Milestone_3_Submission

April 28, 2017

0.0.1 Milestone 3: Traditional statistical and machine learning methods, due Wednesday, April 19, 2017

Think about how you would address the genre prediction problem with traditional statistical or machine learning methods. This includes everything you learned about modeling in this course before the deep learning part. Implement your ideas and compare different classifiers. Report your results and discuss what challenges you faced and how you overcame them. What works and what does not? If there are parts that do not work as expected, make sure to discuss briefly what you think is the cause and how you would address this if you would have more time and resources.

You do not necessarily need to use the movie posters for this step, but even without a background in computer vision, there are very simple features you can extract from the posters to help guide a traditional machine learning model. Think about the PCA lecture for example, or how to use clustering to extract color information. In addition to considering the movie posters it would be worthwhile to have a look at the metadata that IMDb provides.

You could use Spark and the ML library to build your model features from the data. This may be especially beneficial if you use additional data, e.g., in text form.

You also need to think about how you are going to evaluate your classifier. Which metrics or scores will you report to show how good the performance is?

The notebook to submit this week should at least include:

- Detailed description and implementation of two different models
- Description of your performance metrics
- Careful performance evaluations for both models
- Visualizations of the metrics for performance evaluation
- Discussion of the differences between the models, their strengths, weaknesses, etc.
- Discussion of the performances you achieved, and how you might be able to improve them in the future

Preliminary Peer Assessment It is important to provide positive feedback to people who truly worked hard for the good of the team and to also make suggestions to those you perceived not to be working as effectively on team tasks. We ask you to provide an honest assessment of the contributions of the members of your team, including yourself. The feedback you provide should reflect your judgment of each team member's:

- Preparation – were they prepared during team meetings?
- Contribution – did they contribute productively to the team discussion and work?
- Respect for others' ideas – did they encourage others to contribute their ideas?

- Flexibility – were they flexible when disagreements occurred?

Your teammate's assessment of your contributions and the accuracy of your self-assessment will be considered as part of your overall project score.

Preliminary Peer Assessment: https://goo.gl/forms/WOYC7pwRCSU0yV3l1

0.1 Below is code to implement our models. After its implementation, we describe the purpose and functionalities of the 2 models chosen applied to 1 dataset.

We implemented both a SGD and Random forest once on a dataset consisting of a traditional feature set holding predictors such as crew, director, actors, etc. We also implemented the same models after creating a vectorization of each movies overview description using most frequent words for classification. Below is the code to create and tune all 4 models.

In [1]:
```
import collections
import cPickle
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import make_scorer, hamming_loss, classification_repor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import MultiLabelBinarizer, StandardScaler, Norm
from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report, accuracy_score, hamming_
import matplotlib.pyplot as plt
import re
from scipy import sparse
```

In [2]:
```
def load_part(file_name):
    with open(file_name, 'rb') as handle:
        return cPickle.load(handle)
```

In [3]:
```
def cutoff_labels(labels, cutoff):
    mlb = MultiLabelBinarizer()
    label_df = pd.DataFrame(mlb.fit_transform(labels))
    label_df.columns = mlb.classes_
    label_number_df = pd.DataFrame({'cnt': label_df.sum(axis=0)})
    major_genres = set(label_number_df[label_number_df['cnt'] > cutoff].inc
    return major_genres


def get_major_genres(plot_dict):
    labels = np.array([d['genres'] for d in plot_dict.values() if 'genres'
```

```
        # only leave genres mentioned in 2000 movies or more
        major_genres = cutoff_labels(labels, 2000)
        return major_genres

In [4]: def report_to_df(clfreport):
        lines = clfreport.split('\n')
        lines = filter(lambda l: not len(l.strip()) == 0, lines)

        header = lines[0]
        cls_lines = lines[1:-1]
        avg_line = lines[-1]

        assert header.split() == ['precision', 'recall', 'f1-score', 'support']
        assert avg_line.split()[0] == 'avg'
        cls_field_width = len(header) - len(header.lstrip())

        def parse_line(l):
            """Parse a line of classification_report"""
            cls_name = l[:cls_field_width].strip()
            precision, recall, fscore, support = l[cls_field_width:].split()
            precision = float(precision)
            recall = float(recall)
            fscore = float(fscore)
            support = int(support)
            return [cls_name, precision, recall, fscore, support]

        data = collections.OrderedDict()
        for l in cls_lines:
            ret = parse_line(l)
            data[ret[0]] = ret[1:]
        data['Total/avg'] = parse_line(avg_line)[1:]
        df = pd.DataFrame.from_dict(data, orient = 'index')
        df.columns = ['precision', 'recall', 'f1-score', 'support']
        return df

In [5]: root_folder = '..'

In [6]: # load TMDB movies dataset
        tmdb_movies = load_part(root_folder + '/data/tmdb_info.pickle')

        plot_dict = load_part(root_folder + '/data/plot.pickle')

        cast_dict = load_part(root_folder + '/data/cast10K.pickle')

        major_genres = get_major_genres(plot_dict)

        sample_tmdb_ids = np.array([tmdb_id for tmdb_id, d in cast_dict.items() if
                                    'genres' in d and 'genres' in plot_dict[tmdb_id
```

```
                                    plot_dict[tmdb_id]])
        print len(sample_tmdb_ids)
```

```
7669
```

In [7]:
```
def prepare_text_data(tmdb_dict, plot_dict, major_genres, sample_tmdb_ids):
    # add 'overview' from TMDB to 'plot' from IMDB (it is a list)
    for tmdb_id, imdb_movie in plot_dict.iteritems():
        if ('plot' in imdb_movie and tmdb_id in tmdb_dict and 'overview' in
                    tmdb_dict[tmdb_id].__dict__['overview'] is not None):
            imdb_movie['plot'].append(tmdb_dict[tmdb_id].__dict__['overview

    labels = np.array(
        [major_genres.intersection(plot_dict[tmdb_id]['genres']) for tmdb_i
         'genres' in plot_dict[tmdb_id] and 'plot' in plot_dict[tmdb_id]])
    print len(labels)

    # create the labels vector with only major genres
    mlb = MultiLabelBinarizer()
    y = mlb.fit_transform(labels)
    # the plot consists of a few parts, join them together
    features = np.array([''.join(plot_dict[tmdb_id]['plot']) for tmdb_id in
                         'genres' in plot_dict[tmdb_id] and 'plot' in plot_

    vectorizer = TfidfVectorizer(
        stop_words=stopwords.words("english"),
        token_pattern='[a-zA-Z]+[0-9]*',
        max_df=0.9,
        min_df=0.0001,
        dtype=np.float32,
    )
    return features, y, mlb.classes_, vectorizer
```

In [8]:
```
def prepare_cast_data(tmdb_dict, cast_dict, major_genres, sample_tmdb_ids):
    columns = [
        'director',
        'cast',
        'casting director',
        'miscellaneous crew',
        'original music',
        'producer',
        'cinematographer',
        'costume designer',
        'art direction']

    labels = np.array(
        [major_genres.intersection(cast_dict[tmdb_id]['genres']) for tmdb_i
```

```
            'genres' in cast_dict[tmdb_id]])
        # create the labels vector with only major genres
        mlb = MultiLabelBinarizer()
        y = mlb.fit_transform(labels)

        # combine all names separated by '|'
        features = []
        for tmdb_id in sample_tmdb_ids:
            imdb_movie = cast_dict[tmdb_id]
            if 'genres' not in imdb_movie:
                continue
            l = []
            for c in columns:
                if (c in imdb_movie):
                    l = l + [c['name'].encode('utf-8') for c in imdb_movie[c]]
            # add crew and cast from TMDB
            if (tmdb_id in tmdb_dict):
                tmdb_movie = tmdb_dict[tmdb_id].__dict__
                if ('crew' in tmdb_movie):
                    l = l + [c['name'].encode('utf-8') for c in tmdb_movie['cre
                if ('cast' in tmdb_movie):
                    l = l + [c['name'].encode('utf-8') for c in tmdb_movie['cas
            # remove duplicates before joiniing
            features.append('|'.join(set(l)))

        vectorizer = CountVectorizer(
            max_df=0.99,
            min_df=0.0002,
            stop_words=stopwords.words("english"),
            tokenizer=lambda x: x.split('|'),
            dtype=np.float32)

        return features, y, mlb.classes_, vectorizer
```

In [9]:
```
# get labels / features from the cast / crew data
cast_features, cast_y, cast_mlb_classes, cast_vectorizer =  prepare_cast_da
    tmdb_movies, cast_dict, major_genres, sample_tmdb_ids)

print np.shape(cast_features)
print np.shape(cast_y)
print len(cast_mlb_classes)

mlb_classes = cast_mlb_classes

# get labels / features from the text data
text_features, text_y, text_mlb_classes, text_vectorizer = prepare_text_dat
    tmdb_movies, plot_dict, major_genres,sample_tmdb_ids)
```

```
        print np.shape(text_features)
        print np.shape(text_y)
        print len(text_mlb_classes)
```

```
(7669L,)
(7669L, 15L)
15
7669
(7669L,)
(7669L, 15L)
15
```

In [10]:
```
# split into test / train data
cast_F_train, cast_F_test, text_F_train, text_F_test, y_train, y_test =  t
    cast_features, text_features, cast_y, test_size=0.25, random_state=42)

cast_X_train = cast_vectorizer.fit_transform(cast_F_train)
cast_X_test = cast_vectorizer.transform(cast_F_test)

print np.shape(cast_X_train)
print np.shape(cast_X_test)

text_X_train = text_vectorizer.fit_transform(text_F_train)
text_X_test = text_vectorizer.transform(text_F_test)

print np.shape(text_X_train)
print np.shape(text_X_test)

X_train = sparse.hstack((cast_X_train, text_X_train))
X_test = sparse.hstack((cast_X_test, text_X_test))

print np.shape(X_train)
print np.shape(X_test)
```

```
(5751, 32775)
(1918, 32775)
(5751, 37402)
(1918, 37402)
(5751, 70177)
(1918, 70177)
```

In [11]:
```
def sgd(X_train, y_train):
    param_grid = { 'estimator__alpha': np.logspace(-5, -1, num=50)}
    model = OneVsRestClassifier(SGDClassifier(class_weight='balanced', ran
    model_tuning = GridSearchCV(
        model,
        param_grid = param_grid,
```

```
                scoring=make_scorer(hamming_loss, greater_is_better=False),
                cv=3,
                n_jobs=-1,
                verbose=1,
            )
            model_tuning.fit(X_train, y_train)
            return model_tuning
```

SGD model based on full features data set

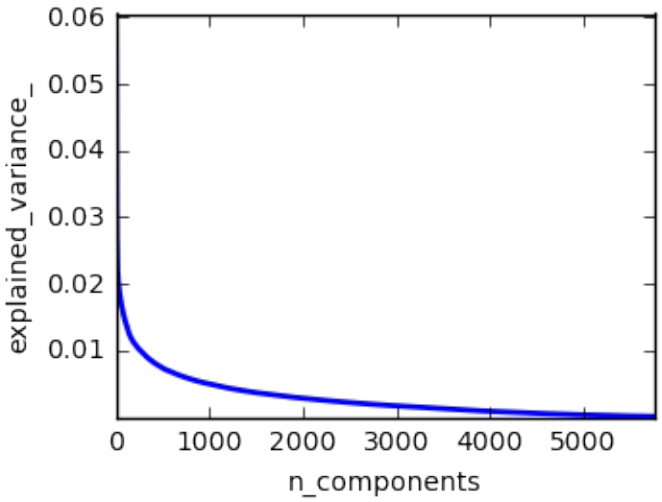
```
In [12]: sgd_model = sgd(X_train, y_train)

Fitting 3 folds for each of 50 candidates, totalling 150 fits


[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.7s
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:   31.5s finished


In [13]: y_train_pred = sgd_model.predict(X_train)
         y_test_pred = sgd_model.predict(X_test)

         train_report = classification_report(y_train, y_train_pred, target_names=m
         test_report =  classification_report(y_test, y_test_pred, target_names=mlb

         sgd_model_performance_df = report_to_df(test_report)

         print sgd_model.best_params_
         print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
         print 'Train performance: '
         print train_report
         print 'Test performance: '
         print test_report
```

```
{'estimator__alpha': 0.0049417133613238332}
hamming loss: 0.108
Train performance:
             precision    recall  f1-score   support

     Action       0.954     0.889     0.920       682
  Adventure       0.952     0.918     0.935       451
  Animation       0.974     0.815     0.888       373
     Comedy       0.989     0.764     0.862      1609
      Crime       0.985     0.918     0.950       561
Documentary       0.648     0.994     0.784       655
      Drama       0.969     0.928     0.948      2490
     Family       0.968     0.905     0.935       399
    Fantasy       0.890     0.913     0.902       311
     Horror       0.956     0.904     0.930       606
```

```
    Mystery       0.965     0.935     0.950       322
    Romance       0.972     0.886     0.927       796
     Sci-Fi       0.931     0.940     0.936       318
      Short       0.781     0.990     0.874       621
   Thriller       0.974     0.884     0.927       793

avg / total       0.938     0.896     0.911     10987

Test performance:
             precision    recall  f1-score   support

     Action       0.625     0.150     0.242       200
  Adventure       0.708     0.107     0.186       159
  Animation       0.930     0.452     0.608       146
     Comedy       0.795     0.229     0.355       525
      Crime       0.438     0.077     0.131       181
Documentary       0.506     0.938     0.657       227
      Drama       0.698     0.602     0.647       837
     Family       0.820     0.336     0.476       149
    Fantasy       0.348     0.076     0.125       105
     Horror       0.652     0.253     0.364       178
    Mystery       0.333     0.065     0.108        93
    Romance       0.563     0.136     0.219       295
     Sci-Fi       0.500     0.128     0.203        94
      Short       0.526     0.655     0.584       200
   Thriller       0.528     0.144     0.226       264

avg / total       0.638     0.354     0.407      3653
```

SGD model based on reduced features (PCA) data set

```
In [14]: pca = PCA( )
         X_train_pca = pca.fit_transform(X_train.toarray())
         X_test_pca = pca.transform(X_test.toarray())
         plt.figure(1, figsize=(4, 3))
         plt.clf()
         plt.axes([.2, .2, .7, .7])
         plt.plot(pca.explained_variance_, linewidth=2)
         plt.axis('tight')
         plt.xlabel('n_components')
         plt.ylabel('explained_variance_')
         plt.show()
```




```
In [15]: sgd_pca_model = sgd(X_train_pca, y_train)

Fitting 3 folds for each of 50 candidates, totalling 150 fits


[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:  7.7min finished


In [16]: y_train_pred = sgd_pca_model.predict(X_train_pca)
         y_test_pred = sgd_pca_model.predict(X_test_pca)

         train_report = classification_report(y_train, y_train_pred, target_names=n
         test_report =  classification_report(y_test, y_test_pred, target_names=mlb

         sgd_pca_model_performance_df = report_to_df(test_report)

         print sgd_pca_model.best_params_
         print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
         print 'Train performance: '
         print train_report
         print 'Test performance: '
         print test_report

{'estimator__alpha': 0.012648552168552958}
hamming loss: 0.120
Train performance:
             precision    recall  f1-score   support

     Action       0.952     0.723     0.822       682
```

```
  Adventure       0.948     0.734     0.827        451
  Animation       0.980     0.641     0.775        373
     Comedy       0.993     0.447     0.617       1609
      Crime       0.963     0.786     0.866        561
Documentary       0.434     0.998     0.605        655
      Drama       0.957     0.884     0.919       2490
     Family       0.954     0.779     0.858        399
    Fantasy       0.926     0.685     0.787        311
     Horror       0.969     0.622     0.758        606
    Mystery       0.962     0.792     0.869        322
    Romance       0.946     0.637     0.761        796
     Sci-Fi       0.947     0.780     0.855        318
      Short       0.969     0.498     0.657        621
   Thriller       0.963     0.689     0.803        793

avg / total       0.931     0.714     0.786      10987

Test performance:
             precision    recall  f1-score   support

     Action       0.625     0.100     0.172        200
  Adventure       0.600     0.075     0.134        159
  Animation       0.950     0.390     0.553        146
     Comedy       0.864     0.109     0.193        525
      Crime       0.455     0.055     0.099        181
Documentary       0.319     0.982     0.481        227
      Drama       0.683     0.583     0.629        837
     Family       0.868     0.309     0.455        149
    Fantasy       0.333     0.029     0.053        105
     Horror       0.565     0.073     0.129        178
    Mystery       0.375     0.032     0.059         93
    Romance       0.478     0.075     0.129        295
     Sci-Fi       0.700     0.074     0.135         94
      Short       0.875     0.245     0.383        200
   Thriller       0.562     0.068     0.122        264

avg / total       0.648     0.281     0.316       3653
```

```
In [17]: def random_forest(X_train, y_train):
             param_grid = {
                 'n_estimators': (10, 50),
                 'min_samples_leaf': (2, 3, 10),
                 'max_features': (0.2, 0.3),
             }
             model = RandomForestClassifier(random_state = 333, class_weight='balar
             model_tuning = GridSearchCV(
```

```
            model,
            param_grid=param_grid,
            scoring=make_scorer(hamming_loss, greater_is_better=False),
            cv=3,
            n_jobs=-1,
            verbose=3,
        )
        model_tuning.fit(X_train, y_train)
        return model_tuning
```

Random forest model based on full features data set

```
In [18]: forest_model = random_forest(X_train, y_train)

Fitting 3 folds for each of 12 candidates, totalling 36 fits


[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  2.6min
[Parallel(n_jobs=-1)]: Done  34 out of  36 | elapsed:  5.2min remaining:   18.3s
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  5.2min finished


In [19]: y_train_pred = forest_model.predict(X_train)
         y_test_pred = forest_model.predict(X_test)

         train_report = classification_report(y_train, y_train_pred, target_names=m
         test_report =  classification_report(y_test, y_test_pred, target_names=mlb

         forest_model_performance_df = report_to_df(test_report)

         print forest_model.best_params_
         print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
         print 'Train performance: '
         print train_report
         print 'Test performance: '
         print test_report
```

```
{'max_features': 0.3, 'n_estimators': 50, 'min_samples_leaf': 2}
hamming loss: 0.118
Train performance:
             precision    recall  f1-score   support

     Action      1.000     0.894     0.944       682
  Adventure      1.000     0.874     0.933       451
  Animation      0.989     0.952     0.970       373
     Comedy      0.995     0.807     0.891      1609
      Crime      0.998     0.906     0.950       561
Documentary      0.981     0.809     0.887       655
      Drama      0.997     0.832     0.907      2490
```

```
     Family      0.992     0.880     0.932       399
    Fantasy      1.000     0.916     0.956       311
     Horror      1.000     0.837     0.911       606
    Mystery      1.000     0.916     0.956       322
    Romance      0.997     0.832     0.907       796
     Sci-Fi      1.000     0.934     0.966       318
      Short      0.989     0.853     0.916       621
   Thriller      1.000     0.875     0.933       793

avg / total      0.996     0.854     0.919     10987

Test performance:
             precision    recall  f1-score   support

     Action      1.000     0.005     0.010       200
  Adventure      1.000     0.006     0.013       159
  Animation      0.821     0.158     0.264       146
     Comedy      0.857     0.103     0.184       525
      Crime      0.667     0.033     0.063       181
Documentary      0.880     0.291     0.437       227
      Drama      0.639     0.258     0.368       837
     Family      0.833     0.134     0.231       149
    Fantasy      0.000     0.000     0.000       105
     Horror      1.000     0.006     0.011       178
    Mystery      0.000     0.000     0.000        93
    Romance      0.889     0.027     0.053       295
     Sci-Fi      0.000     0.000     0.000        94
      Short      0.846     0.165     0.276       200
   Thriller      0.400     0.008     0.015       264

avg / total      0.718     0.118     0.183      3653



C:\Anaconda\lib\site-packages\sklearn\metrics\classification.py:1113: UndefinedMetr
  'precision', 'predicted', average, warn_for)
```

1 Detailed description and implementation of two different models

The evolution of our dataset: We initially wanted to run 2 models across two types of datasets. However, we saw it as unnecessarily confusing, computationally expensive and simpler if we combined the data. In its original form, one dataset consisted of a movies crew/director/actors and other production specific features. The other dataset was simply the overview (a description of the movie), within TMDB. These are different types of data so we had to ensure they were properly represented as features. Through vectorization, we were able to transform the overview into a feature set. From there, we combined it with a feature set consisting of production specific

feature groups detailed above. With our new dataset, we fit the chosen models and found different measures of accuracy. We were aware that given out multilabel classes, some models would work and some would need prior adjustment. We also considered the drastic class imbalance that occurs as some genres are much more common than others. Our models accounted for this, but we selected our performance metrics accordingly.

After analyzing many classification models and comparing baseline results, we have decided to use a stochastic gradient dissent linear support vector machine model and a random forest for this milestone. We chose and tuned these models for several reasons based on their performance, similarities and differences.

The stochastic gradient dissent linear support vector machine model aka SGD model: After testing out different support vector machine kernels and logistic regressions, we found that using an SGD learning mechanism on a linear support vector machine was computationally efficient and yielded good results. This model cannot natively handle multilabel classes so we were forced to use one vs rest classification (this fits a new model for each class). Through cross validation, we tuned the parameters according to a Hamming Loss. We were aware that this kind of model comparison can favor models that over classify the dominant class (especially with the amount of class imbalance in the dataset). So we made sure it weighted classes differently based on the observations falling under that class.

The Random Forest model: We wanted to utilize a model that could natively support multilabel classes (unlike our SGD) and creatively handle some of the more specific features in our dataset. We chose a random forest because of its performance. Through cross validation we tuned the minimum sample leaf and max features. We also accounted for class imbalance using the built in method in Sklearn. We found that our random forest underperformed compared to the SGD model, but still yielded important information on our dataset.

2 Description of your performance metrics

As we described in our last milestone, we will be using a multilabel classification approach. Some movies have one genres while most have multiple genres. This naturally creates a problem in classification as some observations fall under multiple classes. Further, making a class for every possible combination of genres is infeasible as it result in drastic class imbalances and under representation (the highly specific combinations will only have one or two corresponding movies). However, multilabel classification also has limitations, particularly when implementing and observing performance metrics. As we described previously, to run an SGD (or logistic regression, and other classifier models), we had to use an onevsrest classifier. A random forest can handle multilabel classification natively and we wanted to express the pros and cons of each model.

There are also performance considerations. One vs the rest method fits as many models as there are labels in the dataset. First we tried to use traditional linear support vector classifier but having over 20 labels even on 3K dataset the grid search usually takes hours on our hardware. That is why we decided to use SGD based classifier with the appropriate loss function ('hinge' for SVC or 'log' for logistic regression). This proved to be better solution easily scaling to handle 10K dataset with lots of features. We also wanted to evaluate a random forest classifier because it can handle multilabel classification natively. We also discovered that random forest scales quite good and can easily handle our 10K dataset.

To analyze the models, we decided to use 3 accuracy/error based measures. Using the classification_report function in sklearn, we returned precision, recall and the f-1 score of our models across genres. In this context, precision is the fraction of correct results divided by the number of

all returned results while recall is the number of correct results divided by the number of results that should have been returned. Finally, the f1-score is a measure of the model's accuracy, it considers both precision and recall to compute a score. We decided to use the f-1 score to compare the accuracy across models; we did this for several reasons, mainly because of massive class imbalance, we describe below. We considered using a roc or auc curve, but it is traditionally used in binary classification. It can adapt to the multilabel classification methodology, but it would be necessary to binarize the output of the model. Lastly, we applied the same performance metrics to the two used datasets (crew/actor/director dataset vs overview dataset). Cross comparing accuracy across datasets isn't as helpful as comparing accuracy within a dataset across models. The f1-score allows us to analyze the performance of our models which both use very different algorithms. Our visualizations reflect the differences in performance between models across both datasets.

3 Careful performance evaluations for both models

Baseline classifier (dummy classifier) and metrics To further check our models, we implemented three dummy classifiers. The stratified classifier is a classification algorithm that takes the training set class distribution and assigns classes to the test set based on that distribution. The uniform dummy classifier assigns classes in uniform to all observations in the testing set. Finally, the most frequent dummy classifier assigns the most frequent class to every observation. However, in this context, our classes are multilabels. We compared accuracy across genres of both our SGD and Random Forest model against each other and the dummy classifiers. Choosing the right metric is a challenge within itself as some may be misleading while others are not relevant.

```
In [20]: def evaluate_baseline(X_train, X_test, y_train, y_test, strategy):
             num_y_columns = y_train.shape[1]

             y_test_pred_list = []
             for i in xrange(num_y_columns):
                 one_y_train = y_train[:, i]
                 model = DummyClassifier(strategy=strategy)
                 model.fit(X_train, one_y_train)
                 one_y_test_pred = model.predict(X_test)
                 y_test_pred_list.append(one_y_test_pred)

             y_test_pred = np.array(y_test_pred_list).T

             return y_test_pred
```

Stratified dummy classifier

```
In [21]: y_test_pred = evaluate_baseline(X_train, X_test, y_train, y_test, 'stratif

         test_report = classification_report(y_test, y_test_pred, target_names = ml
         stratified_model_performance_df = report_to_df(test_report)

         print 'accuracy score: %.3f' % accuracy_score(y_test, y_test_pred)
         print 'jaccard similarity score: %.3f' % jaccard_similarity_score(y_test,
```

```
        print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
        print 'zero one loss: %.3f' % zero_one_loss(y_test, y_test_pred)
        print test_report

accuracy score: 0.026
jaccard similarity score: 0.128
hamming loss: 0.203
zero one loss: 0.974
             precision    recall  f1-score   support

     Action       0.099     0.120     0.108       200
  Adventure       0.052     0.050     0.051       159
  Animation       0.115     0.075     0.091       146
     Comedy       0.294     0.290     0.292       525
      Crime       0.130     0.138     0.134       181
Documentary       0.081     0.084     0.082       227
      Drama       0.426     0.424     0.425       837
     Family       0.069     0.060     0.065       149
    Fantasy       0.055     0.048     0.051       105
     Horror       0.102     0.118     0.110       178
    Mystery       0.023     0.032     0.027        93
    Romance       0.140     0.132     0.136       295
     Sci-Fi       0.031     0.032     0.031        94
      Short       0.092     0.095     0.094       200
   Thriller       0.150     0.144     0.147       264

avg / total       0.202     0.200     0.201      3653
```

Uniform dummy classifier

```
In [22]: y_test_pred = evaluate_baseline(X_train, X_test, y_train, y_test, 'uniform

         test_report = classification_report(y_test, y_test_pred, target_names = ml
         uniform_model_performance_df = report_to_df(test_report)

         print 'accuracy score: %.3f' % accuracy_score(y_test, y_test_pred)
         print 'jaccard similarity score: %.3f' % jaccard_similarity_score(y_test,
         print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
         print 'zero one loss: %.3f' % zero_one_loss(y_test, y_test_pred)
         print test_report

accuracy score: 0.000
jaccard similarity score: 0.114
hamming loss: 0.501
zero one loss: 1.000
             precision    recall  f1-score   support
```

```
     Action       0.103     0.515     0.171       200
  Adventure       0.093     0.572     0.160       159
  Animation       0.081     0.534     0.140       146
     Comedy       0.270     0.488     0.347       525
      Crime       0.100     0.541     0.169       181
Documentary       0.111     0.463     0.178       227
      Drama       0.442     0.497     0.468       837
     Family       0.079     0.530     0.138       149
    Fantasy       0.055     0.495     0.099       105
     Horror       0.096     0.506     0.161       178
    Mystery       0.047     0.484     0.086        93
    Romance       0.163     0.553     0.252       295
     Sci-Fi       0.045     0.457     0.082        94
      Short       0.090     0.430     0.149       200
   Thriller       0.138     0.481     0.215       264

avg / total       0.205     0.502     0.263      3653
```

Most frequent dummy classifier

```
In [23]: y_test_pred = evaluate_baseline(X_train, X_test, y_train, y_test, 'most_fr

         test_report = classification_report(y_test, y_test_pred, target_names = ml
         most_frequent_model_performance_df = report_to_df(test_report)

         print 'accuracy score: %.3f' % accuracy_score(y_test, y_test_pred)
         print 'jaccard similarity score: %.3f' % jaccard_similarity_score(y_test,
         print 'hamming loss: %.3f' % hamming_loss(y_test, y_test_pred)
         print 'zero one loss: %.3f' % zero_one_loss(y_test, y_test_pred)
         print test_report
```

```
accuracy score: 0.029
jaccard similarity score: 0.029
hamming loss: 0.127
zero one loss: 0.971
             precision    recall  f1-score   support

     Action       0.000     0.000     0.000       200
  Adventure       0.000     0.000     0.000       159
  Animation       0.000     0.000     0.000       146
     Comedy       0.000     0.000     0.000       525
      Crime       0.000     0.000     0.000       181
Documentary       0.000     0.000     0.000       227
      Drama       0.000     0.000     0.000       837
     Family       0.000     0.000     0.000       149
```

```
     Fantasy       0.000     0.000     0.000       105
      Horror       0.000     0.000     0.000       178
     Mystery       0.000     0.000     0.000        93
     Romance       0.000     0.000     0.000       295
      Sci-Fi       0.000     0.000     0.000        94
       Short       0.000     0.000     0.000       200
    Thriller       0.000     0.000     0.000       264

avg / total       0.000     0.000     0.000      3653
```

Hamming Loss is often used for classification accuracy, but as we stated before, the dataset has massive class imbalance. This can result in the model over assigning observations to the majority class, simply because it will be more accurate on the whole dataset if it does so. However, a model like this has very poor applicability as it doesn't give us real information as to which observations belong to which class. The SGD and Random Forest can take arguments in Sklearn noting that there may class imbalance. We also tuned our model with relevant parameters to avoid class imbalance. As one can see in our most frequent dummy classifier (which has the highest Hamming Loss) simple measures of accuracy can be deceitful as to the overall efficacy of a model.

```
In [24]: print 'Random forest model run on full features dataset'
         print forest_model_performance_df
         print 'SGD model run on full features dataset'
         print sgd_model_performance_df
```

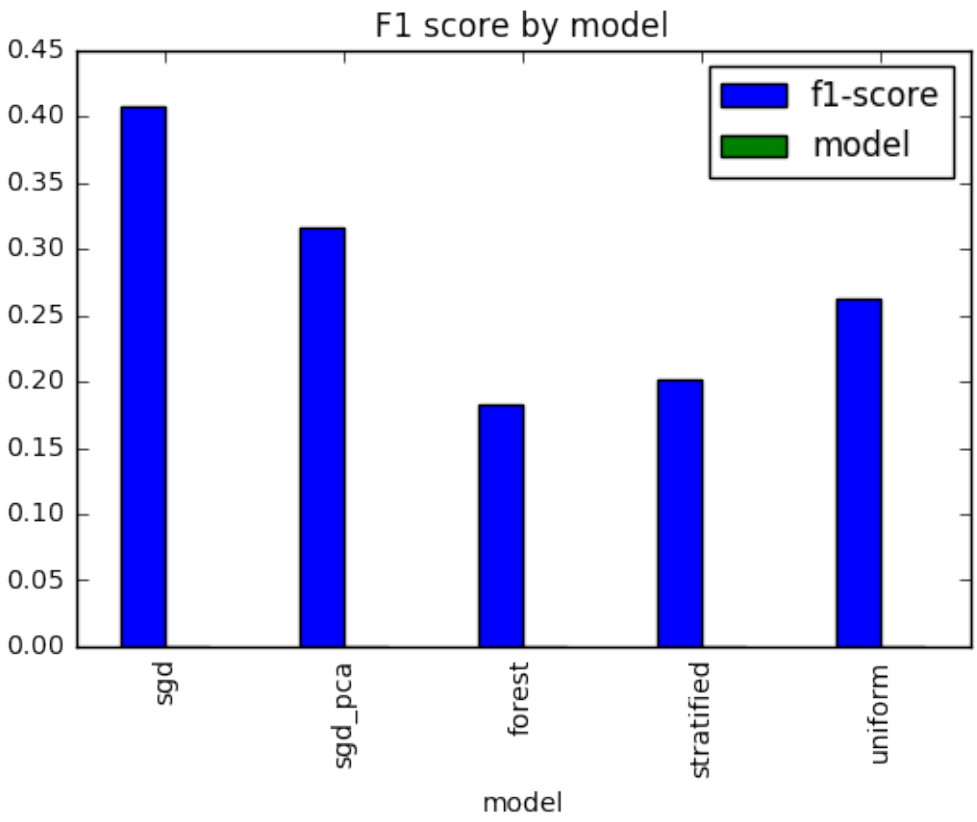
```
Random forest model run on full features dataset
             precision  recall  f1-score  support
Action           1.000   0.005     0.010      200
Adventure        1.000   0.006     0.013      159
Animation        0.821   0.158     0.264      146
Comedy           0.857   0.103     0.184      525
Crime            0.667   0.033     0.063      181
Documentary      0.880   0.291     0.437      227
Drama            0.639   0.258     0.368      837
Family           0.833   0.134     0.231      149
Fantasy          0.000   0.000     0.000      105
Horror           1.000   0.006     0.011      178
Mystery          0.000   0.000     0.000       93
Romance          0.889   0.027     0.053      295
Sci-Fi           0.000   0.000     0.000       94
Short            0.846   0.165     0.276      200
Thriller         0.400   0.008     0.015      264
Total/avg        0.718   0.118     0.183     3653
SGD model run on full features dataset
             precision  recall  f1-score  support
Action           0.625   0.150     0.242      200
Adventure        0.708   0.107     0.186      159
```

```
Animation        0.930   0.452     0.608      146
Comedy           0.795   0.229     0.355      525
Crime            0.438   0.077     0.131      181
Documentary      0.506   0.938     0.657      227
Drama            0.698   0.602     0.647      837
Family           0.820   0.336     0.476      149
Fantasy          0.348   0.076     0.125      105
Horror           0.652   0.253     0.364      178
Mystery          0.333   0.065     0.108       93
Romance          0.563   0.136     0.219      295
Sci-Fi           0.500   0.128     0.203       94
Short            0.526   0.655     0.584      200
Thriller         0.528   0.144     0.226      264
Total/avg        0.638   0.354     0.407     3653
```

As one can see from the f1 scores for each sub category, our SGD model generally outperformed our random forest model. It is worth noting that some of the f1-scores of the random forest dataset are 0 (precision and recall are as well for those genres). This is because some of the under-represented classes are being phased out of the classification model given their lack of impact in creating the random forest. In a sense, they model is ignorant to them while it still classifies drama (the most used genre) incredibly well (better than the SGD). This is a natural weakness of class imbalance that we try to account for, while preventing our model from overfitting the training set in its entire detail.

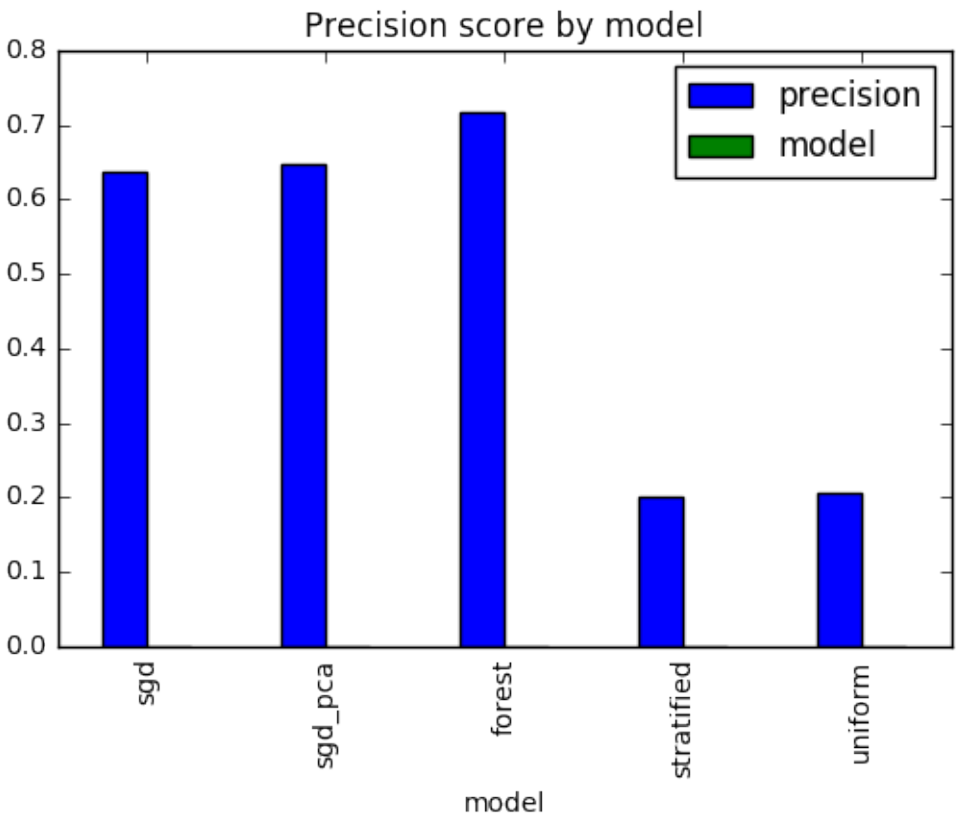
4 Visualizations of the metrics for performance evaluation

```
In [25]: # combine performance data into a single dataframe
         sgd_model_performance_df['model'] = 'sgd'
         sgd_pca_model_performance_df['model'] = 'sgd_pca'
         forest_model_performance_df['model'] = 'forest'
         stratified_model_performance_df['model'] = 'stratified'
         uniform_model_performance_df['model'] = 'uniform'

         combined_performance_df = result = pd.concat(
             [
             sgd_model_performance_df,
             sgd_pca_model_performance_df,
             forest_model_performance_df,
             stratified_model_performance_df,
             uniform_model_performance_df
             ]
         )
         combined_performance_df.set_index('model', append=True, inplace=True)

In [26]: combined_performance_df.loc['Total/avg', ['f1-score', 'model']].plot(kind=
         plt.show()
```

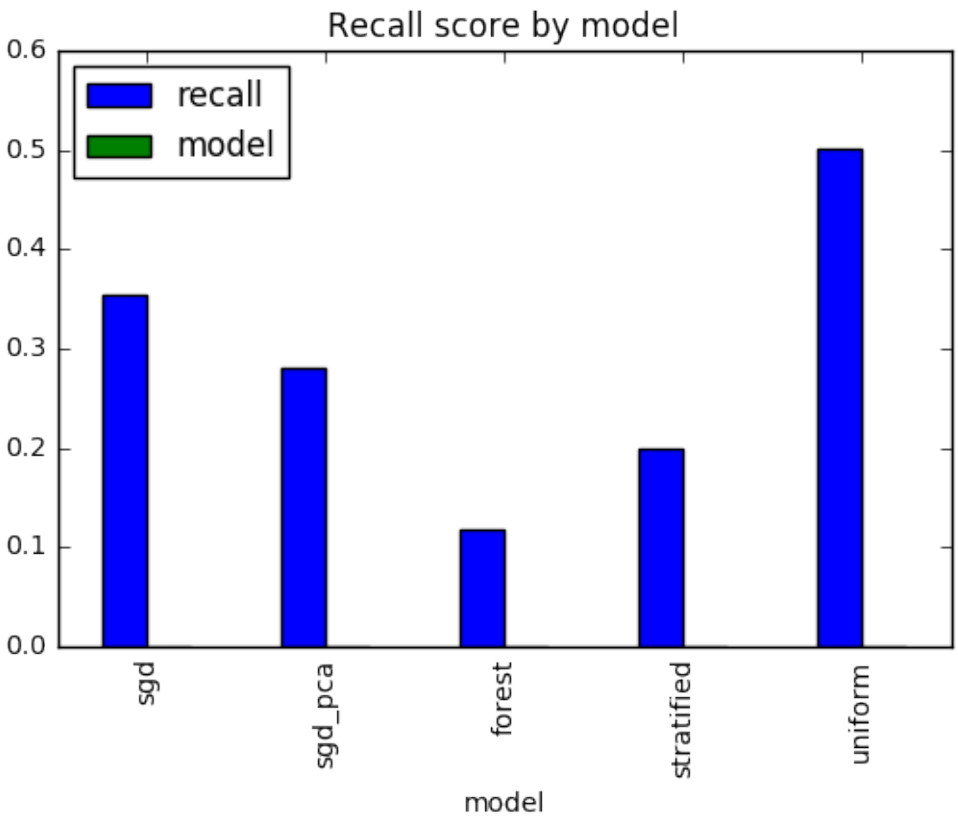


SGD model based on the features reduced using PCA decomposition does not perfrom better than the SGD model based on full features data set.

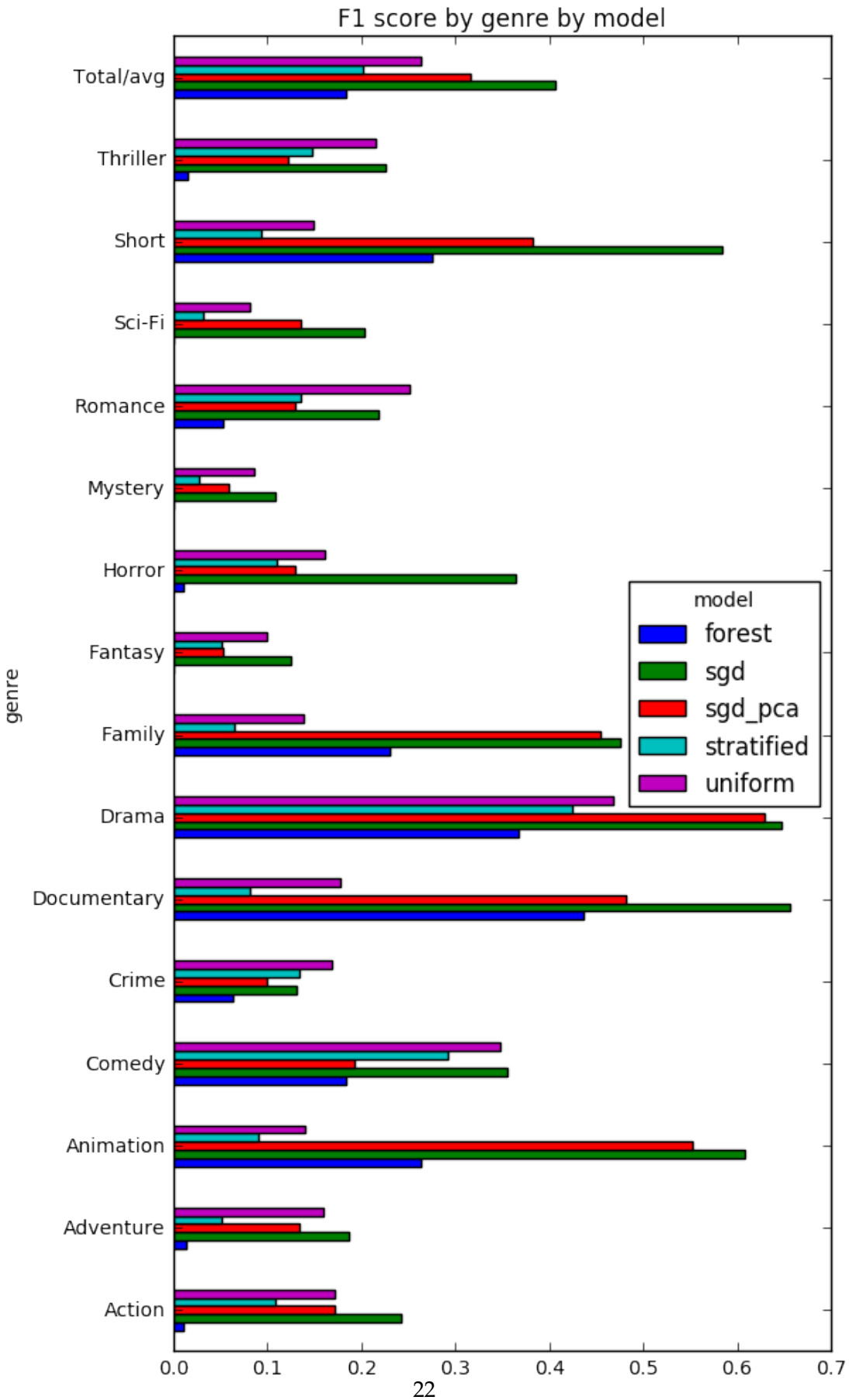
```
In [27]: combined_performance_df.loc['Total/avg', ['precision', 'model']].plot(kind
         plt.show()
```




```
In [28]: combined_performance_df.loc['Total/avg', ['recall', 'model']].plot(kind="b
         plt.show()
```



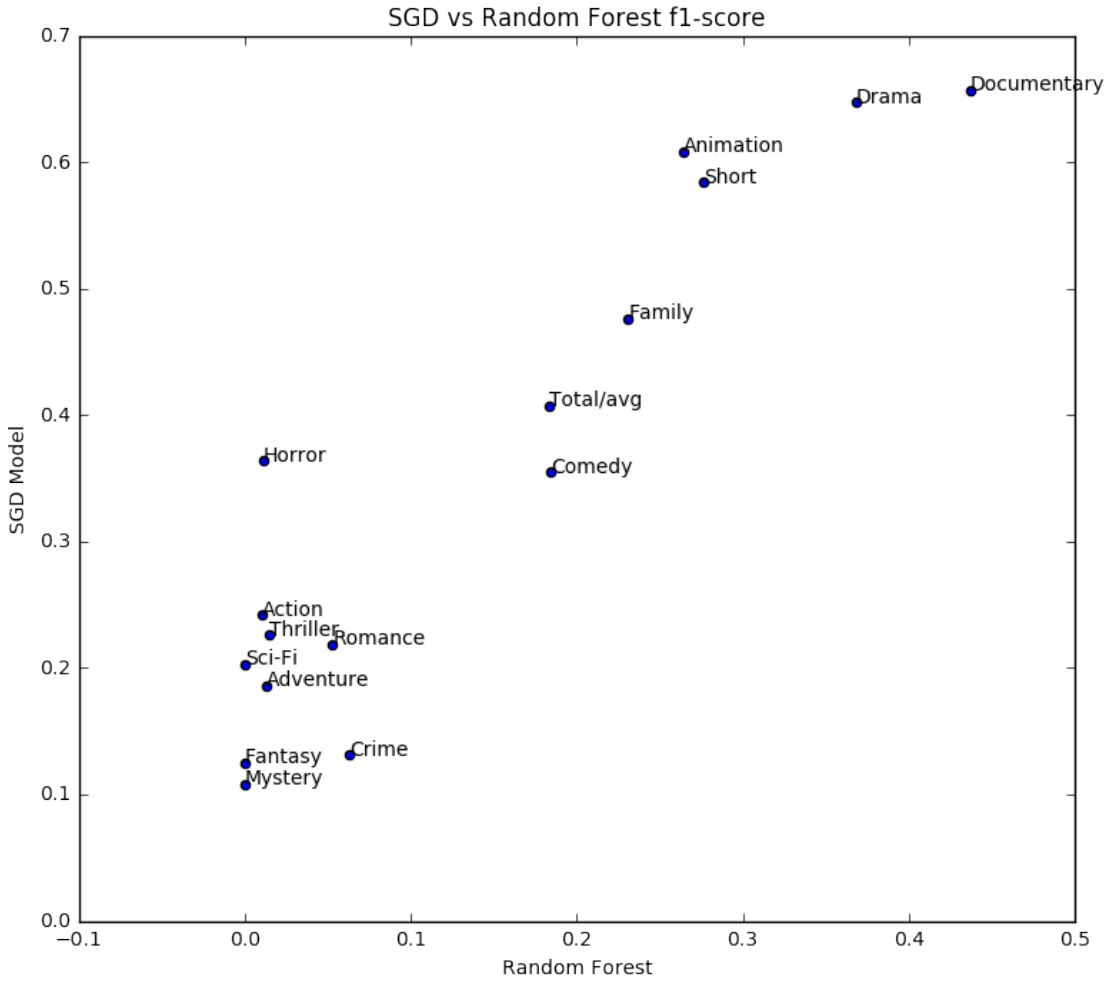

```
In [29]: combined_performance_df['genre'] = combined_performance_df.index.get_level
         combined_performance_df['model'] = combined_performance_df.index.get_level
         by_genre_df = combined_performance_df.pivot(index='genre', columns='model'
         by_genre_df.reset_index()
         by_genre_df.plot.barh(figsize = (6, 12), title = 'F1 score by genre by mod
         plt.show()
```


F1 score by genre by model
Total/avg
Thriller
Short
Sci-Fi
Romance
Mystery
Horror
Fantasy
Family
Drama
Documentary
Crime
Comedy
Animation
Adventure
Action
genre
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
model
forest
sgd
sgd_pca
stratified
uniform

The SGD classifer outperformes the random forest classifier for every label.

4.0.1 Models based on crew/director/feature set data

```
In [49]: x = forest_model_performance_df['f1-score']
         y = sgd_model_performance_df['f1-score']
         l = forest_model_performance_df.index

         fig, ax = plt.subplots(figsize = (9, 8))
         ax.scatter(x, y)

         for i, txt in enumerate(l):
             ax.annotate(txt, (x[i],y[i]))

         ax.set_ylabel('SGD Model')
         ax.set_xlabel('Random Forest')
         ax.set_title('SGD vs Random Forest f1-score')
         plt.show()
```



We decided to use the f1-score as our main source of comparing models. It shows us a more complete representation of a models accuracy and relevance than just precision or recall. In the above visualizations, each point in the graph is a genre. The axis consists of each model. We can clearly see that some genres are just harder to classify than others, there is a sizeable cluster of lower performing genres (in terms of f1-score). But from there, using the axis scale, we can also see that the SGD model generally classifies more genres better than a Random Forest model (with our current parameters).

5 Discussion of the differences between the models, their strengths, weaknesses, etc.

Given the inherent difficulties of multilabel classification and dealing with an imbalanced dataset, the models we select will also have some limitations. One of the biggest limitations of our SGD model is its inability to natively classify multilabel classes. As we described earlier, we had to use a one vs rest classification that fits as many models as we have labels. The SGD model is defined as "Linear classifiers (SVM, logistic regression, a.o.) with SGD training." in Sklearn. This means

weaknesses in an SVM may show in an SGD model. Finally, because of the size of our dataset and its complexity, we opted to only use a linear kernel when originally toying with an SVM. It greatly improves runtime and requires less tuning parameters but may sacrifice predictive ability. Our observations on our linear SVM drove us to implement the SGD.

The SGD model does have pros to its cons. It seems to be fairly accurate and does a good job, when accounting for imbalanced class data, at finding movies in each genre. Its computation time is extremely quick and with the proper tuning it can become a reliable model. Further, it is based on a methodology that can be applied to logistic regression or support vector machines. It inherits the robustness of these model algorithms (in this case the Linear SVM).

The Random Forest model clearly has limitations in accuracy and computation time. While it is fast to train, making a highly accurate forest model requires more trees which will greatly slow down computation. Given the intricate dataset we have, a highly trained forest may be extremely accurate but infeasible if computation time is too long. Our dataset has many labels some underrepresented and finding those little groves that define a multilabel that has one or two observations will be extremely difficult. Further, if we were to create a highly specific model, it will be rigid and unmalleable when trying to adapt to different datasets about movies even with the same features (over-fitted). We cross validated when choosing parameters to account for this but it may lead to some inaccuracy on the final testing dataset (because a lack of specificity).

The random forest also has some pros. First off, it natively supports multilabel classes. This is huge as it takes out the need to use one vs rest classifier. Secondly, I believe that if we had a sub sampled dataset with relatively no class imbalance, it would perform very well. Here it could account for the specific groves and curves through a high number of trees while still ensuring that the less represented classes can be classified.

6 Discussion of the performances you achieved, and how you might be able to improve them in the future

As described at the beginning of our document, our models are the creation of analysis on other similar models used on a dataset that has been cut and combined. While we believe we have taken the best steps to finding a good classification model, able to predict genre, there are other avenues we want to explore. First, when combining datasets based on features, we believe our feature set may have extensively increased. This could benefit from some form of dimensionality reduction like PCA. However, we will have to consider this effect on textual data from movie overviews.

We have also experimented with color palette see if the posters color palette convey any genre information. It might make some sense considering the poster artists likely to use some similar color tricks to channel genre information to the potential audience. This process is computationally expensive and creating a feature set out of the color palette will take a while. We hope to analyze this as a possible 3rd major feature predictive set.

Lastly, although we tuned based on appropriate loss parameters, we hope to tune our models further. We especially want to investigate the pros and cons of manipulating our forest's trees more. We are aware of the computational limitations but would look for a work around. Given the impressive performance of our SGD model, we may want to apply the same stochastic learning process to a logistic regression or alternate support vector machine with a degree or radial kernel (if computation allows). We have also begun experimenting with different metrics to compare models when cross validating. There may be a better metric than Hamming Loss when finding the best models given a set of parameters. This will, hopefully, help us in finding the most accurate and most applicable model.