submission-notebook

March 3, 2017

0.1 Imports

In [1]:
```
%pylab inline
%config InlineBackend.figure_format = 'retina'
import pandas as pd
import seaborn as sns
```

```
Populating the interactive namespace from numpy and matplotlib
```

0.2 Preprocessing

In [2]:
```
%%time
# Open data files
path = "./data/"

train = pd.read_csv(path+'train.csv', encoding='iso-8859-1')[::]
test = pd.read_csv(path+'test.csv')
test_ticket_id = np.array(test['ticket_id'])

train = train.set_index('ticket_id')
test = test.set_index('ticket_id')

# Drop the violators who were found not responsible
train.dropna(subset=['compliance'], inplace=True)

# Drop some uninformative features
for column_name in ['inspector_name', 'violator_name',
                    'violation_zip_code', 'violation_street_number', 'viola
                    'mailing_address_str_number', 'mailing_address_str_name
                    'state', 'zip_code', 'non_us_str_code', 'country',
                    'violation_description',
                    'admin_fee', 'state_fee', 'late_fee']:
    test.drop(column_name, axis=1, inplace=True)



# Convert datetime columns into years/months/days
```

```
for column_name in ['ticket_issued_date', 'hearing_date']:
    print('Converting datetime to years/months/days...', column_name)

    # test
    day_time = pd.to_datetime(test[column_name])
    test.drop(column_name, axis=1, inplace=True)
    test[column_name+'_month'] = np.array(day_time.dt.month)
    test[column_name+'_year'] = np.array(day_time.dt.year)
    test[column_name+'_day'] = np.array(day_time.dt.day)
    test[column_name+'_dayofweek'] = np.array(day_time.dt.dayofweek)

    # train
    day_time = pd.to_datetime(train[column_name])
    train.drop(column_name, axis=1, inplace=True)
    train[column_name+'_month'] = np.array(day_time.dt.month)
    train[column_name+'_year'] = np.array(day_time.dt.year)
    train[column_name+'_day'] = np.array(day_time.dt.day)
    train[column_name+'_dayofweek'] = np.array(day_time.dt.dayofweek)

# Convert string columns to categorical
cols = test.select_dtypes(exclude=['float', 'int']).columns
len_train = len(train)
temp_concat = pd.concat((train[cols], test[cols]), axis=0)

# Some filtering on violation_code to make it more manageable
temp_concat['violation_code'] = temp_concat['violation_code'].apply(lambda
temp_concat['violation_code'] = temp_concat['violation_code'].apply(lambda
temp_concat['violation_code'][temp_concat['violation_code'].apply(lambda x:

# Make all codes with < 10 occurrences null
counts = temp_concat['violation_code'].value_counts()
temp_concat['violation_code'][temp_concat['violation_code'].isin(counts[cou

for column_name in cols:
    print('Converting to categorical...', column_name, '# variables:', len(
    dummies = pd.get_dummies(temp_concat[column_name])
    temp_concat[dummies.columns] = dummies
    temp_concat.drop(column_name, axis=1, inplace=True)
    train.drop(column_name, axis=1, inplace=True)
    test.drop(column_name, axis=1, inplace=True)

train[temp_concat.columns] = temp_concat.loc[train.index]
test[temp_concat.columns] = temp_concat.loc[test.index]

features = list( test.columns )
target = ['compliance']

print("Number of features:", len(features))
```

```
        # Train Set
        X = train[features]
        y = np.array(train[target]).ravel()

        # Normalize
        mn = X.mean()
        std = X.std()
        X = (X - mn)/std

        X = X.replace([np.inf, -np.inf], np.nan)
        X[pd.isnull(X)] = 0

        # Submissions Set
        Xtest = (test[features] - mn) / std
        Xtest = Xtest.replace([np.inf, -np.inf], np.nan)
        Xtest[pd.isnull(Xtest)] = 0
```

```
<string>:2: DtypeWarning: Columns (11,12,31) have mixed types. Specify dtype option
```

```
Converting datetime to years/months/days... ticket_issued_date
Converting datetime to years/months/days... hearing_date
Converting to categorical... agency_name # variables: 5
Converting to categorical... violation_code # variables: 72
Converting to categorical... disposition # variables: 8
Converting to categorical... grafitti_status # variables: 2
Number of features: 97
CPU times: user 5.49 s, sys: 2.1 s, total: 7.59 s
Wall time: 7.43 s
```

0.3 Evaluation

0.3.1 Define Models

In [3]:
```
        # Imports
        from sklearn.linear_model import SGDClassifier, Perceptron
        from sklearn.ensemble import GradientBoostingClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble     import RandomForestClassifier
        from sklearn.ensemble     import RandomForestRegressor
        from sklearn.neural_network import MLPClassifier
        from sklearn.cross_validation import train_test_split
        from sklearn.model_selection import cross_val_score

        # Add classifiers
        classifiers = {
```

```
            "ASGD": SGDClassifier(average=True),
            "SAG": LogisticRegression(solver='sag', tol=1e-1, C=1.e4 / train[featur
            "RF_C": RandomForestClassifier(max_depth=25),
            "GradBoost": GradientBoostingClassifier(learning_rate=0.05, max_depth=1
            "VanillaGrad": GradientBoostingClassifier(),
            'MLP': MLPClassifier()
        }

        # Create Train/Test split for evaluation.
        # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
```

```
/Users/thealex/anaconda3/lib/python3.5/site-packages/sklearn/cross_validation.py:44
  "This module will be removed in 0.20.", DeprecationWarning)
```

0.3.2 Evaluate with K-Fold Cross Validation

In [4]:
```
%%time

        scores = dict()
        # Select the model
        for classifier_type in classifiers.keys():
            # Train classifier
            clf = classifiers[classifier_type]
            # Score classifier
            # If using cross_val_score, there is no need for train test split.
            model_score = cross_val_score(clf, X, y, cv=30, n_jobs=-1)
            # Record score
            scores[classifier_type] = model_score

        scores = pd.DataFrame(data=scores)
        display(scores)
```

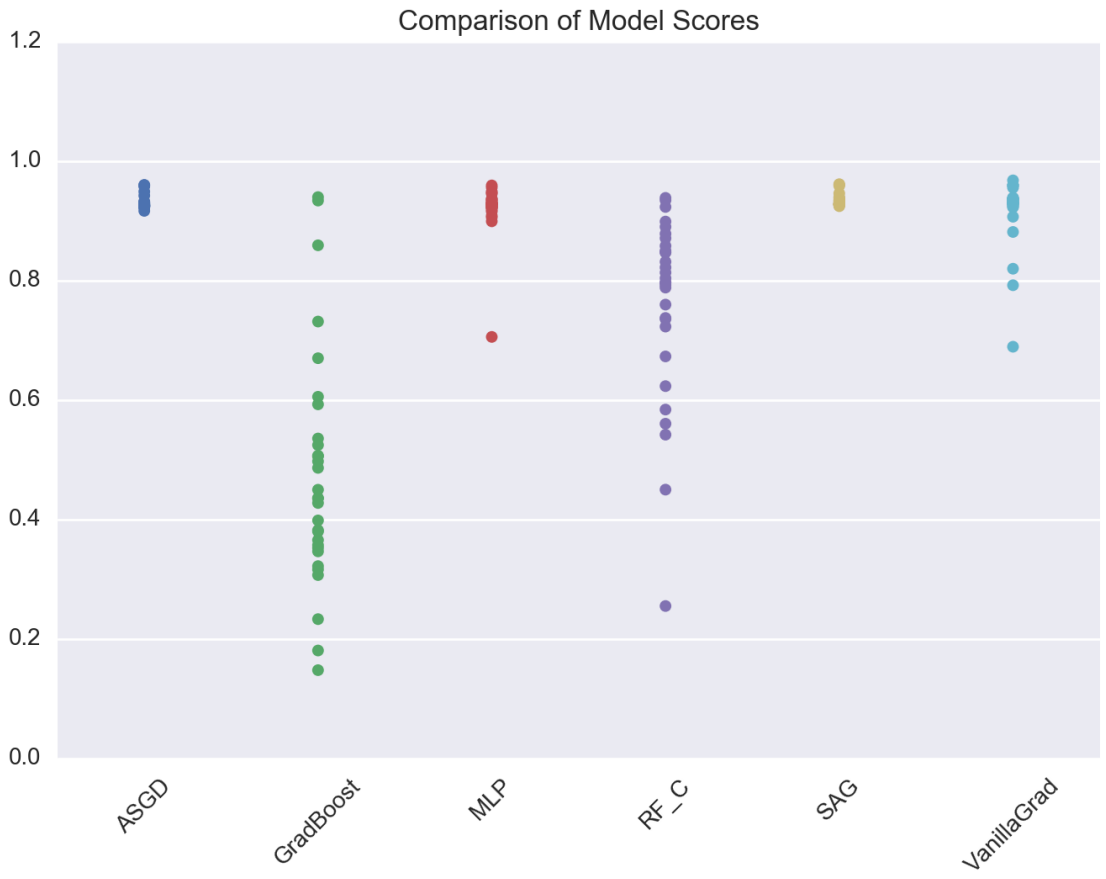
	ASGD	GradBoost	MLP	RF_C	SAG	VanillaGrad
0	0.916698	0.148218	0.705816	0.255535	0.924578	0.792308
1	0.925328	0.322326	0.899250	0.623452	0.927205	0.881426
2	0.924203	0.380300	0.908630	0.822139	0.927392	0.928143
3	0.922514	0.497561	0.921951	0.847280	0.927580	0.930957
4	0.925328	0.316698	0.921576	0.673171	0.928330	0.819887
5	0.926266	0.307317	0.922702	0.560600	0.927017	0.928893
6	0.925704	0.450094	0.928893	0.735835	0.928518	0.930394
7	0.924390	0.535835	0.927580	0.813508	0.928330	0.929268
8	0.924015	0.507129	0.923827	0.804128	0.928143	0.931144
9	0.930394	0.486679	0.931707	0.723077	0.932645	0.937336
10	0.942214	0.233583	0.945403	0.584240	0.939024	0.955535
11	0.960038	0.357411	0.959287	0.878424	0.961351	0.959287
12	0.957598	0.670169	0.946904	0.870919	0.960413	0.959850
13	0.959850	0.859099	0.955910	0.923265	0.958724	0.967730

```
14  0.948968   0.731520  0.949156  0.849906  0.945966     0.959099
15  0.926642   0.436023  0.905816  0.737523  0.927392     0.922702
16  0.924953   0.382364  0.915572  0.759850  0.928330     0.927767
17  0.927003   0.506099  0.924188  0.890036  0.925877     0.930569
18  0.924751   0.398761  0.928504  0.858135  0.928317     0.937699
19  0.926065   0.524864  0.936198  0.847251  0.927754     0.936011
20  0.923062   0.592982  0.926253  0.796772  0.927191     0.924188
21  0.924188   0.365922  0.924564  0.788328  0.927754     0.925502
22  0.925127   0.346969  0.929630  0.831488  0.928129     0.934322
23  0.928116   0.435811  0.928866  0.796922  0.928303     0.935060
24  0.928303   0.427928  0.928679  0.792605  0.930368     0.934685
25  0.927928   0.605668  0.932995  0.898836  0.930931     0.936562
26  0.930931   0.933934  0.935248  0.934497  0.935248     0.936749
27  0.930368   0.939752  0.937125  0.938438  0.934497     0.938814
28  0.928116   0.352853  0.918168  0.542230  0.938814     0.689377
29  0.932620   0.181119  0.927553  0.450263  0.932432     0.906907


CPU times: user 46.4 s, sys: 2.71 s, total: 49.1 s
Wall time: 2h 58min 30s
```

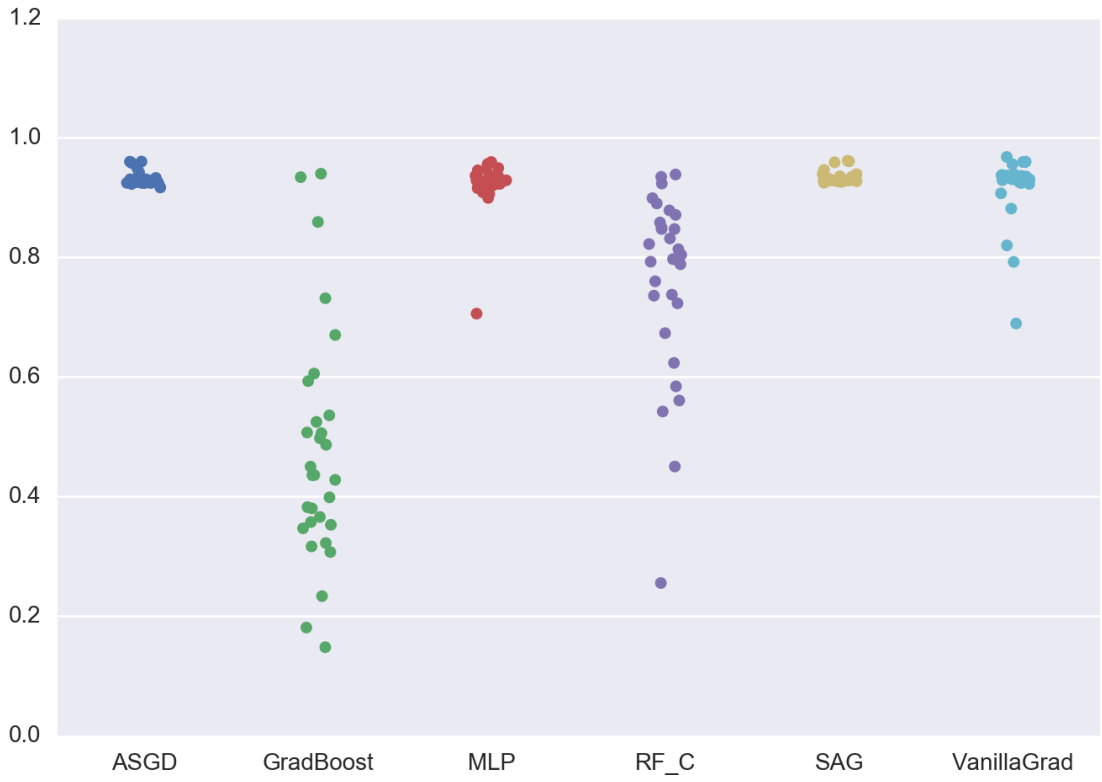
```
In [5]: sns.stripplot(data=scores)
        plt.xticks(rotation=45)
        plt.title('Comparison of Model Scores')

Out[5]: <matplotlib.text.Text at 0x114c2e630>
```



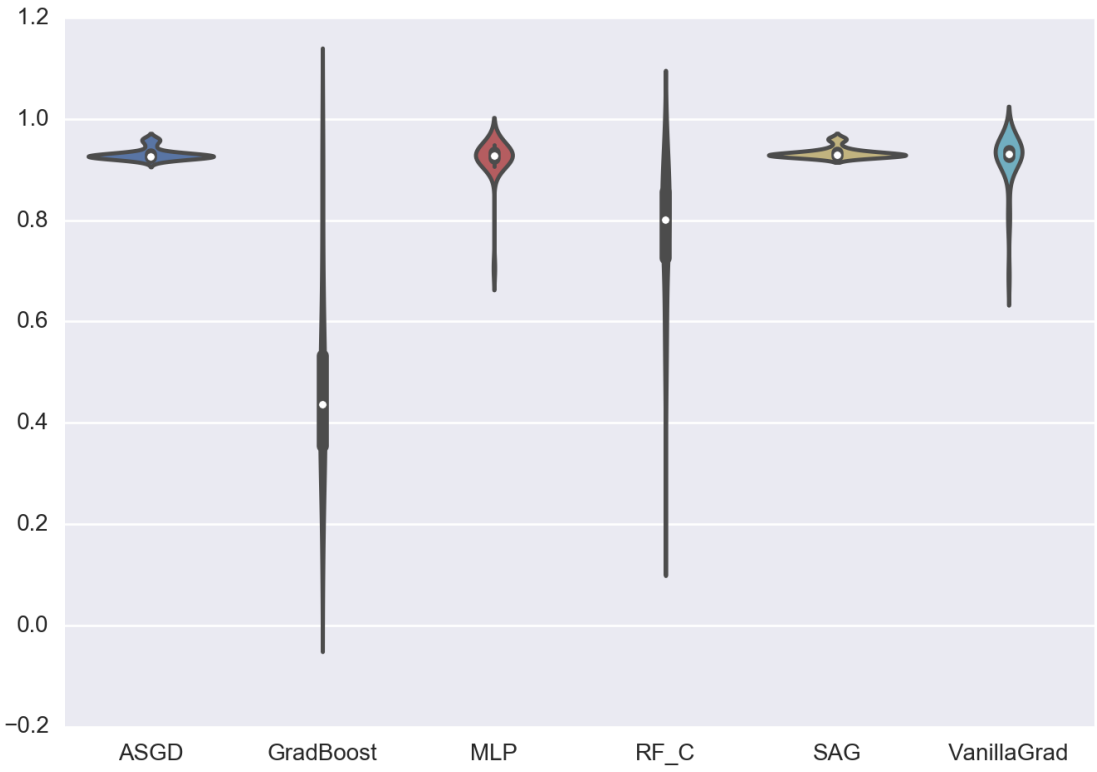

```
In [6]: sns.stripplot(data=scores, jitter=True)
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1145df2e8>
```




```
In [7]: sns.violinplot(data=scores)

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1123d2c18>
```



0.4 Submission

```
In [9]: def make_submission(model_name, clf):
            # Train classifier
            clf.fit(X, y)

            # Predict
            y_pred = np.array(clf.predict(Xtest))
            y_pred = y_pred - y_pred.min()
            y_pred = y_pred / y_pred.max()

            # Save to CSV
            df = {"ticket_id":test_ticket_id, "compliance":y_pred}
            df = pd.DataFrame(df, columns=["ticket_id", "compliance"])
            df.to_csv("./data/submission_%s.csv" % model_name, index=False)

        def make_all_submissions(classifiers):
            """classifiers: A dictionary of classifier name keys and sklearn classi
            for model_name, clf in classifiers.items():
                make_submission(model_name, clf)

        # Make all submissions
```

```
        # make_all_submissions(classifiers)

        # Make individual submission
        target_clf = "MLP"
        make_submission(target_clf, classifiers[target_clf])
```

In [ ]: