In [1]:
```
#####################################################################################
#  Purpose: Identify a claim can be fast-tracked using Machine learning.
#  Created by: Suriya Mohan
#  Created on: 12-Nov-2016
#####################################################################################

# Import python - scikit-learn machine learning packages.
import numpy as np;
import pandas as pd;
from pandas import Series,DataFrame;
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Imputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import learning_curve
from sklearn.model_selection import validation_curve
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline
%matplotlib inline
```

In [2]:
```
# Load the data in CSV file into panda dataframe.
df_claims = pd.read_csv('C:\Users\gbu4moh\Desktop\ML - FAST TRACK\owning_adjuster
```

In [3]:
```
def claim_labelEncode(df,column_name):
    '''
    Purpose: Function used to convert the nominal features (fields with string va
    Input: Dataframe and column name to encode
    Output: Modify the passed dataframe by adding encoded field to end and drop t
    '''
    df[column_name].fillna(value=df[column_name].value_counts().idxmax(), inplace

    le = LabelEncoder()
    encoder = le.fit_transform(df[column_name].values)
    df.insert(0,'code' + column_name,encoder.astype(int))
    df.drop(column_name,axis=1,inplace=True)
```

In [4]:
```
def claim_oneHotEncode(df,column_array):
    '''
    Purpose: Function used to hot encode fields - example field having 3 distinct
    Input: Dataframe and column array
    Output: Returns the data frame with encoded values.
    '''
    ohe = OneHotEncoder(categorical_features=column_array,sparse=False)
    df_ohe = ohe.fit_transform(df)
    return df_ohe
```

In [5]:
```
def claim_standardScaler(df):
    '''
    Purpose: Normalize the dataframe so the features are on the same scale.
    Input: Dataframe
    Output: Returns the data frame with standarized values.
    '''
    sc = StandardScaler()
    df_std = sc.fit_transform(df)
    return df_std
```

In [6]:
```
def claim_Imputer(df):
    '''
    Purpose: Normalize the dataframe so the features are on the same scale.
    Input: Dataframe
    Output: Returns the data frame with standarized values.
    '''
    imr = Imputer(missing_values=np.NaN, strategy='most_frequent',axis=0)
    imr = imr.fit_transform(df)
    return imr
```

In [7]:
```
# Display all the fields in the input dataframe.
df_claims.columns
```

Out[7]:
```
Index([u'CLAIM_NUM', u'CASUALTY_UNIT_IND', u'FATALITY_IND',
       u'HIGH_VALUE_CLAIM_IND', u'ACCIDENT_FAULT_IND', u'LEGAL_ENTITY_NM',
       u'WEATHER_CD_DESC', u'CLAIM_JURISDICTION_ST_CD_DESC',
       u'LOSS_TYPE_CD_DESC', u'CLAIM_COMPLEXITY_CD_DESC',
       u'CLAIM_FILE_TYPE_DESC', u'CLAIM_TIER_CD_DESC', u'FAULT_RATING_CD_DESC',
       u'CATASTROPHE_IND', u'LOSS_SUB_TYPE_DESC', u'LIABILITY_CD_DESC',
       u'LIABILITY_PERCENTAGE', u'INCIDENT_ONLY_IND', u'GLASS_ONLY_IND',
       u'NON_CHARGEABLE_CLAIM_IND', u'TOTAL_LOSS_IND', u'PRODUCT_TYPE_DESC',
       u'ATTORNEY_ON_CLAIM_IND', u'VEHICLE_TOW_STORAGE_IND',
       u'COVERAGE_TYPE_CD_DESC', u'TOW_TYPE_CD_DESC', u'INCIDENT_TYPE_CD_DESC',
       u'COVERAGE_SUB_TYPE_CD_DESC', u'CLAIM_TYPE_DESC', u'COVERAGE_NM',
       u'PROPERTY_DESC', u'DAMAGE_DESC', u'VEH_COUNT',
       u'CLAIM_ADJUSTER_GROUP_TYPE'],
      dtype='object')
```

In [1]:
```
# Check value and count of each value in all the fields in the dataframe.
# for col in df_claims.columns:
#     if col in ['CLAIM_NUM']:
#         continue

#     print col
#     value_cnt = df_claims[col].value_counts()
#     print value_cnt
#     print '************'
```

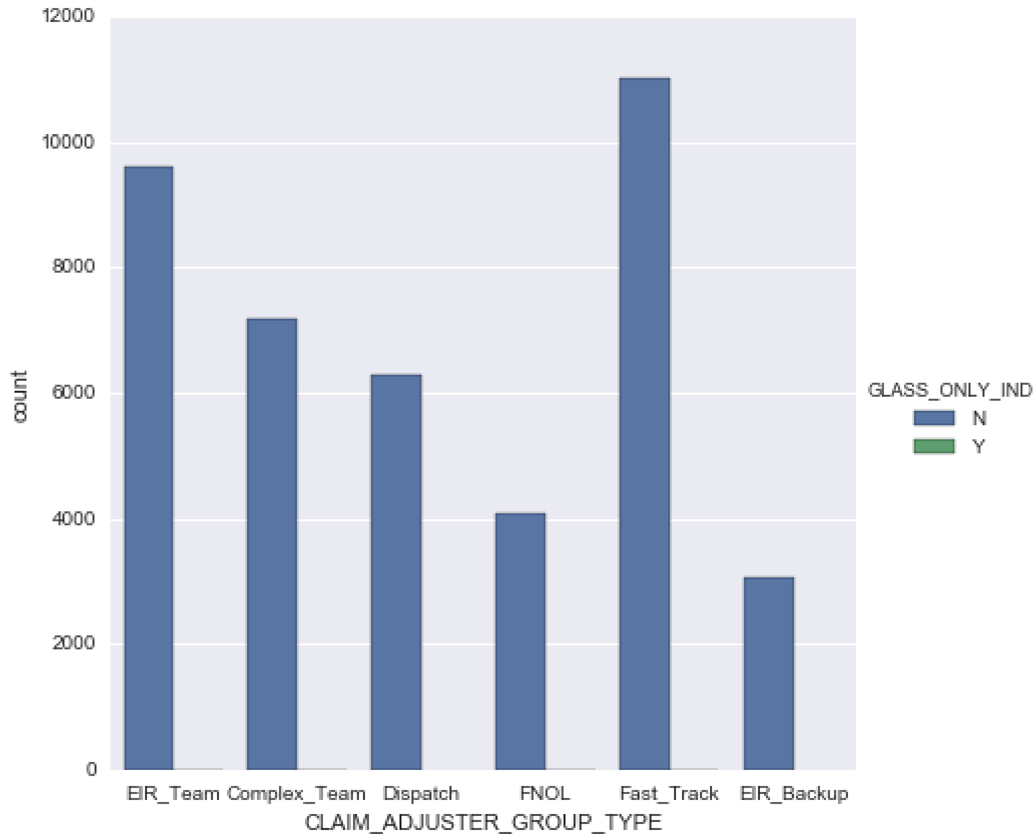
In [10]:
```
# Move the target field to array.['CLAIM_NUM','CLAIM_ADJUSTER_GROUP_TYPE']
df_target = df_claims[['CLAIM_NUM','CLAIM_ADJUSTER_GROUP_TYPE']]

df_target.groupby('CLAIM_ADJUSTER_GROUP_TYPE').count()
```

Out[10]:

	CLAIM_NUM
CLAIM_ADJUSTER_GROUP_TYPE	
Complex_Team	7181
Dispatch	6297
EIR_Backup	3047
EIR_Team	9616
FNOL	4089
Fast_Track	11022

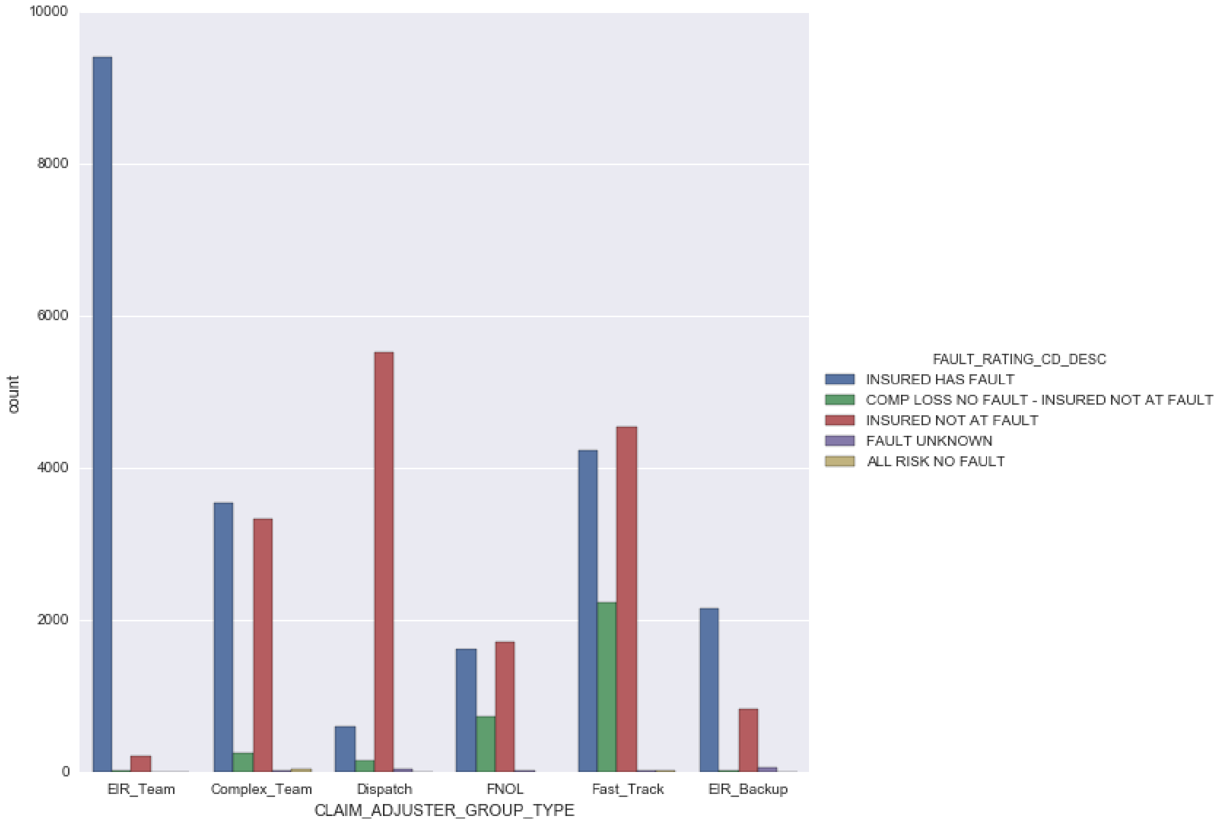
In [11]:
```
# Plot the glass only indicator and claim adjuster group.
df_report = df_claims[df_claims['CLAIM_ADJUSTER_GROUP_TYPE'] != 'FNOL']

sns.factorplot('CLAIM_ADJUSTER_GROUP_TYPE',data=df_claims,hue='GLASS_ONLY_IND' ,k
```

Out[11]: <seaborn.axisgrid.FacetGrid at 0x3f43c50>



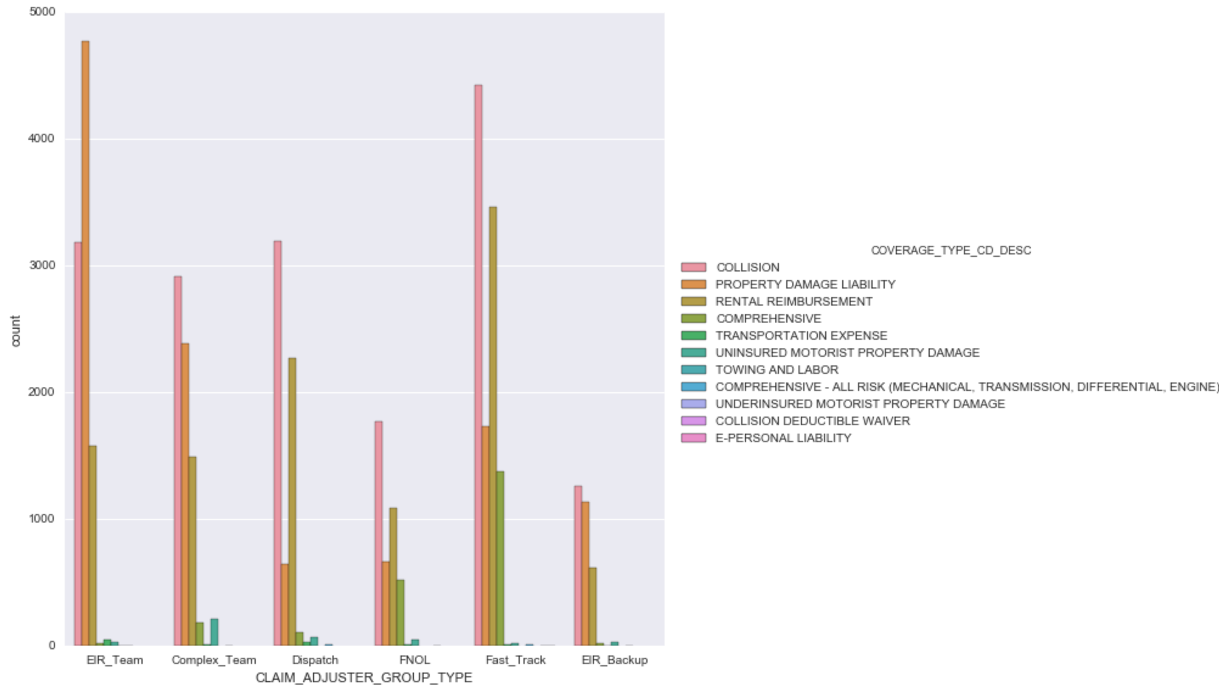
In [12]:
```
# Plot the fault rating code and claim adjuster group.
df_report = df_claims[df_claims['CLAIM_ADJUSTER_GROUP_TYPE'] != 'FNOL']

sns.factorplot('CLAIM_ADJUSTER_GROUP_TYPE',data=df_claims,hue='FAULT_RATING_CD_DE
```

Out[12]: <seaborn.axisgrid.FacetGrid at 0xd555e80>



In [13]:
```
# Plot the coverage code and claim adjuster group.
df_report = df_claims[df_claims['CLAIM_ADJUSTER_GROUP_TYPE'] != 'FNOL']

sns.factorplot('CLAIM_ADJUSTER_GROUP_TYPE',data=df_claims,hue='COVERAGE_TYPE_CD_D
```

Out[13]: <seaborn.axisgrid.FacetGrid at 0xd555f60>



In [14]:
```
# drop fields which are not useful -
df_claims.drop(['DAMAGE_DESC', 'PROPERTY_DESC', 'CLAIM_TYPE_DESC','INCIDENT_TYPE_
'NON_CHARGEABLE_CLAIM_IND', 'INCIDENT_ONLY_IND','LOSS_SUB_TYPE_DESC','CLAIM_FILE_
```

In [15]:
```
# Encode the nominal features - String to integers.
for col in df_claims.columns:
    if col in ['VEH_COUNT','LIABILITY_PERCENTAGE']:
        continue

    claim_labelEncode(df_claims,col)
```

In [16]:
```
# Impute the missing values in the dataframe.
claim_imp = claim_Imputer(df_claims)
```

In [17]:
```
# Convert to dataframe.
df_claim_imp = pd.DataFrame(data=claim_imp, columns=df_claims.columns)
```

In [18]:

```
# Standarize the features so the features are on same - scale.
# array_std =  claim_standardScaler(df_claim_imp[['LIABILITY_PERCENTAGE']])

# df_std = pd.DataFrame(data=array_std,columns=['LIABILITY_PERCENTAGE'])

# df_claim_imp.drop(['LIABILITY_PERCENTAGE'], inplace=True,axis=1)

# df_claim_std = pd.concat([df_claim_imp,df_std],axis=1)

df_claim_std = df_claim_imp
df_claim_std.head()
```

Out[18]:

	codeCOVERAGE_NM	codeCOVERAGE_SUB_TYPE_CD_DESC	codeTOW_TYPE_CD_DES
0	1.0	0.0	4.0
1	5.0	5.0	4.0
2	6.0	6.0	4.0
3	1.0	0.0	4.0
4	5.0	5.0	4.0

5 rows × 21 columns

In [19]:
```
# Create target dataframe. Set the fast track values to 1 and other groups to 0.
df_target.loc[df_target['CLAIM_ADJUSTER_GROUP_TYPE'] != 'Fast_Track', 'CLAIM_ADJU
df_target.loc[df_target['CLAIM_ADJUSTER_GROUP_TYPE'] == 'Fast_Track', 'CLAIM_ADJU
df_target.groupby('CLAIM_ADJUSTER_GROUP_TYPE').count()
```

```
C:\Users\gbu4moh\AppData\Local\Enthought\Canopy\User\lib\site-packages\pandas\c
ore\indexing.py:128: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  self._setitem_with_indexer(indexer, value)
C:\Users\gbu4moh\AppData\Local\Enthought\Canopy\User\lib\site-packages\ipykerne
l\__main__.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  from ipykernel import kernelapp as app
C:\Users\gbu4moh\AppData\Local\Enthought\Canopy\User\lib\site-packages\ipykerne
l\__main__.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
  app.launch_new_instance()
```

Out[19]:

	CLAIM_NUM
CLAIM_ADJUSTER_GROUP_TYPE	
0	30230
1	11022

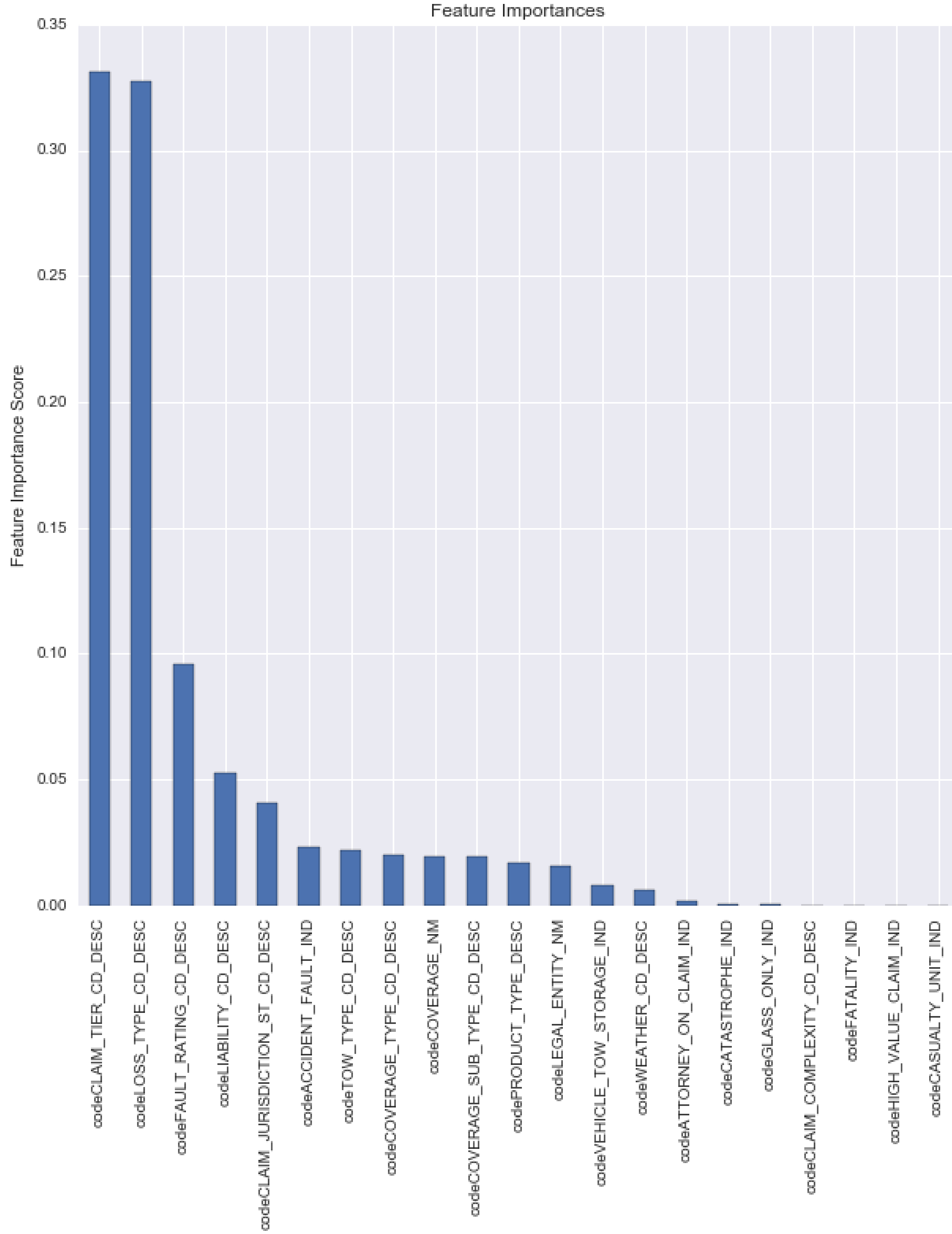
In [20]:

```python
# Use the Random forest classifier to identify the variable importance.
forest = RandomForestClassifier(n_estimators=1000, random_state=0,n_jobs=-1)

X_train, X_test, y_train, y_test = train_test_split(df_claim_std, list(df_target[

labels = df_claim_std.columns
forest.fit(X_train,y_train)
importances = forest.feature_importances_
indices = np.argsort(importances) [::-1]

for f in range(X_train.shape[1]):
    print(labels[f], importances[indices[f]])

feat_imp = pd.Series(importances, labels).sort_values(ascending=False)
feat_imp.plot(kind='bar', title='Feature Importances',figsize=(10,10))
plt.ylabel('Feature Importance Score')
```

```
('codeCOVERAGE_NM', 0.33120909816568078)
('codeCOVERAGE_SUB_TYPE_CD_DESC', 0.32737019119593369)
('codeTOW_TYPE_CD_DESC', 0.095948524702808485)
('codeCOVERAGE_TYPE_CD_DESC', 0.052279342014457809)
('codeVEHICLE_TOW_STORAGE_IND', 0.040380598935093506)
('codeATTORNEY_ON_CLAIM_IND', 0.023023529470336007)
('codePRODUCT_TYPE_DESC', 0.021875706496223352)
('codeGLASS_ONLY_IND', 0.019732626939262192)
('codeLIABILITY_CD_DESC', 0.019581345394470415)
('codeCATASTROPHE_IND', 0.019073826759692811)
('codeFAULT_RATING_CD_DESC', 0.016952099438646343)
('codeCLAIM_TIER_CD_DESC', 0.01566834599296309)
('codeCLAIM_COMPLEXITY_CD_DESC', 0.0079588240237430802)
('codeLOSS_TYPE_CD_DESC', 0.0060141628108448435)
('codeCLAIM_JURISDICTION_ST_CD_DESC', 0.0019210044761429716)
('codeWEATHER_CD_DESC', 0.00076214346082284555)
('codeLEGAL_ENTITY_NM', 0.0002486297228772104)
('codeACCIDENT_FAULT_IND', 0.0)
('codeHIGH_VALUE_CLAIM_IND', 0.0)
('codeFATALITY_IND', 0.0)
('codeCASUALTY_UNIT_IND', 0.0)
```

Out[20]:

```
<matplotlib.text.Text at 0xd8d3b00>
```



In [21]:
```
# first 23 fields needs to be hot-encoded.
ohe = OneHotEncoder(categorical_features=np.arange(0,21))
df_claim_hot = ohe.fit_transform(df_claim_std)
```

In [22]:
```
# Create data frame of hot-encoded array.
df_claim_mod = pd.DataFrame(df_claim_hot.toarray())
```

In [23]:
```
# Split the dataset to train and test dataset. 70% of data is trained and 30% of
X_train, X_test, y_train, y_test = train_test_split(df_claim_mod, list(df_target[
```

In [24]:
```
# Train the Logistic Regression model.
param_range = [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09]

for param in param_range:
    print('C : ',param)
    lr = LogisticRegression(penalty='l2',C=param)
    lr.fit(X_train,y_train)
    print('training accuracy:', lr.score(X_train,y_train))
    print('test accuracy: ', lr.score(X_test, y_test))
```

```
('C : ', 0.01)
('training accuracy:', 0.77392990718936139)
('test accuracy: ', 0.76656431803490632)
('C : ', 0.02)
('training accuracy:', 0.7774622523895276)
('test accuracy: ', 0.76979638009049778)
('C : ', 0.03)
('training accuracy:', 0.78078681257791938)
('test accuracy: ', 0.77302844214608923)
('C : ', 0.04)
('training accuracy:', 0.78109849009558108)
('test accuracy: ', 0.7738364576599871)
('C : ', 0.05)
('training accuracy:', 0.78175647596620035)
('test accuracy: ', 0.77674531351001941)
('C : ', 0.06)
('training accuracy:', 0.78186036847208751)
('test accuracy: ', 0.77674531351001941)
('C : ', 0.07)
('training accuracy:', 0.78196426097797478)
('test accuracy: ', 0.77682611506140919)
('C : ', 0.08)
('training accuracy:', 0.78203352264856629)
('test accuracy: ', 0.77682611506140919)
('C : ', 0.09)
('training accuracy:', 0.78213741515445356)
('test accuracy: ', 0.77690691661279898)
```

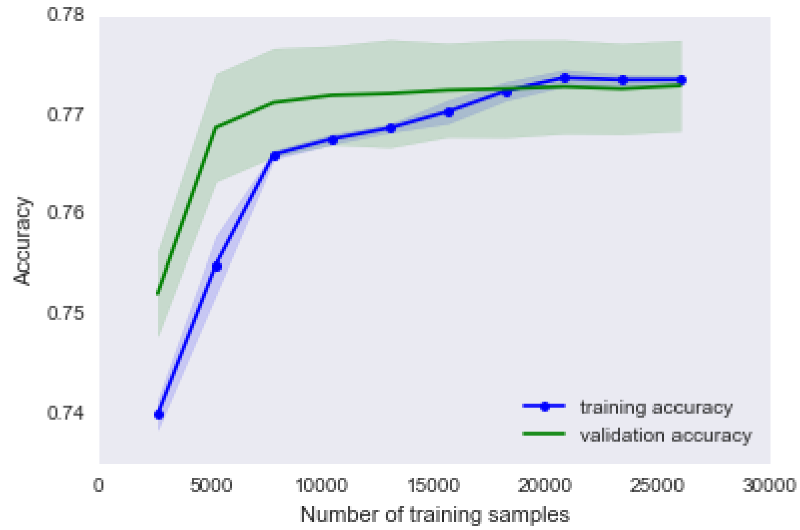
In [25]:
```
# Train the SVM model.
param_range = [0.0001,0.001,0.01,0.1,1.0,10.0,100.0,1000.0]


for param in param_range:
    svm = SVC(kernel ='sigmoid',C=param,random_state= 0)
    svm.fit(X_train,y_train)

    print('training accuracy:', svm.score(X_train,y_train))
    print('test accuracy: ', svm.score(X_test, y_test))
```

```
('training accuracy:', 0.73566283418756062)
('test accuracy: ', 0.7261635423400129)
('training accuracy:', 0.73566283418756062)
('test accuracy: ', 0.7261635423400129)
('training accuracy:', 0.73566283418756062)
('test accuracy: ', 0.7261635423400129)
('training accuracy:', 0.74993073832940849)
('test accuracy: ', 0.73965740142210734)
('training accuracy:', 0.76856212771852062)
('test accuracy: ', 0.75816095669036843)
('training accuracy:', 0.7746225238952763)
('test accuracy: ', 0.76454427925016155)
('training accuracy:', 0.73074525557556447)
('test accuracy: ', 0.72745636716224948)
('training accuracy:', 0.73922981022302259)
('test accuracy: ', 0.73553652230122823)
```

In [26]:

```
# Plot the training Curve
pipe_lr = Pipeline([('clf',LogisticRegression(penalty='l2',random_state=0,C=0.01)

train_sizes,train_scores,test_scores =     learning_curve(estimator=pipe_lr,
                  X=X_train,
                  y=y_train,
                  train_sizes=np.linspace(0.1,1.0,10),
                  cv=10,
                  n_jobs=1
                  )

# for x,y,z in zip(train_sizes,train_scores,test_scores):
#     print('train size ', x)
#     print('train scores ',y)
#     print('test scores ', z)


train_mean = np.mean(train_scores,axis=1)
train_std = np.std(train_scores,axis=1)
test_mean = np.mean(test_scores,axis=1)
test_std = np.std(test_scores,axis=1)

print('train_mean ', train_mean)
print('train_std ', train_std)
print('test_mean ', test_mean)
print('test_std ', test_std)

plt.plot(train_sizes,train_mean,color='blue',marker='o',markersize=5,label='train

plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, alp

plt.plot(train_sizes,test_mean,color='green',marker='x',markersize=5,label='valid

plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, alpha=0

plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
# plt.ylim([0.8,1.0])
plt.show()
```

```
('train_mean ', array([ 0.73991532,  0.75495478,  0.76608517,  0.76765442,  0.7
687678 ,
        0.77042073,  0.77249038,  0.77379864,  0.7735933 ,  0.77360988]))
('train_std ', array([ 0.00150115,  0.00289429,  0.00042232,  0.00051272,  0.00
041475,
        0.00117043,  0.00090807,  0.0008127 ,  0.00053942,  0.00042531]))
('test_mean ', array([ 0.75211247,  0.76880529,  0.77133337,  0.77206079,  0.77
223383,
        0.77258009,  0.77271865,  0.7729264 ,  0.77271855,  0.77303008]))
('test_std ', array([ 0.00422554,  0.00541723,  0.0054412 ,  0.00495734,  0.005
41199,
        0.00470505,  0.00489368,  0.00470259,  0.00455039,  0.00455052]))
```



In [27]:
```
# confusion matrix

y_predict = lr.predict(X_test)

confusion_matrix(y_test, y_predict)

#print ("Accuracy : %.4g" % metrics.accuracy_score(y_test, y_predict))
#print ("AUC Score (Train): %f" % metrics.roc_auc_score(y_test, y_predict))

#cv_score = cross_validation.cross_val_score(alg, dtrain[predictors], dtrain['Dis
# print "CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mea
```

Out[27]:
```
array([[8103,  884],
       [1877, 1512]])
```

In [28]:
```
# Gradient boosting classifier.
param_test = {'n_estimators':range(20,81,10)}
gsearch1 = GridSearchCV(estimator = GradientBoostingClassifier(learning_rate=0.1,
param_grid = param_test, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
gsearch1.fit(X_train,y_train)
```

Out[28]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=Non
e,
              learning_rate=0.1, loss='deviance', max_depth=8,
              max_features='sqrt', max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=50,
              min_samples_split=500, min_weight_fraction_leaf=0.0,
              n_estimators=100, presort='auto', random_state=10,
              subsample=0.8, verbose=0, warm_start=False),
       fit_params={}, iid=False, n_jobs=4,
       param_grid={'n_estimators': [20, 30, 40, 50, 60, 70, 80]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='roc_auc', verbose=0)
```

In [29]:
```
gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
C:\Users\gbu4moh\AppData\Local\Enthought\Canopy\User\lib\site-packages\sklearn
\model_selection\_search.py:662: DeprecationWarning: The grid_scores_ attribute
 was deprecated in version 0.18 in favor of the more elaborate cv_results_ attr
ibute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

Out[29]:
```
([mean: 0.84697, std: 0.00728, params: {'n_estimators': 20},
  mean: 0.84918, std: 0.00776, params: {'n_estimators': 30},
  mean: 0.85005, std: 0.00813, params: {'n_estimators': 40},
  mean: 0.85093, std: 0.00823, params: {'n_estimators': 50},
  mean: 0.85120, std: 0.00795, params: {'n_estimators': 60},
  mean: 0.85174, std: 0.00808, params: {'n_estimators': 70},
  mean: 0.85182, std: 0.00810, params: {'n_estimators': 80}],
 {'n_estimators': 80},
 0.85181943328741738)
```

In [30]:
```
y_predict = gsearch1.predict(X_test)

confusion_matrix(y_test, y_predict)
```

Out[30]:
```
array([[7591, 1396],
       [1233, 2156]])
```

In [31]:
```
print('training accuracy:', gsearch1.score(X_train,y_train))
print('test accuracy: ', gsearch1.score(X_test, y_test))
```

```
('training accuracy:', 0.85568745454417727)
('test accuracy: ', 0.85105734347665818)
```

In [35]:
```
import theano
from theano import tensor

# from keras.models import Sequential
# from keras.layers.core import Dense
# from keras.optimizers import SGD

# np.random.seed(1)

# model = Sequential()
# model.add(Dense(input_dim=X_train.shape[1],
#                 output_dim=50,
#                 init='uniform',
#                 activation='tanh'))

# model.add(Dense(input_dim=50,
#                 output_dim=50,
#                 init='uniform',
#                 activation='tanh'))

# model.add(Dense(input_dim=50,
#                 output_dim=y_train_ohe.shape[1],
#                 init='uniform',
#                 activation='softmax'))

# sgd = SGD(lr=0.001, decay=1e-7, momentum=.9)
# model.compile(loss='categorical_crossentropy', optimizer=sgd)

# model.fit(X_train, y_train,
#           nb_epoch=50,
#           batch_size=300,
#           verbose=1,
#           validation_split=0.1,
#           show_accuracy=True)
```

In [ ]: