered_items_per_order', 'avg_days_since_prior', 'mode_order_c
```

Creating a holdout set for finally checking the model performance

In [ ]:

```
# X = X.loc[:, final_cols]
#X_holdout = X_holdout.loc[:, final_cols]
# final_model = Lasso(alpha = final_alpha)
# final_fit = final_model.fit(X, y)
```

Determining feature importance

1. Checking correlations

In [83]:

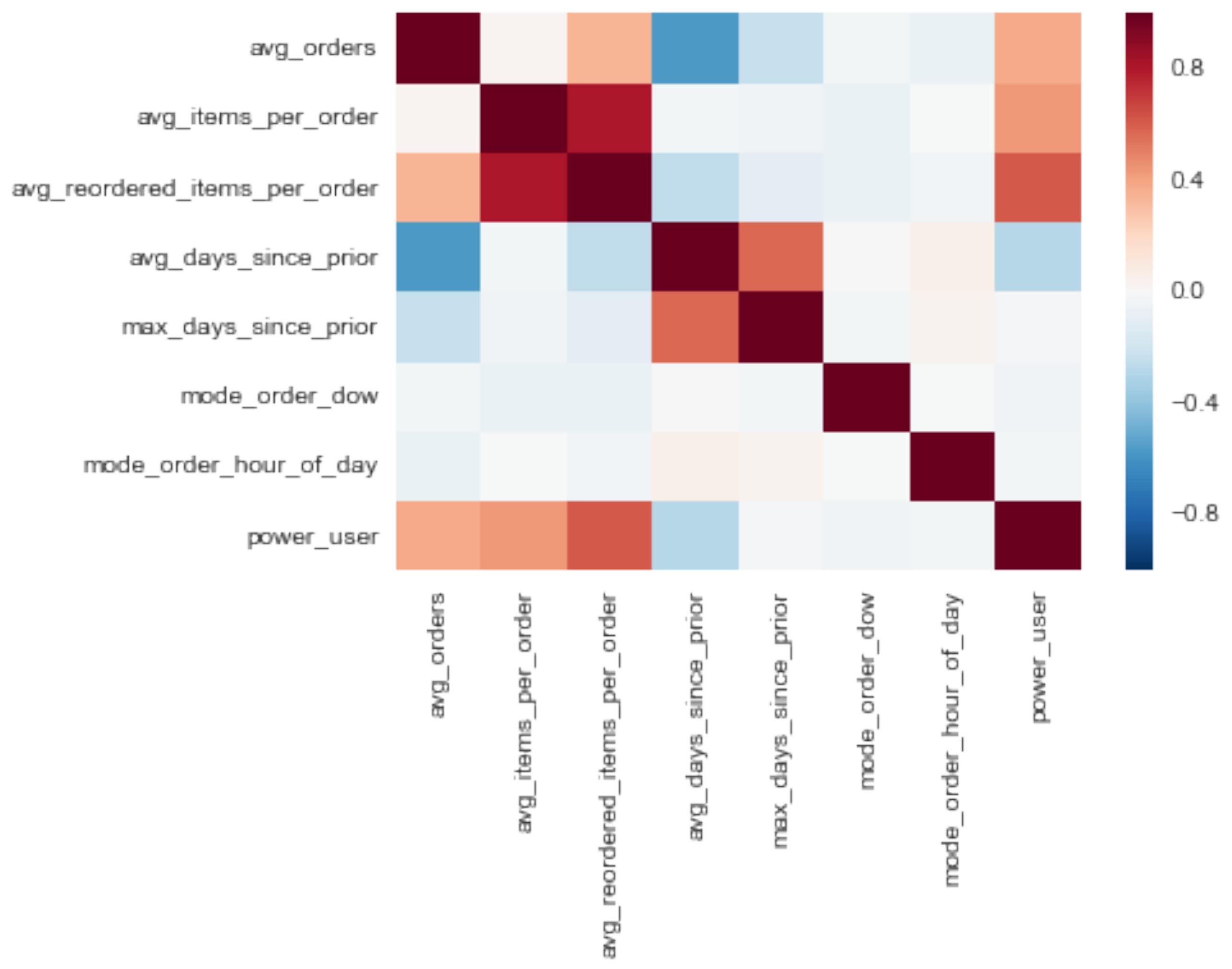
```
instacart_df.corr()
```

```
...
```

```
In [213]:
```

```
ic_corr = instacart_df.iloc[:,15:].corr()
sns.heatmap(ic_corr,
            xticklabels=ic_corr.columns.values,
            yticklabels=ic_corr.columns.values)
```

```
Out[213]:

<matplotlib.axes._subplots.AxesSubplot at 0x12b6ba160>
```

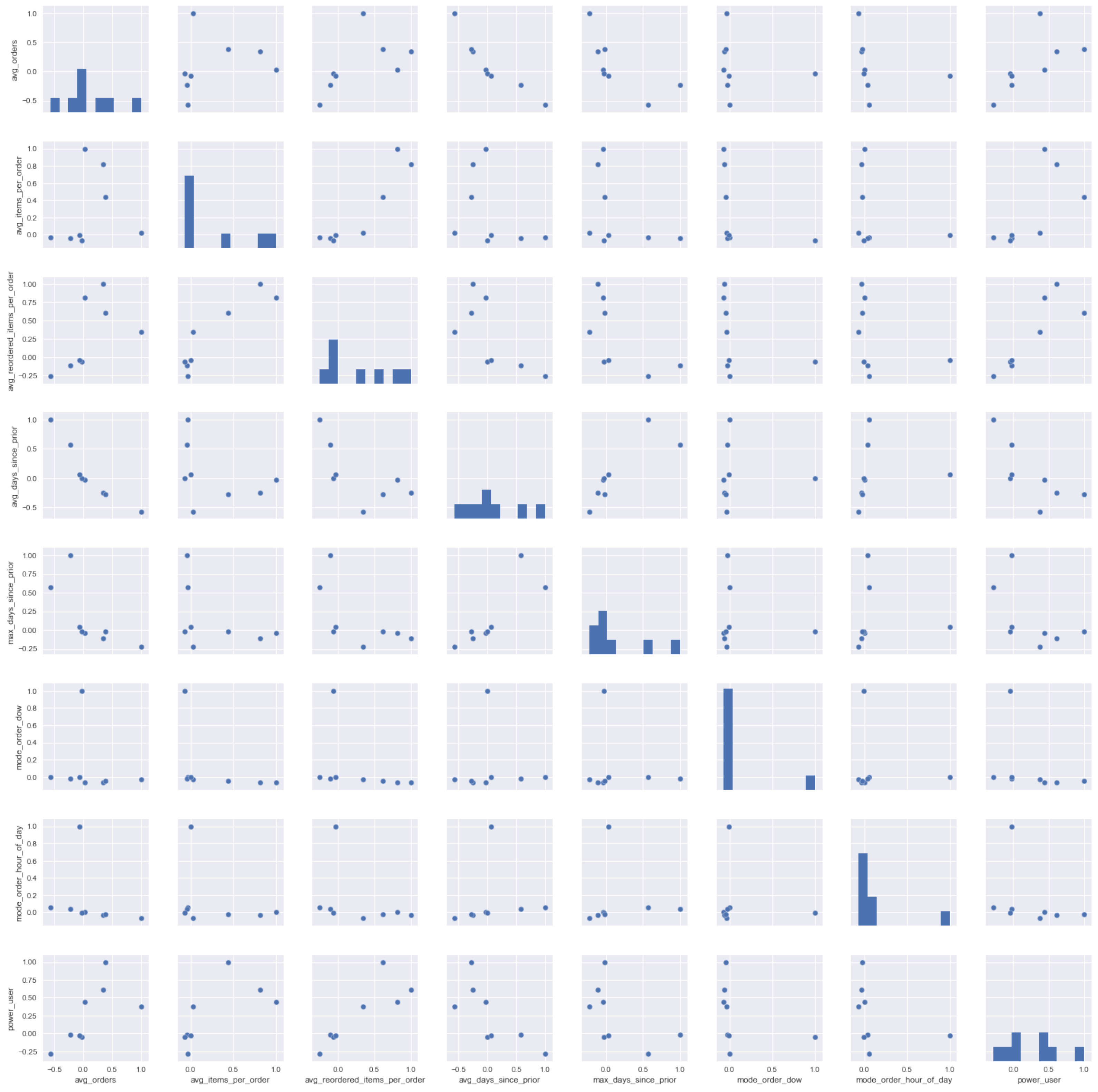


In [214]:

```
sns.pairplot(ic_corr)
```

Out[214]:

```
<seaborn.axisgrid.PairGrid at 0x12afb2e48>
```



2. Using Random Forest

In [ ]:

In [84]:

```
X = instacart_df.ix[:,['avg_reordered_items_per_order', 'avg_days_since_prior', 'moc

#X = ic_inter_df5.iloc[:,9:20]
y = instacart_df.iloc[:,-1]

X.head()
```

Out[84]:

	avg_reordered_items_per_order	avg_days_since_prior	mode_order_dow	mode_order_
0	5.0	20.0	4	8
1	7.0	19.0	1	11
2	4.0	13.0	0	18
3	7.0	14.0	0	18
4	5.0	23.0	1	0

In [85]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
rf = RandomForestRegressor(n_estimators=200)
rf.fit(X_train, y_train)
rf.score(X_test, y_test)

imp = rf.feature_importances_

imp = pd.DataFrame(np.array(imp).T, columns = ['imp'], index = X.columns)
imp.sort_values('imp', ascending = False, inplace = True)
#imp.to_csv("important_features.csv")
print(imp)
```

```
                                    imp
avg_reordered_items_per_order  0.518378
avg_days_since_prior           0.206350
mode_order_hour_of_day         0.170200
mode_order_dow                 0.105072
```

3. Using Multinomial Naive Bayes

In [86]:

```
# Create the model
mnb = MultinomialNB()

# Fit the model to the training data
mnb.fit(X_train, y_train)
# Score the model against the test data
mnb.score(X_test, y_test)

#mnb.feature_log_prob_
```

Out[86]:

```
0.83161447590474402
```

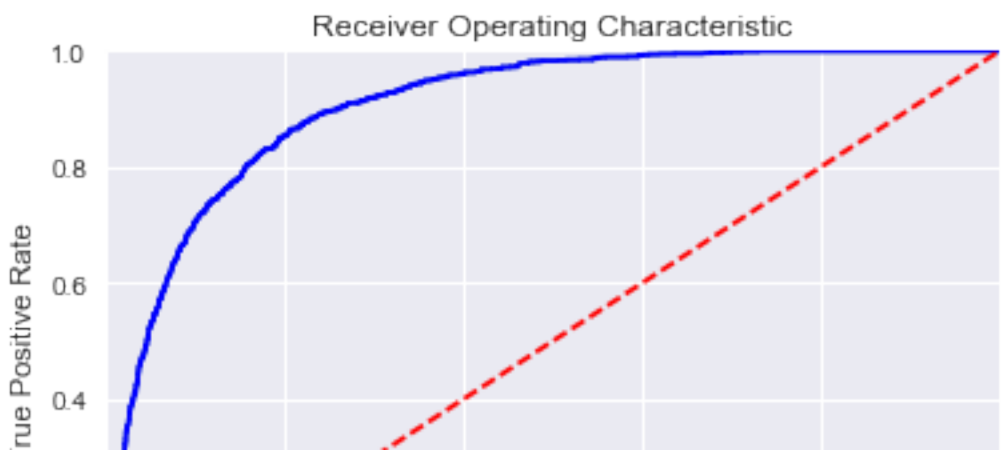
Modeling

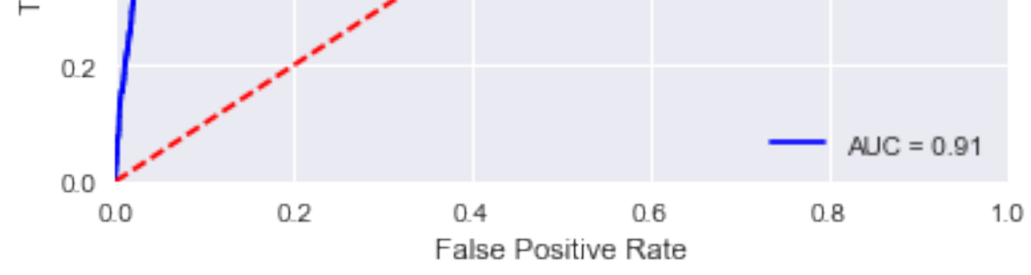
Logistic Regression with variables 'avg_reordered_items_per_order', 'avg_days_since_prior', 'mode_order_dow', 'mode_order_hour_of_day'

In [223]:

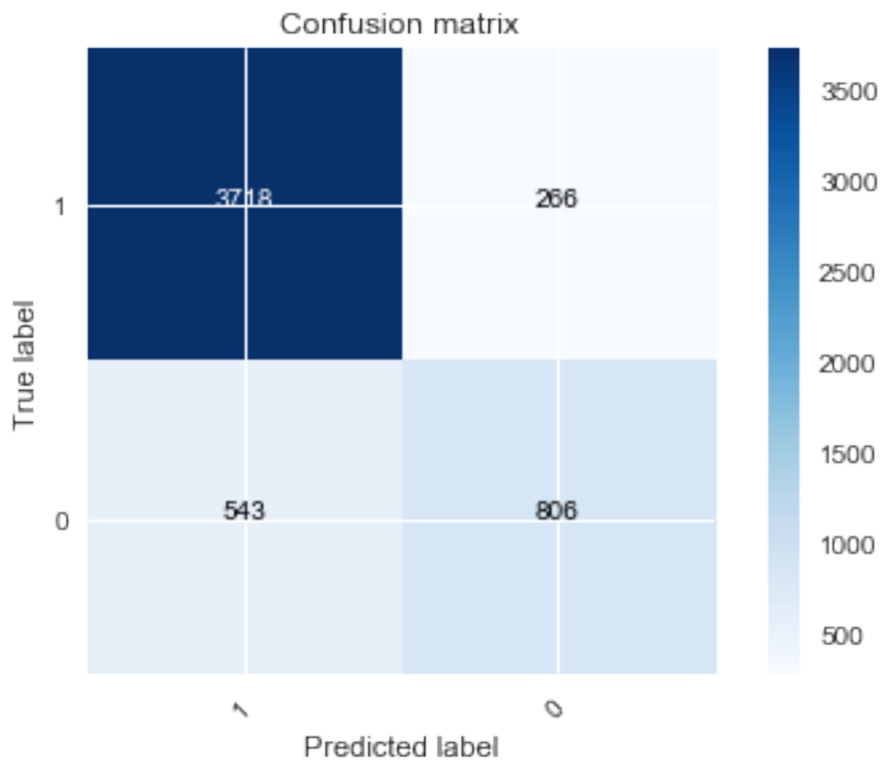
```
X = instacart_df.ix[:,['avg_reordered_items_per_order', 'avg_days_since_prior', 'moc

y = instacart_df.iloc[:,-1]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat

accuracy, recall, precision, metrics_df = logistic_regression(X_train,y_train,X_test
```

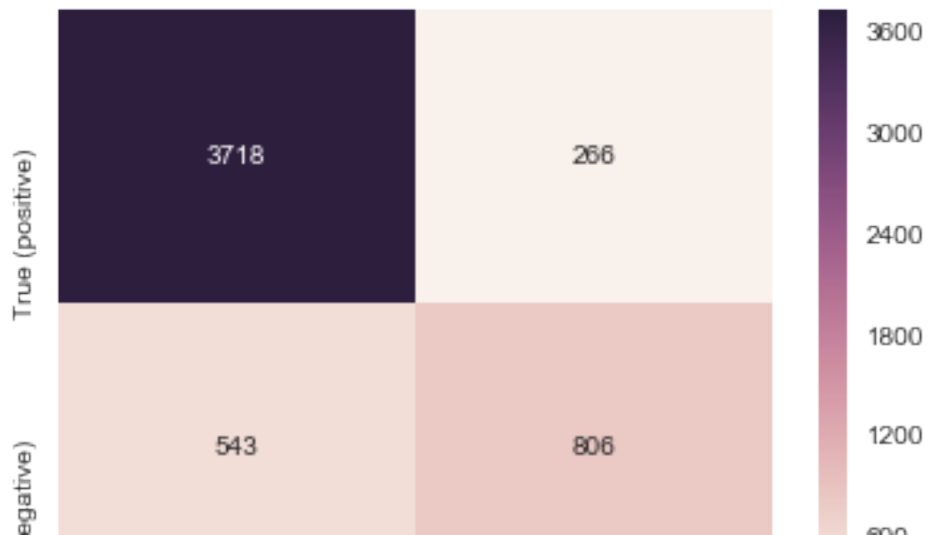
```
FPR:[   0.00e+00   0.00e+00   2.51e-04 ...,   9.96e-01   9.96e-01   1.0
0e+00]
TPR:[   7.41e-04   8.15e-03   8.15e-03 ...,   1.00e+00   1.00e+00   1.0
0e+00]
Thresholds:[ 1.    1.    1.   ...,  0.01  0.01  0.  ]
Intercept : [-2.83]
Coefficients : [[ 0.52 -0.09 -0.02  0.  ]]
```





```
Accuracy : 0.848303018939
Recall : 0.848303018939
Precision : 0.842033799225
ROC_AUC : 0.905314270425
             precision    recall  f1-score   support

          0       0.87      0.93      0.90      3984
          1       0.75      0.60      0.67      1349

avg / total       0.84      0.85      0.84      5333

Confusion matrix, without normalization
[[3718  266]
 [ 543  806]]


<matplotlib.figure.Figure at 0x12da0c860>
```





The model produced a negative intercept value and a weight of 0.52 on avg_reordered_items_per_order, -0.09 on avg_days_since_prior, -0.02 on mode_order_dow and 0. on mode_order_hour_of_day

Logistic Regression with variables 'avg_reordered_items_per_order', 'avg_days_since_prior', 'mode_order_dow', 'mode_order_hour_of_day' and k fold cross validation

In [88]:

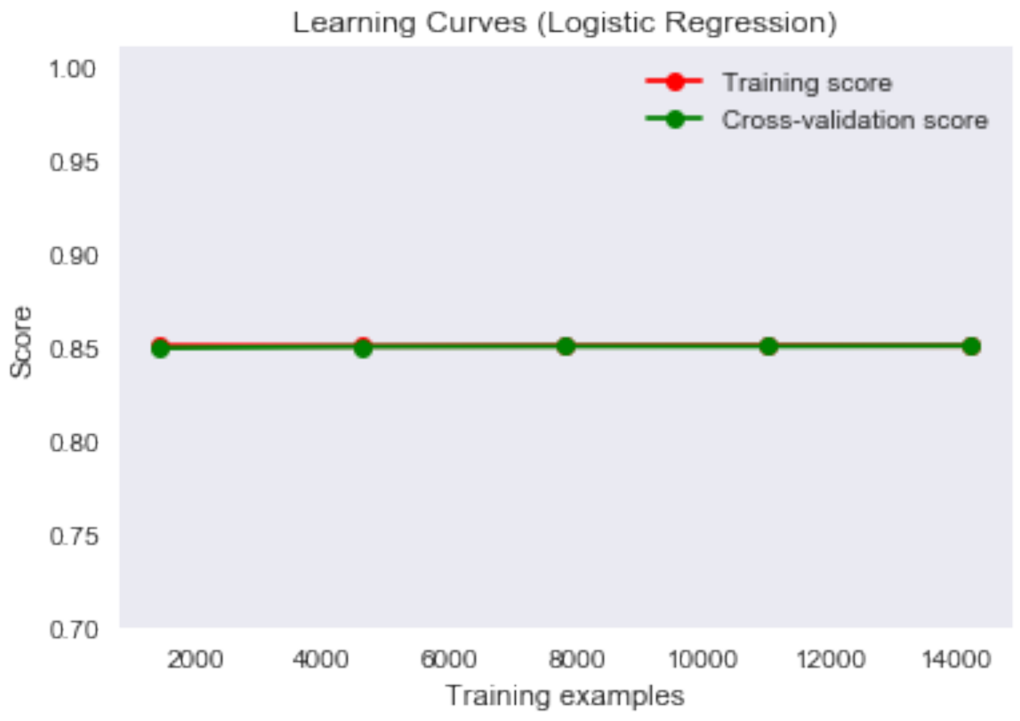
```
accuracy, recall, precision = logistic_regression_cv(X, y, 15)
```

```
Accuracy : 0.2159576962
Precision : 0.771091481707
Recall : 0.592656219264
```

Learning curve for logistic regression

In [109]:

```
title = "Learning Curves (Logistic Regression)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=4444)

estimator = LogisticRegression()
plot_learning_curve(estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)

plt.show()
```



K Nearest Neighbors

In [97]:

```
#Try it with a lot of different k values (number of neighbors), from 1 to 20,
#and on the test set calculate the accuracy (number of correct predictions / number

accuracy = []
accuracy_index = []

for k in range(1,21):
    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train,y_train)
    y_pred = knn.predict(X_test)
    accuracy.append(accuracy_score(y_test,y_pred))
    accuracy_index.append(k)

print(accuracy)
print(accuracy_index)
max_accuracy = max(accuracy)
print(max_accuracy)
max_acc_k = accuracy.index(max_accuracy)+1
print(max_acc_k)
```

```
[0.8151134445902869, 0.82542658916182265, 0.8385524095255954, 0.843240
20251265708, 0.84830301893868365, 0.85055315957247324, 0.8529908119257
4533, 0.85467841740108752, 0.85449090568160513, 0.85767860491280701, 0
.85617851115694732, 0.8589911869491843, 0.85767860491280701, 0.8599287
4554659671, 0.85617851115694732, 0.8606787924245265, 0.859178698668666
81, 0.8588036752297018, 0.8616163510219389, 0.860866304144009]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
]
0.861616351022
19
```

In [284]:

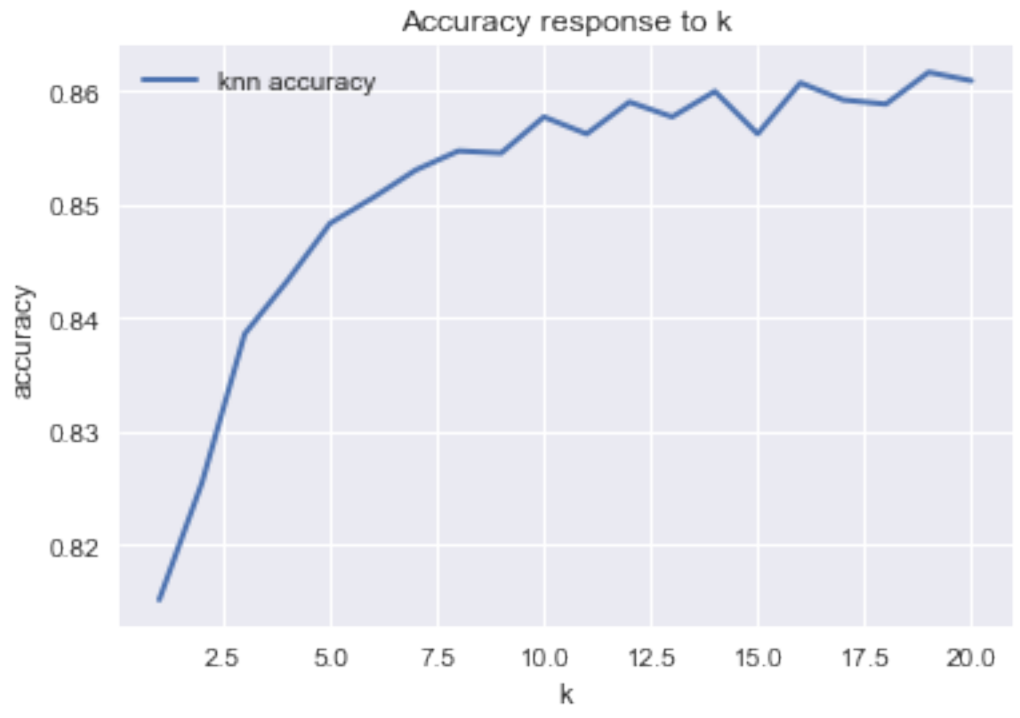
```
k=19
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_stat
knn_19 = KNeighborsClassifier(n_neighbors=k)
knn_19.fit(X_train,y_train)
y_pred = knn_19.predict(X_test)

# Compute confusion matrix
    #class_names =[1,0]
cnf_matrix = confusion_matrix(y_test, y_pred)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix using matplotlib
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=[1,0],
                      title='Confusion matrix')

# Plot normalized confusion matrix
#plt.figure()
#plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
#                     title='Normalized confusion matrix')
plt.savefig('knn_conf_mat.png')
plt.show()
#plt.savefig('knn_conf_mat.jpg')


accuracy = accuracy_score(y_test, y_pred)
#f1_score = f1_score(dep_var_test, y_pred,average='weighted')
recall =  recall_score(y_test, y_pred,average='weighted')
precision = precision_score(y_test, y_pred,average='weighted')

fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)


print("Accuracy :", accuracy)
print("Recall :", recall)
print("Precision :", precision)
print("ROC_AUC :", roc_auc)
```

...

KNN - Accuracy as a function of k

In [98]:

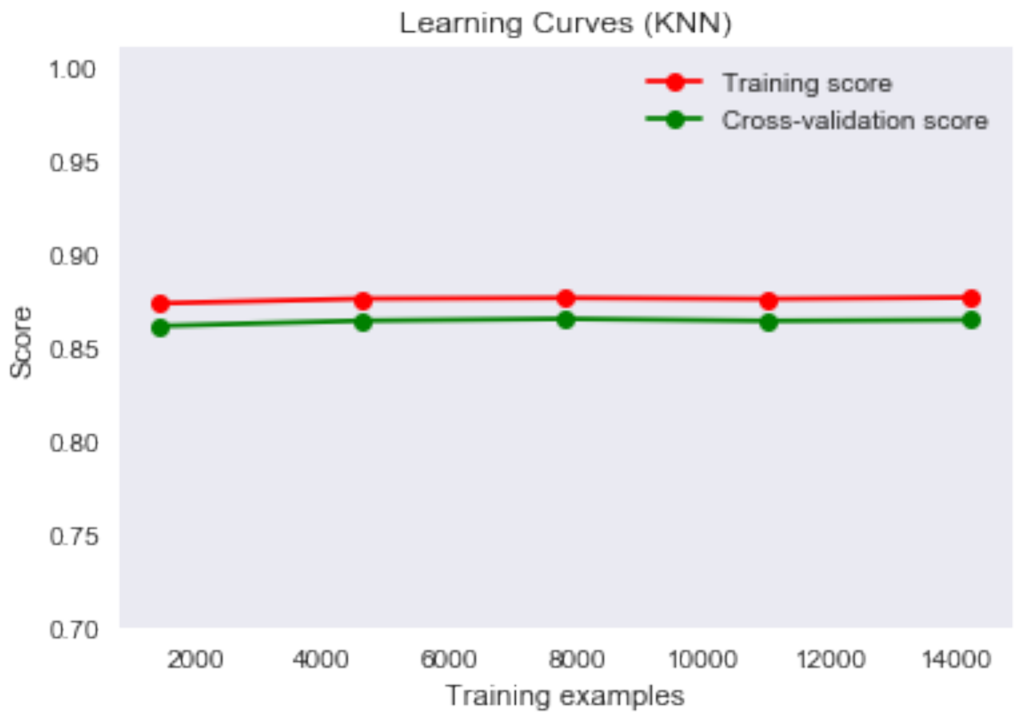
```
plot_response(accuracy_index,accuracy)
```



Learning Curve for KNN where k is the one with maximum accuracy

In [111]:

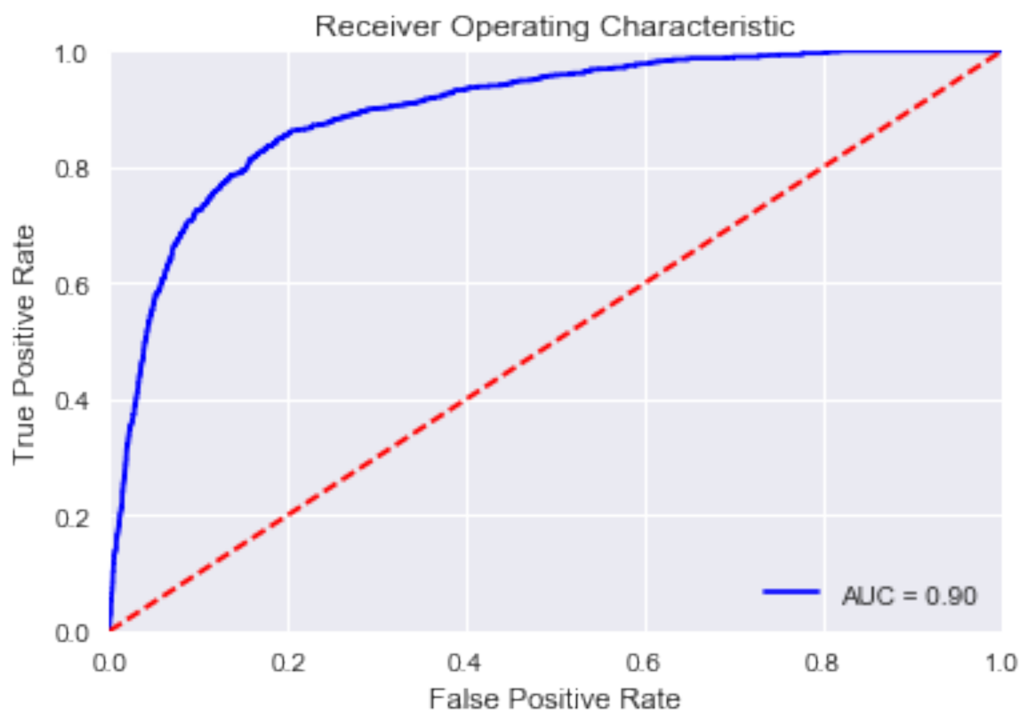
```
title = "Learning Curves (KNN)"
# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=1000)

estimator = KNeighborsClassifier(n_neighbors=19)
plot_learning_curve(estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)

plt.show()
```



Gaussian Naive Bayes

In [90]:

```
accuracy, recall, precision = gaussian_nb(X_train,y_train,X_test,y_test)
```

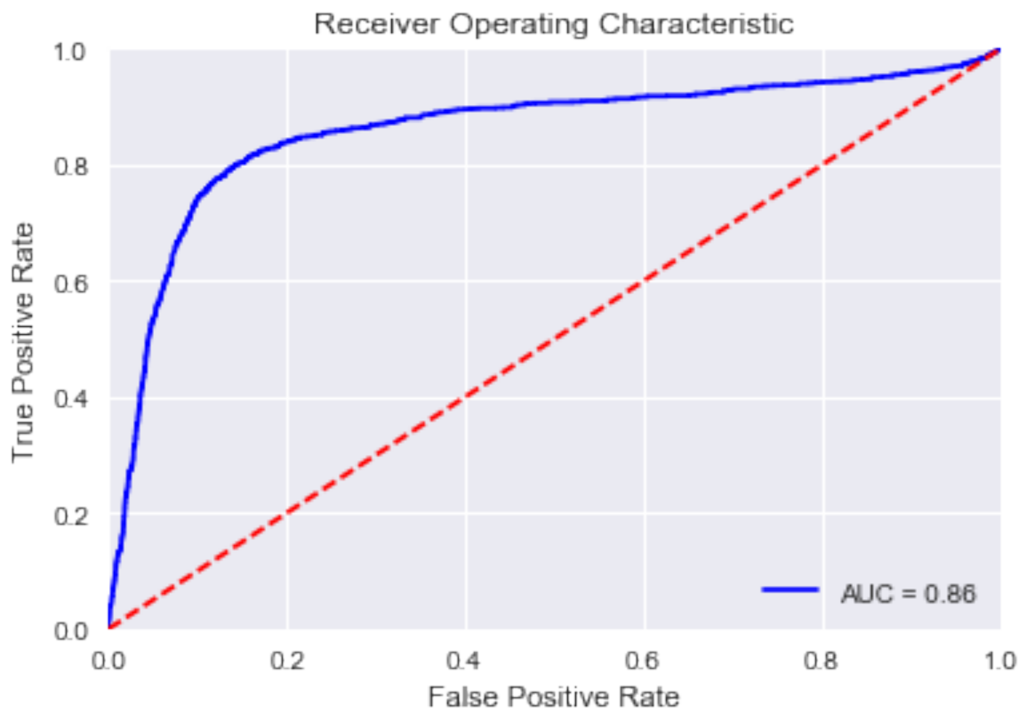


```
Accuracy : 0.854490905682
Recall : 0.854490905682
Precision : 0.849164185812
ROC_AUC : 0.897884439909
```

Support Vector Machine

In [91]:

```
accuracy, recall, precision = support_vector_machine(X_train,y_train,X_test,y_test)
```

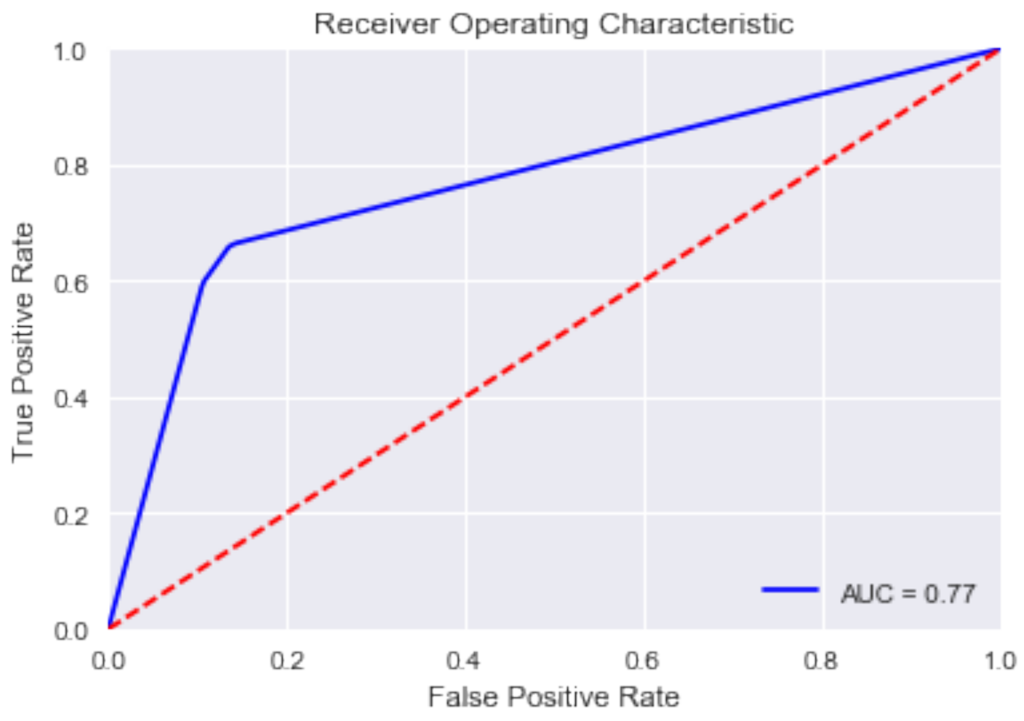


```
Accuracy : 0.857678604913
Recall : 0.857678604913
Precision : 0.854305531346
ROC_AUC : 0.858608916764
```

Decision Tree

In [92]:

```
accuracy, recall, precision = decision_tree(X_train,y_train,X_test,y_test)
```

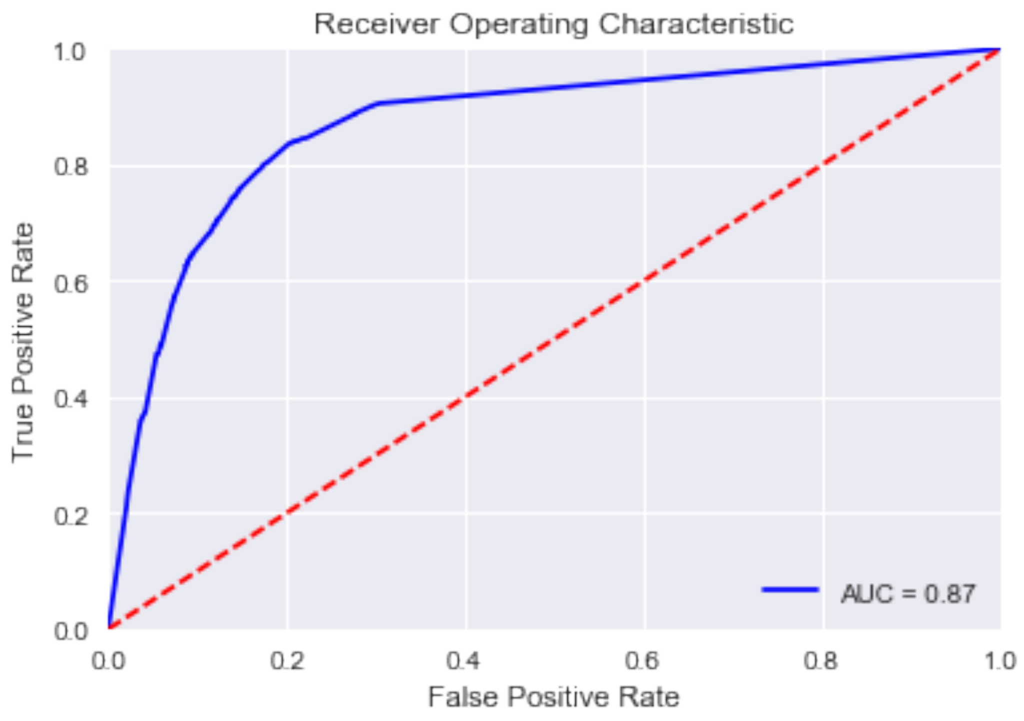


```
Accuracy : 0.818301143821
Recall : 0.818301143821
Precision : 0.813761851879
ROC_AUC : 0.768207559668
```

Random Forest

In [93]:

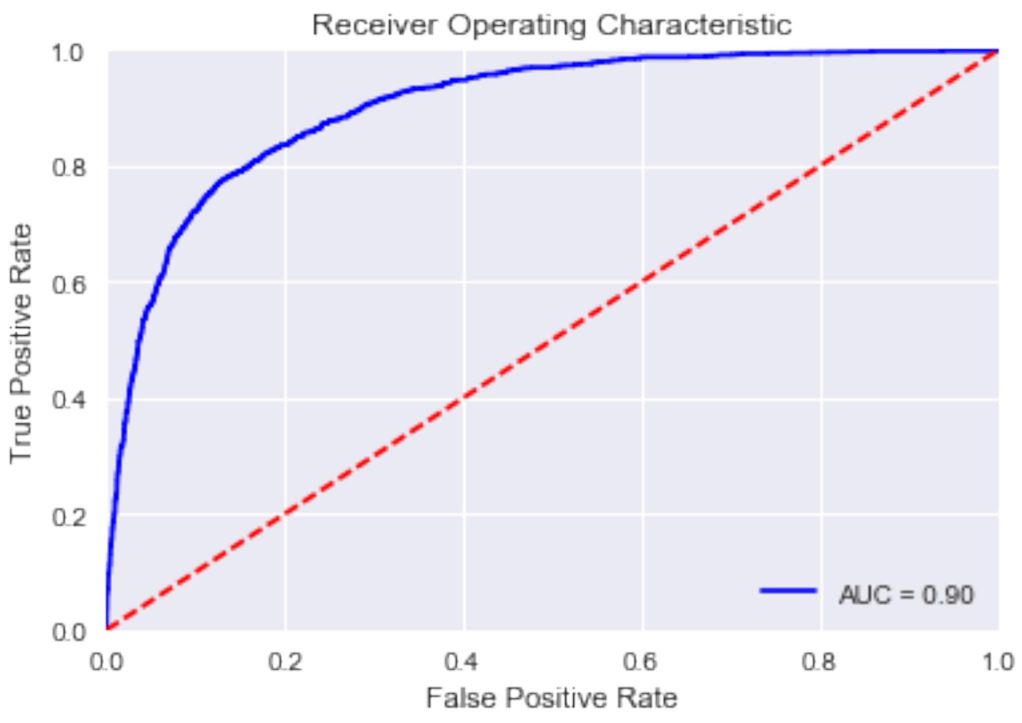
```
accuracy, recall, precision = random_forest(X_train,y_train,X_test,y_test)
```



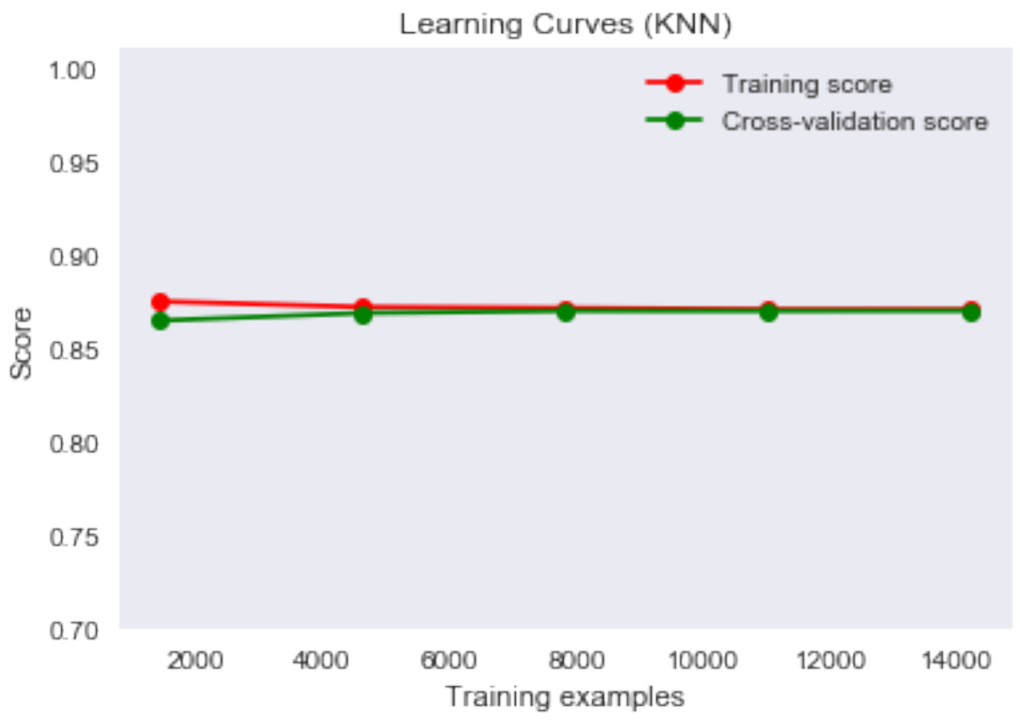
```
Accuracy : 0.840052503281
Recall : 0.840052503281
Precision : 0.83649864831
ROC_AUC : 0.869188391818
```

In [217]:

```
accuracy, recall, precision = logistic_regression_poly(X_train,y_train,X_test,y_test
plot_learning_curve(est, title, X, y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)

plt.show()
```

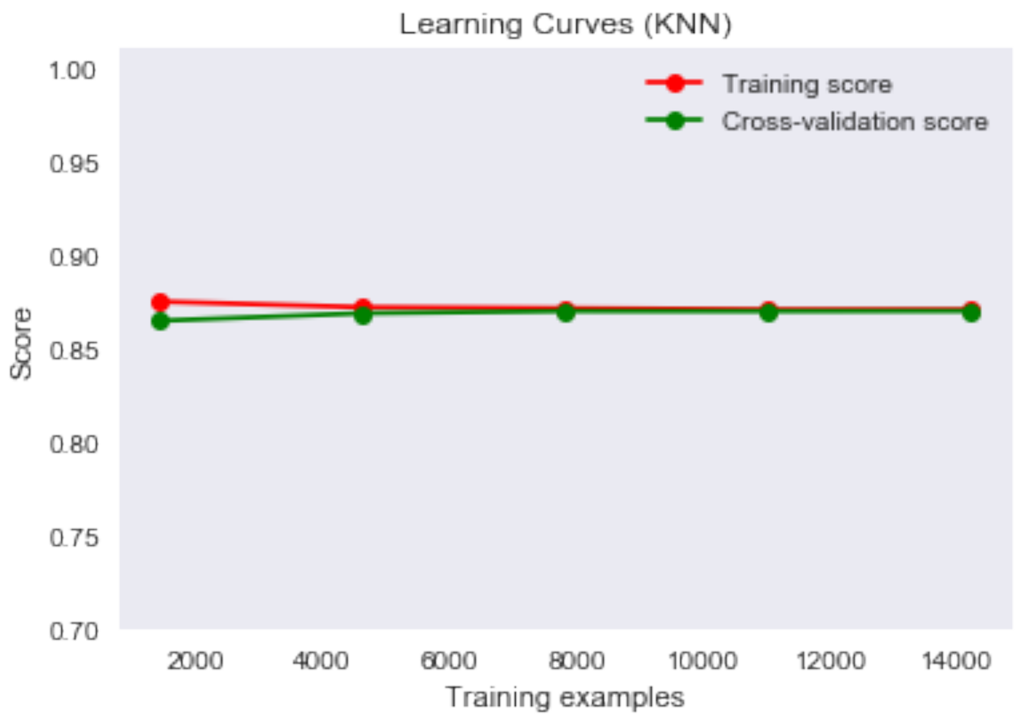


```
Accuracy : 0.858241140071
Recall : 0.858241140071
Precision : 0.855683687216
ROC_AUC : 0.901685224962
```



In [205]:

```
degree = 3
# Generate the model type with make_pipeline
# This tells it the first step is to generate 3rd degree polynomial features in the
# a linear regression on the resulting features
est = make_pipeline(PolynomialFeatures(degree), LogisticRegression())
# Fit our model to the training data
est.fit(X_train, y_train)
est.score(X_test,y_test)

plot_learning_curve(est, title, X, y, ylim=(0.7, 1.01), cv=cv, n_jobs=4)

plt.show()
```



Hold-out set in practice: Classification

Evaluating a model with tuned hyperparameters on a hold-out set.

In addition to C, logistic regression has a 'penalty' hyperparameter which specifies whether to use 'l1' or 'l2' regularization. Your job in this exercise is to create a hold-out set, tune the 'C' and 'penalty' hyperparameters of a logistic regression classifier using GridSearchCV on the training set, and then evaluate its performance against the hold-out set.

In [94]:

```
cv = 5
logistic_regression_holdout(X_train,y_train,X_test,y_test,cv)
```

```
...
```

Comparison of models using cross validation

In [ ]:

```
import sys
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.dummy import DummyClassifier

bestKValue = 19
models = {}
models = {'logres': LogisticRegression(), # Takes about 1 second elapsed time
          'knn with K=%d' % bestKValue : KNeighborsClassifier(n_neighbors=bestKValue
          'gaussianNB': GaussianNB(),
          'random forest': RandomForestClassifier(),
          'decision tree': DecisionTreeClassifier(),
          'svm': svm.SVC(kernel='rbf',probability=True),
          'baseline' : DummyClassifier(strategy='stratified')
#          'randomforest with nrEst=%d and maxFeat=%d' % (bestNrEst,bestMaxFeat): Ra
          }

scorerType = 'roc_auc'

nrCrossValidationFolds=10
# We MUST shuffle, because the data seem to be somehow ordered
cvGenerator = KFold(len(X), n_folds=nrCrossValidationFolds, shuffle=True)

fig = plt.figure(1,(9,6))
plt.title("%d-fold cross-validation %s scores for various model types" % (nrCrossVal
plt.xlabel("Fold #")
plt.ylabel(scorerType)
plt.grid()

for modelName, model in models.items():
#     print >> sys.stderr, "Building %s model ..." % modelName,
#     print >> sys.stderr, "applying it ...",
    scores = cross_val_score(model, X, y=y, scoring=scorerType, cv=cvGenerator, n_jo
#     print >> sys.stderr, "done"
    plt.plot(range(1,nrCrossValidationFolds+1), scores, 'o-', label="%s (%2.2f%% +/-

plt.legend(loc='best',fontsize = 'large')
plt.savefig('model_comp_kfolds.png')
plt.show()
```

...

ROC curves - Balancing true positives and false positives

In [3]:

```
import mpld3
mpld3.enable_notebook()
```

```
import re
from math import log
import json
import copy

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_stat

finalPredictions = {}

for modelName, model in models.items():
    fittedModel = model.fit(X_train, y_train)

    if hasattr(model, "predict_proba"):
        #print >> sys.stderr, "Predicting probabilities for %s model ..." % modelNar
        finalPredictions[modelName] = {'train': model.predict_proba(X_train),
                                       'test': model.predict_proba(X_test)}
        #print >> sys.stderr, "done"

def plotROCCurve(figSize=7):
    # Define some CSS to control our custom labels
    css = """
table {
  border-collapse: collapse;
}
th {
  color: #ffffff;
  background-color: #000000;
}
td {
  padding: 2px;
  background-color: #cccccc;
}
table, th, td {
  font-family:Arial, Helvetica, sans-serif;
  border: 1px solid black;
  text-align: right;
}
"""

    jsonROCData = {}
    rocCurveFigure, ax = plt.subplots(figsize=(figSize,figSize))
    ax.grid(True, alpha=0.3)
    for modelName, probs in finalPredictions.items():
        modelNameShort = re.split("\s+", modelName)[0]
        y_probs = [x[1] for x in finalPredictions[modelName]['test']]
        fpr, tpr, thresholds = roc_curve(y_true=y_test,
                                         y_score=y_probs, pos_label=1)

        roc_auc = roc_auc_score(y_true=y_test, y_score=y_probs)
        jsonROCData[modelNameShort] = {}
        jsonROCData[modelNameShort]['fpr'] = [x for x in fpr]
        jsonROCData[modelNameShort]['tpr'] = [ x for x in tpr]
        jsonROCData[modelNameShort]['thresholds'] = [np.asscalar(np.float32(x)) for
        jsonROCData[modelNameShort]['roc_auc'] = roc_auc
        points = plt.plot(fpr, tpr, 'x-', label="%s (AUC = %1.2f%%)" % (modelName, r
        labels = ["<table><th colspan='2'>%s</th><tr><td>FPR</td><td>%0.1f%%</td></t
        mpld3.plugins.connect(rocCurveFigure, mpld3.plugins.PointHTMLTooltip(points[
    ax.set_title("ROC curve for prediction of prime user", y=1.06, fontsize=14 + log
    ax.set_xlabel("False Positive Rate (FP/FP+TN)", labelpad=15 + log(figSize), font
    ax.set_ylabel("True Positive Rate (TP/TP+FN)", labelpad=15 + log(figSize), fonts
    plt.legend(loc="best",fontsize = 'xx-large')
    plt.show()
    mpld3.save_html(rocCurveFigure,'ROC_comp')
    #plt.savefig('ROC_comp')
    rocCurveFigure.savefig('ROC_comp2')
    return jsonROCData

# Export data to JSON file for visualization in D3.js or similar
jsonROCData = plotROCCurve(figSize=9)
#print(jsonROCData)
#with open('d3/ROCCurve.json', 'w') as outfile:
with open('ROCCurve.json', 'w') as outfile:
    json.dump(jsonROCData, outfile)

#from IPython.html.widgets import interact, fixed
from ipywidgets import interact, interactive, fixed

interact(plotROCCurve, figSize=(5,10))
```

...