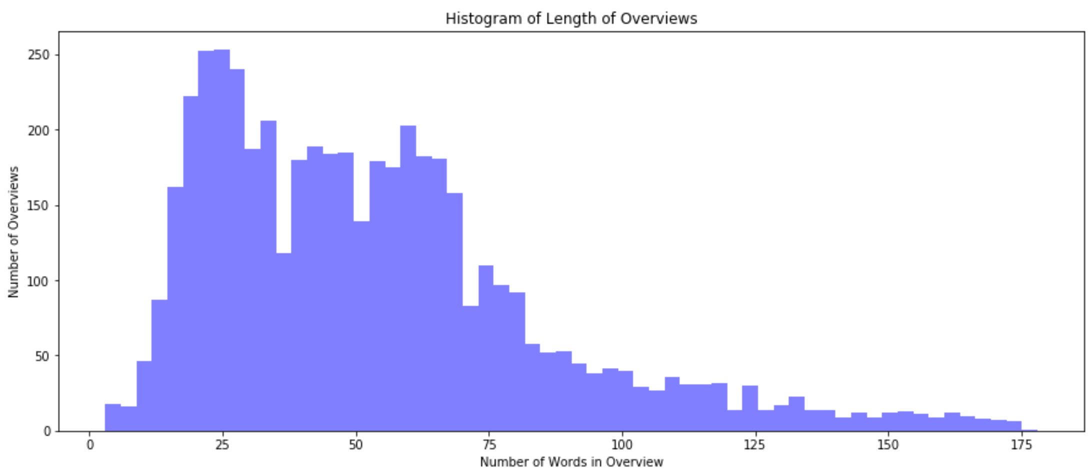
Part 1 Processing Text Predictors (Title and Overview)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

#Load the data
df = pd.read_csv('tmdb_metadata.csv', delimiter=',')
```

In [2]:
```
# Analysis of length of overview
# #Break each posts into words and count the number of words
lengths = df['overview'].apply(lambda x: len(x.split(' ')))

print 'range of overview lengths:', np.max(lengths), '-', np.min(lengths)
print 'mean of overview lengths:', np.mean(lengths)

fig, ax = plt.subplots(1, 1, figsize=(15, 6))

#Histogram of the word counts in each post
ax.hist(lengths, color='blue', bins=60, alpha=0.5)

ax.set_xlabel('Number of Words in Overview')
ax.set_ylabel('Number of Overviews')
ax.set_title('Histogram of Length of Overviews')

plt.show()
```

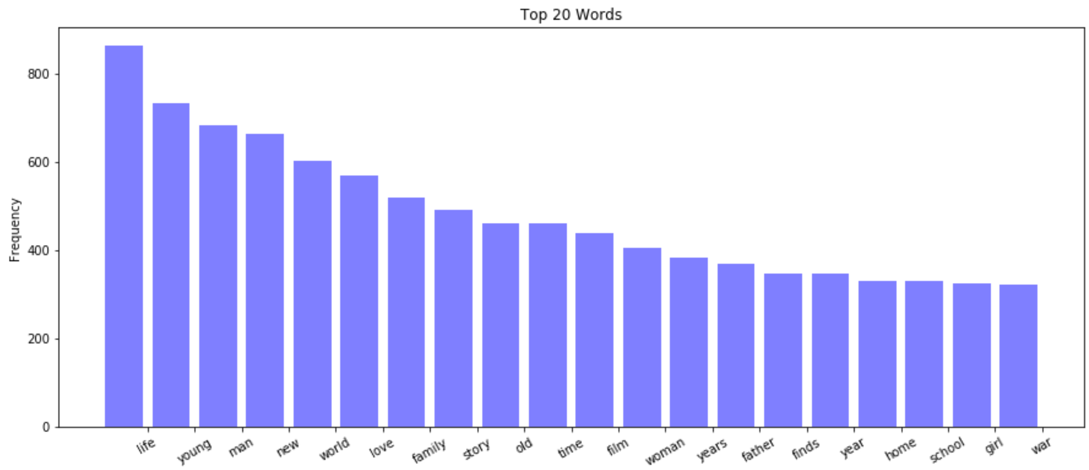
```
range of overview lengths: 178 - 3
mean of overview lengths: 53.8396572827
```



In [3]:
```
# boolean for including titles in text analysis
include_titles = 1

#Create a text vectorizer (turns text into array of numbers)
#using a common list of English stop words
vectorizer = CountVectorizer(stop_words='english', min_df=1)

### Top words over all genres
#Get all the text from data
corpus = df['overview'].values

if include_titles:
    titles = df['title'].values
    corpus = np.concatenate([corpus,titles])

#Turn each text into an array of word counts
x = vectorizer.fit_transform(corpus)
x = x.toarray()

#Get the names of all the words we're counting
feature_names = vectorizer.get_feature_names()

print 'data shape:', x.shape
print 'some features:', feature_names[0:10]

#Number of top words
n = 20

#Count the number of time each word occurs in the entire dataset
word_freq = x.sum(axis=0)

#Sort the words by their total frequency in the dataset
words = zip(word_freq, feature_names)
top_words = (sorted(words, key=lambda t: t[0], reverse=True))[:n]

#Print the top n words and their frequencies
print top_words
```

```
data shape: (9804, 24041)
some features: [u'00', u'000', u'000th', u'007', u'009', u'01', u'03', u'05
pm', u'10', u'100']
[(862, u'life'), (732, u'young'), (682, u'man'), (664, u'new'), (602, u'wor
ld'), (570, u'love'), (518, u'family'), (490, u'story'), (462, u'old'), (46
2, u'time'), (438, u'film'), (405, u'woman'), (383, u'years'), (368, u'fath
er'), (348, u'finds'), (348, u'year'), (331, u'home'), (331, u'school'), (3
25, u'girl'), (321, u'war')]
```

In [4]:
```
fig, ax = plt.subplots(1, 1, figsize=(15, 6))

#Number of bars to use
indices = np.arange(n)
#Where to put the label under each bar
width = 0.5
#Bar plot of the frequencies of the top words
ax.bar(indices, [word[0] for word in top_words], color='blue', alpha=0.5)

ax.set_ylabel('Frequency')
ax.set_title('Top ' + str(n) + ' Words')

#Label the bars with the top words
ax.set_xticks(indices + width)
ax.set_xticklabels([word[1] for word in top_words])

#Turn the labels sideways so they don't overlap
labels = ax.get_xticklabels()
plt.setp(labels, rotation=30, fontsize=10)

plt.show()
```



In [8]:

```
### Top words in each genre
# contains top words of each genre, index is the same as the labels column
in Jason's dataset
genre_top_words_counts = []
genre_top_words = []

def getTopWordsOfGenre(genre):
    #get movies with this genre
    key = 3*genre + 1
    b = df['labels'].str.get(key) == '1'
    df2 = df[b]

    #Number of top words per genre
    n_top = 100

    #Get all the text from data
    corpus2 = df2['overview'].values
    if include_titles:
        titles = df2['title'].values
        corpus2 = np.concatenate([corpus2,titles])

    #Turn each text into an array of word counts
    x2 = vectorizer.fit_transform(corpus2)
    x2 = x2.toarray()

    #Get the names of all the words we're counting
    feature_names2 = vectorizer.get_feature_names()

    #Count the number of time each word occurs in the entire dataset
    word_freq2 = x2.sum(axis=0)

    #Sort the words by their total frequency in the dataset
    words2 = zip(word_freq2, feature_names2)
    top = (sorted(words2, key=lambda t: t[0], reverse=True))[:n_top]
    top_array = np.asarray(zip(*top)[1])
    genre_top_words_counts.append(top)
    genre_top_words.append(top_array)

n_genres = 19
for i in range(n_genres):
    getTopWordsOfGenre(i)
```

In [9]:
```
### creating dataframe for each movie and genre words frequency
genre_top_words_flattened = np.array(genre_top_words).flatten()
genre_top_words_flattened = np.unique(genre_top_words_flattened)

def count_freq(overview):
    #Turn each text into an array of word counts
    x3 = vectorizer.fit_transform(overview)
    x3 = x3.toarray()

    #Get the names of all the words we're counting
    feature_names3 = vectorizer.get_feature_names()

    #Count the number of time each word occurs in the entire dataset
    word_freq3 = x3.sum(axis=0)
    words3 = zip(word_freq3, feature_names3)
    df_freq = pd.DataFrame(words3, columns=['Freq', 'Feature'])

    counts = []
    for i in range(len(genre_top_words_flattened)):
        word_count = df_freq[df_freq['Feature'] == genre_top_words_flattene
d[i]]
        if word_count.empty:
            counts.append(0)
        else:
            val = df_freq[df_freq['Feature'] == genre_top_words_flattened[i
]]['Freq'].iloc[0]
            counts.append(val)
    return counts

movie_words_freq = []
for i in range(len(df)):
    corpus3= df['overview'][i:(i+1)].values
    if include_titles:
        titles = df['title'][i:(i+1)].values
        corpus3= np.concatenate([corpus3,titles])
    a = count_freq(corpus3)
    movie_words_freq.append(a)
```

In [10]:
```
df_genre_word = pd.DataFrame(movie_words_freq, columns=genre_top_words_flat
tened)
df_genre_word.to_csv('genre_words.csv', encoding = 'utf-8')
```

In [38]:
```
from sklearn.decomposition import PCA
x_text = df_genre_word.values
pca = PCA(n_components = 300)
pca.fit(x_text)
text_pca = pca.transform(x_text)
```

In [44]:
```
print "explained variance: " + str(sum(pca.explained_variance_ratio_))
df_genre_word_pca = pd.DataFrame(text_pca)
df_genre_word_pca.to_csv('genre_words_pca.csv', encoding = 'utf-8')
```

```
explained variance: 0.903391526261
```

Milestone3

April 19, 2017

1 Part 2 Multi-label Classification Using Random Forest

In [3]:
```
# import data from milestone 2
import ast
# contains info about 5060 movies
movie_df = pd.read_csv('dataset1.csv')

# to locate buffer overflow error
# import csv
# with open(r"dataset1.csv", 'rb') as f:
#     reader = csv.reader(f)
#     linenumber = 1
#     try:
#         for row in reader:
#             linenumber += 1
#     except Exception as e:
#         print (("Error line %d: %s %s" % (linenumber, str(type(e)), e.mes

movie_df = movie_df.drop('Unnamed: 0', axis=1)
movie_df = movie_df.dropna()
labels = []

# convert 19 genre ids into a label matrix so that
# for each movie, the genre is a 1*19 matrix, 1 meaning it's in the genre
for i in movie_df.genre_ids:
    label_matrix = np.zeros(len(genre_lst.keys()), dtype=int)
    for j in ast.literal_eval(i):
        if j in genre_lst.keys():
            label_matrix[genre_lst.keys().index(j)] = 1
    labels.append(label_matrix)
movie_df['labels'] = labels
len(movie_df)
movie_df.head()
```

Out[3]:
```
                  genre_ids  movie_id  \
0           [14, 10402, 10749]    321612
1                [28, 18, 878]    263115
```

```
2  [16, 35, 18, 10751, 10402]    335797
3                  [28, 12, 14]    293167
4                  [28, 80, 53]    337339

                                          overview  popularity  \
0  A live-action adaptation of Disney's version o...  149.542760
1  In the near future, a weary Logan cares for an...   79.627847
2  A koala named Buster recruits his best friend ...   77.930498
3  Explore the mysterious and dangerous home of t...   61.012215
4  When a mysterious woman seduces Dom into the w...   60.623332

                        poster_path release_date                  title
0  /tWqifoYuwLETmmasnGHO7xBjEtt.jpg   2017-03-16     Beauty and the Beast
1  /45Y1G5FEgttPAwjTYic6czC9xCn.jpg   2017-02-28                    Logan
2  /s9ye87pvq2IaDvjv9x4IOXVjvA7.jpg   2016-11-23                     Sing
3  /5wBbdNb0NdGiZQJYoKHRv6VbiOr.jpg   2017-03-08       Kong: Skull Island
4  /iNpz2DgTsTMPaDRZq2tnbqjL2vF.jpg   2017-04-12  The Fate of the Furious

   vote_average  vote_count                                              lab
0           6.9        1770  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1           7.5        2429  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
2           6.7        1170  [0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
3           6.0        1203  [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
4           7.2         482  [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
```

In [4]:
```
import datetime

# function that converts dates to ints
def to_integer(dt_time):
    return 10000*dt_time.year + 100*dt_time.month + dt_time.day

# drop rows with malformed release date information
malformed_index = []
for i in range(0, len(movie_df)):
    f = movie_df.iloc[i].release_date.split('-')
    if len(f) != 3:
        malformed_index.append(i)
movie_df = movie_df.drop(movie_df.index[malformed_index])

# new dataset length
print 'After deleting malformed dates, the data size is', len(movie_df)

int_dates = []
# convert datetime into integers
for i in range(0, len(movie_df)):
    f = movie_df.iloc[i].release_date.split('-')
    a = datetime.date(int(f[0]), int(f[1]), int(f[2]))
    int_dates.append(to_integer(a))
```

```
movie_df['int_dates'] = int_dates

# final sample size
print 'The final sample size is', len(movie_df)

# write to csv
movie_df.to_csv('tmdb_metadata.csv', encoding = 'utf-8')
```

```
After deleting malformed dates, the data size is 4902
The final sample size is 4902
```

In the following cell, we combine results from text processing with with the numerical predictors. Please refer to TextVectorizing.ipynb to see how we generated genre_words_pca.

```
In [5]: # import dataset
        metadata = pd.read_csv('tmdb_metadata.csv')

        # import results of text processing after pca, 90% variance explained
        df_a = pd.read_csv('genre_words_pca.csv')

        # select other predictors
        df_b = metadata[['popularity','int_dates','vote_average','vote_count']]

        # create dataframe
        treedata_X = pd.concat([df_a, df_b], axis=1)

        treedata_y = np.asarray(metadata['labels'].tolist())

        # examine the data
        treedata_X[25:30]
```

```
Out[5]:     Unnamed: 0         0         1         2         3         4         5
        25          25  0.368577  0.840917  0.272582 -0.435428 -0.135311  0.059444
        26          26  0.764552 -1.377594 -0.142785 -0.615930 -0.332792  0.233683
        27          27  1.553418 -1.984979 -0.104861 -0.828822 -0.502757  0.166974
        28          28  0.358079  0.923268  1.380883 -0.058584 -0.866399 -0.163135
        29          29  0.088955  0.560351  1.049459  0.146033 -0.171834 -0.101794

                   6         7         8     ...          294       295       296
        25 -0.251731 -0.171425  0.123310     ...    -0.189341 -0.063305 -0.401295
        26 -0.465798  0.304202 -0.122500     ...     0.031852 -0.009340 -0.000459
        27 -0.700010  0.355295 -0.188343     ...     0.084633  0.049862 -0.062283
        28  0.157153  0.391488 -0.266951     ...     0.003047  0.010992  0.047739
        29 -0.307366  0.142916 -0.346554     ...    -0.045943 -0.022995 -0.022300

                 297       298       299  popularity  int_dates  vote_average  \
        25 -0.198236  0.209343 -0.243798    1.321019   20051020           5.3
```

```
26  0.019383  0.012910  0.034607    1.321017   19731006           6.2
27  0.013205 -0.050039 -0.131973    1.320991   19871225           6.7
28 -0.007493 -0.041105  0.013070    1.320988   20170407           0.0
29 -0.087274  0.019076  0.031588    1.320942   20150313           3.2

    vote_count
25           2
26          12
27          29
28           0
29           3

[5 rows x 305 columns]
```

This dataset now has over 300 predictors and 4900 entries. We conduct pca on the data to enable faster tuning.

```
In [6]: # create tree_data_pca
        from sklearn.decomposition import PCA
        from sklearn import preprocessing

        tree_x = treedata_X.values
        tree_x_scaled = preprocessing.scale(tree_x)
        pca = PCA(n_components = 273)
        pca.fit(tree_x_scaled)
        tree_x_pca = pca.transform(tree_x_scaled)
        print "explained variance: " + str(sum(pca.explained_variance_ratio_))

explained variance: 0.901185187595
```

In this section, we first try a decision tree classifier to establish the baseline, then tunes a random forest classifier to improve the prediction accuracy. Meanwhile, we also develop a different matric to measure prediction accuracy, which we are calling "mean match ratio."

```
In [8]: # establish baseline decision tree accuracy
        # http://scikit-learn.org/stable/modules/tree.html#tree 1.10.3. Multi-outpu
        print(__doc__)
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.cross_validation import train_test_split

        # train test split
        X_train, X_test, y_train, y_test = train_test_split(treedata_X, treedata_y,

        # fit model
        class_1 = DecisionTreeClassifier()
        class_1.fit(X_train, y_train)
```

```
        # predict
        pred_train = class_1.predict(X_train)
        pred_test = class_1.predict(X_test)

        # metric 1 - complete match percentage via .score
        print 'The proportion of complete matches in training set is', class_1.scor
        print 'The proportion of complete matches in test set is', class_1.score(X_
```

```
Automatically created module for IPython interactive environment
The proportion of complete matches in training set is 1.0
The proportion of complete matches in test set is 0.0331463539011
```

Above are the baseline results from decision tree with 305 predictors.

```
In [12]: # with pca
         # train test split
         X_pca_train, X_pca_test, y_train, y_test = train_test_split(tree_x_pca, tr

         # fit model
         class_pca = DecisionTreeClassifier()
         class_pca.fit(X_pca_train, y_train)

         # predict
         pred_train_pca = class_pca.predict(X_pca_train)
         pred_test_pca = class_pca.predict(X_pca_test)

         # metric 1 - complete match percentage via .score
         print 'The proportion of complete matches in training set is', class_pca.s
         print 'The proportion of complete matches in test set is', class_pca.score
```

```
 The proportion of complete matches in training set is 1.0
The proportion of complete matches in test set is 0.0280469148394
```

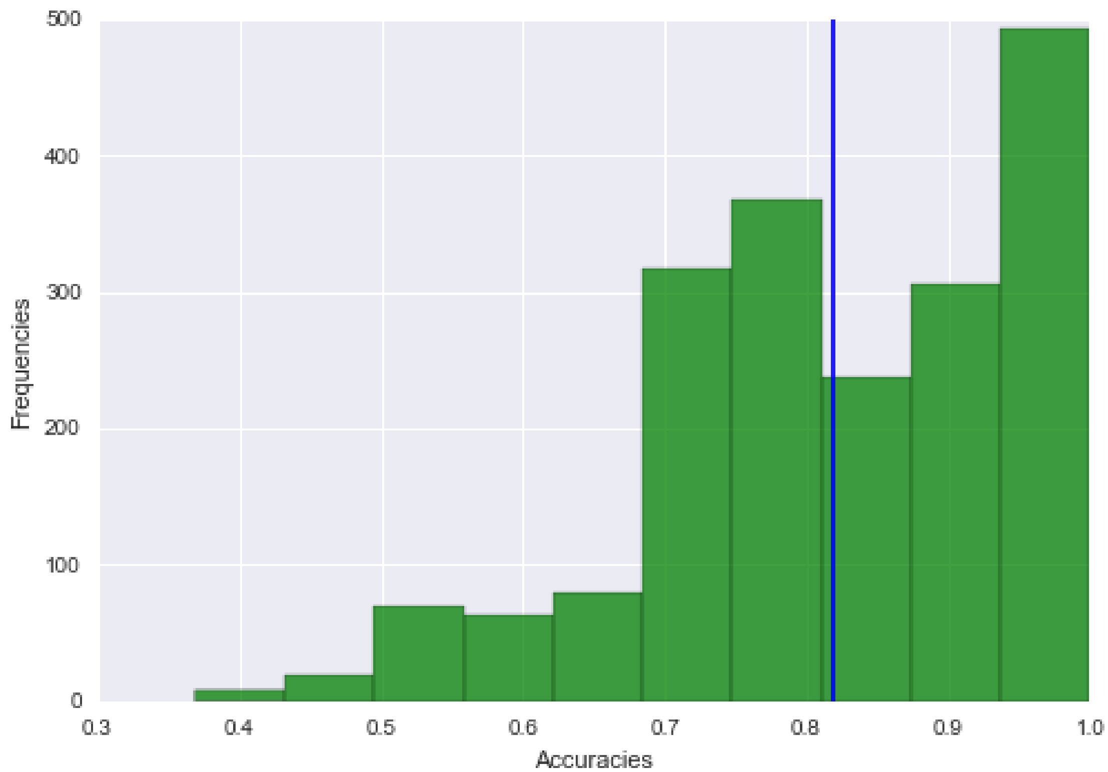
Above are the baseline results from decision tree with 273 principle components.

"Score" is very harsh because it requires the prediction to be correct for all labels to be correct for an entry. We develop a different metric for correctness as follows.

```
In [17]: import difflib
         import numpy as np
         import matplotlib.mlab as mlab
         import matplotlib.pyplot as plt

         # metric 2 - match ratio
         def compute_match_ratio(y_true, y_pred):
             # prediction accuracies according to similarity ratio
             test_similarity = []
```

```
        for i in range(0,len(y_true)):
            sm = difflib.SequenceMatcher(None, a = y_true[i], b = y_pred[i])
            # ratio() returns a measure of the sequences' similarity as a floa
            # Where T is the total number of elements in both sequences, and M
            # this is 2.0*M / T. 1.0 if the sequences are identical, and 0.0 i
            test_similarity.append(sm.ratio())

        # the histogram of the data (optional)
        n, bins, patches = plt.hist(test_similarity, facecolor='green', alpha=
        plt.xlabel('Accuracies')
        plt.ylabel('Frequencies')
        plt.axvline(x = np.mean(test_similarity))
        plt.grid(True)
        plt.show()

        # metric 2: mean and median test similarity
        return np.mean(test_similarity), np.median(test_similarity);

    # single line example
    print 'For example, when true genre labels are', y_test[0]
    print 'And predicted genre labels are', pred_test[0]
    print 'The ratio of matching labels are', difflib.SequenceMatcher(None,y_t
    print 'For the baseline model, the mean and median match ratios are', comp
```

```
For example, when true genre labels are [0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
And predicted genre labels are [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
The ratio of matching labels are 0.824561403509
For the baseline model, the mean and median match ratios are
```




```
(0.81818263148948356, 0.82456140350877194)
```

Therefore, the baseline accuracy of our decision tree models are as follows: - The proportion of complete matches in training set is 1.0. It's definitely overfitting. - The proportion of complete matches in test set is 0.0407955124936 - The mean match ratio is 82%, which means out of all predicted labels, 82% matched true values. We understand that since most values are zeros, the match ratio will always seem high. - The median match ratio is 82.5%.

1.1 Tune Random Forest Classifier

Three important parameters of sklearn's Random Forest module that influence the model fit are the number of trees, n_estimators, the number of predictors to consider for each split, max_features, and the maximum depth of the trees, max_depth. Below, we tune two of these parameters, n_estimators and max_depth, using 5-fold cross-validation on a 2D grid of parameter values.

```
In [14]: from sklearn import tree
         from sklearn import ensemble
         from sklearn.cross_validation import KFold
```

```
In [23]: # Parameters for tuning random forest
         n_trees = np.arange(60, 120, 20)  # Trees and depth are explored on an exp
```

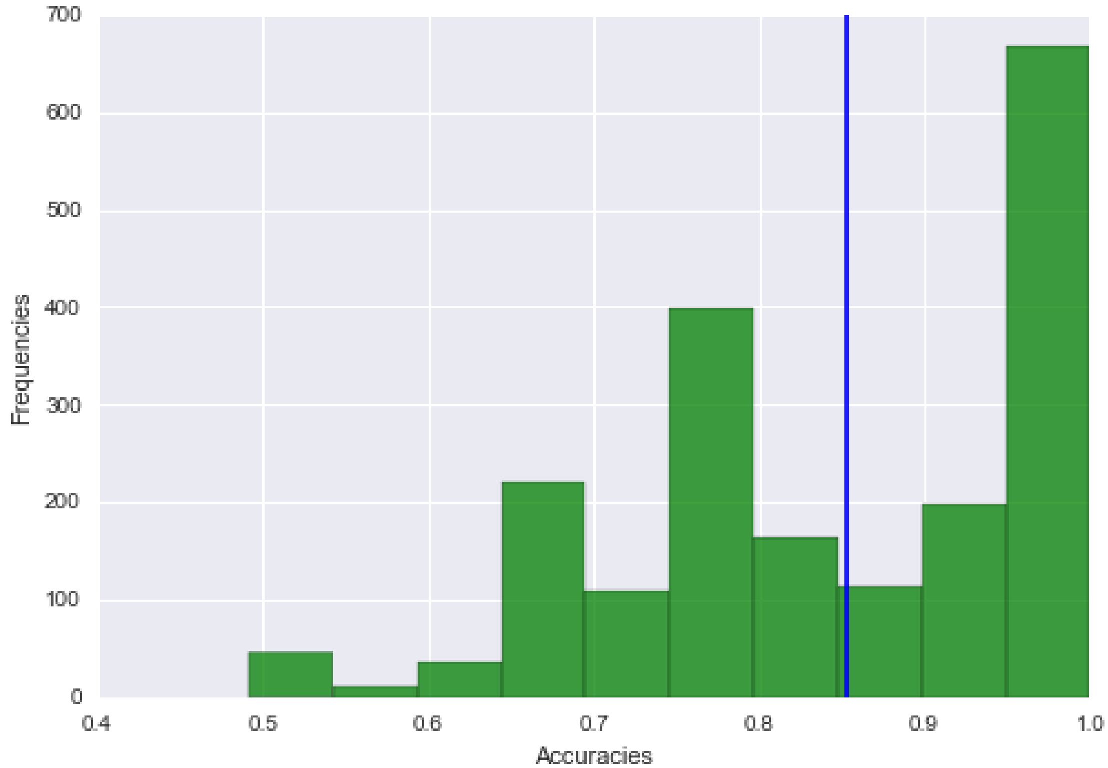
```
depths = np.arange(2, 10)  # since it is assumed that trees and depth wil

# To keep track of the best model
best_score = 0

# Run grid search for model with 5-fold cross validation
print '5-fold cross validation:'

for trees in n_trees:
    for depth in depths:
        # Cross validation for every experiment
        k_folds = KFold(X_train.shape[0], n_folds=5, shuffle=True)
        scores = []
        for train_indices, validation_indices in k_folds:
            # Generate training data
            x_train_cv = X_train.iloc[train_indices]
            y_train_cv = y_train[train_indices]
            # Generate validation data
            x_validate = X_train.iloc[validation_indices]
            y_validate = y_train[validation_indices]

            # Fit random forest on training data
            model = ensemble.RandomForestClassifier(n_estimators=trees, ma
            model.fit(x_train_cv, y_train_cv)
            # Score on validation data
            scores += [model.score(x_validate, y_validate)]

        # Record and report accuracy
        average_score = np.mean(scores)
        print "Trees:", trees, "Depth:", depth, "Score:", average_score

        # Update our record of the best parameters see so far
        if average_score > best_score:
            best_score = average_score
            best_trees = trees
            best_depth = depth
```

```
 5-fold cross validation:
Trees: 60 Depth: 2 Score: 0.089758959611
Trees: 60 Depth: 3 Score: 0.0904461614867
Trees: 60 Depth: 4 Score: 0.0975820888627
Trees: 60 Depth: 5 Score: 0.0975815113821
Trees: 60 Depth: 6 Score: 0.102687594562
Trees: 60 Depth: 7 Score: 0.100308952104
Trees: 60 Depth: 8 Score: 0.110505526489
Trees: 60 Depth: 9 Score: 0.101328205306
Trees: 80 Depth: 2 Score: 0.0901071803934
Trees: 80 Depth: 3 Score: 0.0911247011538
```

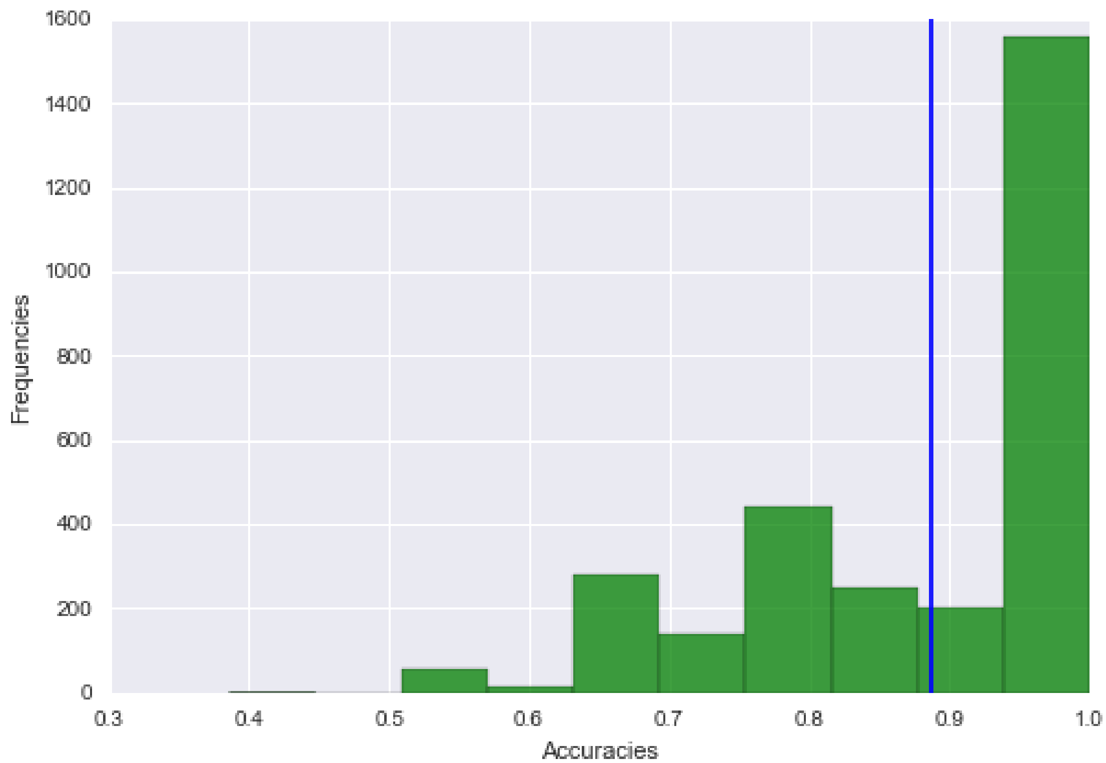
```
Trees: 80 Depth: 4 Score: 0.0952051788457
Trees: 80 Depth: 5 Score: 0.0982716006606
Trees: 80 Depth: 6 Score: 0.0989403231581
Trees: 80 Depth: 7 Score: 0.100647933197
Trees: 80 Depth: 8 Score: 0.104727833408
Trees: 80 Depth: 9 Score: 0.105069124424
Trees: 100 Depth: 2 Score: 0.0901077578739
Trees: 100 Depth: 3 Score: 0.0907868750217
Trees: 100 Depth: 4 Score: 0.0897716641835
Trees: 100 Depth: 5 Score: 0.0935085409376
Trees: 100 Depth: 6 Score: 0.10031530439
Trees: 100 Depth: 7 Score: 0.104049871222
Trees: 100 Depth: 8 Score: 0.102007899934
Trees: 100 Depth: 9 Score: 0.107446611922
```

In [24]:
```
# without pca
print 'For the tuned random forest model'
print 'Chosen number of trees, depth:', best_trees, ',', best_depth
print 'Test accuracy:', model.score(X_test, y_test)
pred_test_tuned = model.predict(X_test)
print compute_match_ratio(y_test, pred_test_tuned)
print 'Above are the mean and median match ratios on the test set'

print 'Train accuracy:', model.score(X_train, y_train)
pred_train_tuned = model.predict(X_train)
print compute_match_ratio(y_train, pred_train_tuned)
print 'Above are the mean and median match ratios on the train set'
```

```
For the tuned random forest model
Chosen number of trees, depth: 60 , 8
Test accuracy: 0.10249872514
```




```
(0.85321667248181643, 0.84210526315789469)
Above are the mean and median match ratios on the test set
Train accuracy: 0.31655899354
```




```
(0.88762624003054214, 0.94736842105263153)
Above are the mean and median match ratios on the train set
```

```
In [19]: # with pca
         # Parameters for tuning random forest
         n_trees = np.arange(80, 140, 20)  # Trees and depth are explored on an exp
         depths = np.arange(10, 15)  # since it is assumed that trees and depth wil

         # To keep track of the best model
         best_score = 0

         # Run grid search for model with 5-fold cross validation
         print '5-fold cross validation:'

         for trees in n_trees:
             for depth in depths:
                 # Cross validation for every experiment
                 k_folds = KFold(X_pca_train.shape[0], n_folds=5, shuffle=True)
                 scores = []
                 for train_indices, validation_indices in k_folds:
                     # Generate training data
                     x_train_cv = X_pca_train[train_indices]
```

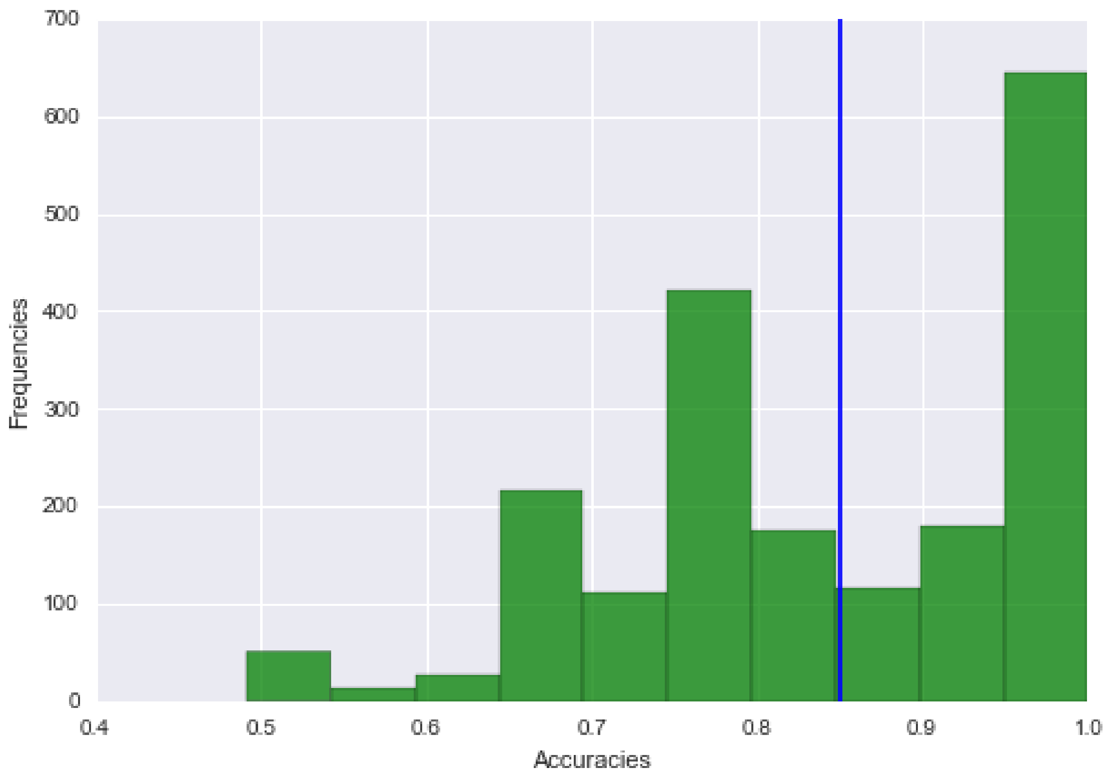
```
                y_train_cv = y_train[train_indices]
                # Generate validation data
                x_validate = X_pca_train[validation_indices]
                y_validate = y_train[validation_indices]

                # Fit random forest on training data
                model = ensemble.RandomForestClassifier(n_estimators=trees, ma
                model.fit(x_train_cv, y_train_cv)
                # Score on validation data
                scores += [model.score(x_validate, y_validate)]

            # Record and report accuracy
            average_score = np.mean(scores)
            print "Trees:", trees, "Depth:", depth, "Score:", average_score

            # Update our record of the best parameters see so far
            if average_score > best_score:
                best_score = average_score
                best_trees = trees
                best_depth = depth
```

```
5-fold cross validation:
Trees: 80 Depth: 10 Score: 0.092480625527
Trees: 80 Depth: 11 Score: 0.0931608976358
Trees: 80 Depth: 12 Score: 0.0935068084959
Trees: 80 Depth: 13 Score: 0.103363824307
Trees: 80 Depth: 14 Score: 0.09792511232
Trees: 100 Depth: 10 Score: 0.0969023942344
Trees: 100 Depth: 11 Score: 0.0907834101382
Trees: 100 Depth: 12 Score: 0.103029463059
Trees: 100 Depth: 13 Score: 0.0928276913482
Trees: 100 Depth: 14 Score: 0.0996269475532
Trees: 120 Depth: 10 Score: 0.0941789958768
Trees: 120 Depth: 11 Score: 0.0941859256436
Trees: 120 Depth: 12 Score: 0.0952011364818
Trees: 120 Depth: 13 Score: 0.0982698682189
Trees: 120 Depth: 14 Score: 0.0986082718317
```

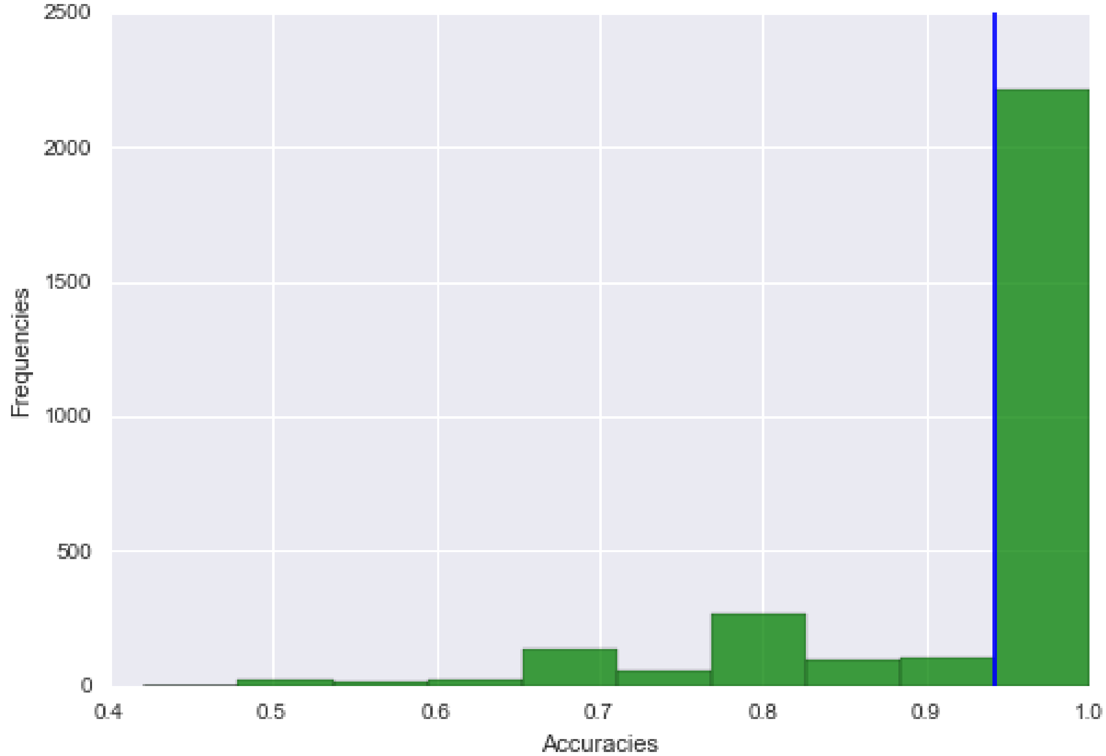
```
In [21]: # with pca
         best_pca_trees = 120
         best_pca_depth = 14

         print 'For the tuned random forest model'
         print 'Chosen number of trees, depth:', best_pca_trees, ',', best_pca_dept
         print 'Test accuracy:', model.score(X_pca_test, y_test)
         pred_test_tuned = model.predict(X_pca_test)
         print compute_match_ratio(y_test, pred_test_tuned)
```

```
    print 'Above are the mean and median match ratios on the test set'

    print 'Train accuracy:', model.score(X_pca_train, y_train)
    pred_train_tuned = model.predict(X_pca_train)
    print compute_match_ratio(y_train, pred_train_tuned)
    print 'Above are the mean and median match ratios on the train set'
```

```
For the tuned random forest model
Chosen number of trees, depth: 120 , 14
Test accuracy: 0.0963793982662
```




```
(0.84976336813476827, 0.84210526315789469)
Above are the mean and median match ratios on the test set
Train accuracy: 0.662699761986
```




```
(0.94168351855497301, 1.0)
Above are the mean and median match ratios on the train set
```

```
    # Metric 2: Mean and median test similarity
    return np.mean(test_similarity), np.median(test_similarity);
```

Part 3 SVM and Logistic Regression

0.1 SVM

In [8]:
```
# Parameters for tuning
C = np.power(10., range(-3, 4))
gammas = np.power(10., range(-3, 4))

# To keep track of the best model
best_score = 0

# Run grid search for model with 5-fold cross validation
print '5-fold cross validation:'
```

```python
for c in C:
    for gamma in gammas:
        # Cross validation for every experiment
        k_folds = KFold(x_train.shape[0], n_folds=5, shuffle=True)
        scores = []
        for train_indices, validation_indices in k_folds:
            # Generate training data
            x_train_cv = x_train.values[train_indices]
            y_train_cv = y_train[train_indices]
            # Generate validation data
            x_validate = x_train.values[validation_indices]
            y_validate = y_train[validation_indices]

            # Fit random forest on training data
            model = OneVsRestClassifier(SVC(kernel = 'rbf', C = c, gamma =
            model.fit(x_train_cv, y_train_cv)
            # Score on validation data
            scores += [model.score(x_validate, y_validate)]
        # Record and report accuracy
        average_score = np.mean(scores)
        print "c:", c, "gamma:", gamma, "Score:", average_score

        # Update our record of the best parameters see so far
        if average_score > best_score:
            best_score = average_score
            best_c = c
            best_gamma = gamma

# Fit model on entire train set using chosen C
ovr_svc_rbf = OneVsRestClassifier(SVC(kernel = 'rbf', C = best_c, gamma = b
ovr_svc_rbf.fit(x_train, y_train)
ovr_svc_rbf_score = ovr_svc_rbf.score(x_test, y_test)
ovr_svc_rbf_predicted = ovr_svc_rbf.predict(x_test)
ovr_svc_rbf_match = compute_match_ratio(y_test, ovr_svc_rbf_predicted)

print 'Chosen c:', best_c
print 'Chosen gamma:', best_gamma
print 'Test accuracy:', ovr_svc_rbf_score
print 'Match ratio: ', ovr_svc_rbf_match
```

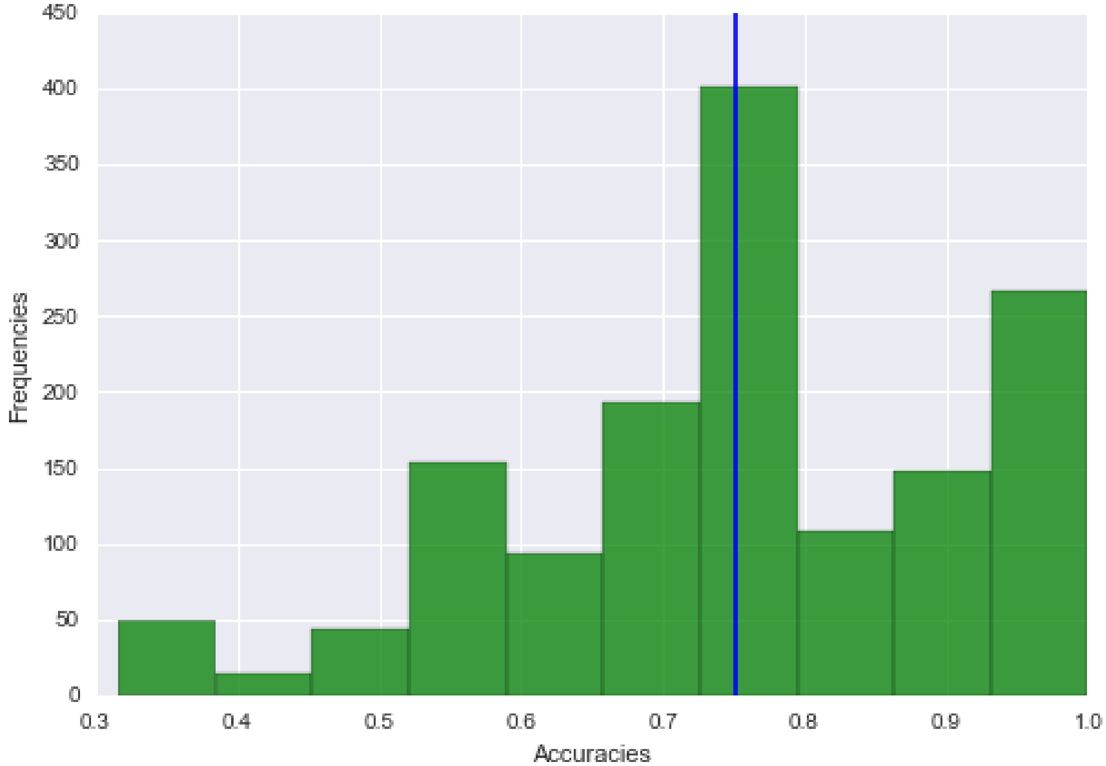
```
5-fold cross validation:
c: 0.001 gamma: 0.001 Score: 0.0204047682704
c: 0.001 gamma: 0.01 Score: 0.0204026463985
c: 0.001 gamma: 0.1 Score: 0.0204034951473
c: 0.001 gamma: 1.0 Score: 0.0204013732754
c: 0.001 gamma: 10.0 Score: 0.0204026463985
```

```
c: 0.001 gamma: 100.0 Score: 0.0204017976498
c: 0.001 gamma: 1000.0 Score: 0.0204030707729
c: 0.01 gamma: 0.001 Score: 0.0204022220242
c: 0.01 gamma: 0.01 Score: 0.0204026463985
c: 0.01 gamma: 0.1 Score: 0.0204051926447
c: 0.01 gamma: 1.0 Score: 0.0204005245267
c: 0.01 gamma: 10.0 Score: 0.0204013732754
c: 0.01 gamma: 100.0 Score: 0.0204047682704
c: 0.01 gamma: 1000.0 Score: 0.0204022220242
c: 0.1 gamma: 0.001 Score: 0.0204022220242
c: 0.1 gamma: 0.01 Score: 0.0204026463985
c: 0.1 gamma: 0.1 Score: 0.0204022220242
c: 0.1 gamma: 1.0 Score: 0.0204030707729
c: 0.1 gamma: 10.0 Score: 0.0204013732754
c: 0.1 gamma: 100.0 Score: 0.0204013732754
c: 0.1 gamma: 1000.0 Score: 0.0204034951473
c: 1.0 gamma: 0.001 Score: 0.0381796885941
c: 1.0 gamma: 0.01 Score: 0.0306033330363
c: 1.0 gamma: 0.1 Score: 0.0212743113465
c: 1.0 gamma: 1.0 Score: 0.0204030707729
c: 1.0 gamma: 10.0 Score: 0.0204030707729
c: 1.0 gamma: 100.0 Score: 0.0204039195216
c: 1.0 gamma: 1000.0 Score: 0.0204022220242
c: 10.0 gamma: 0.001 Score: 0.0282705471459
c: 10.0 gamma: 0.01 Score: 0.0291460314631
c: 10.0 gamma: 0.1 Score: 0.0218616454692
c: 10.0 gamma: 1.0 Score: 0.0204022220242
c: 10.0 gamma: 10.0 Score: 0.0204030707729
c: 10.0 gamma: 100.0 Score: 0.0204056170191
c: 10.0 gamma: 1000.0 Score: 0.0204013732754
c: 100.0 gamma: 0.001 Score: 0.0253567927483
c: 100.0 gamma: 0.01 Score: 0.027105639511
c: 100.0 gamma: 0.1 Score: 0.0221502200381
c: 100.0 gamma: 1.0 Score: 0.0204005245267
c: 100.0 gamma: 10.0 Score: 0.0204030707729
c: 100.0 gamma: 100.0 Score: 0.0204030707729
c: 100.0 gamma: 1000.0 Score: 0.0204030707729
c: 1000.0 gamma: 0.001 Score: 0.0239011886726
c: 1000.0 gamma: 0.01 Score: 0.0262318526912
c: 1000.0 gamma: 0.1 Score: 0.021276008844
c: 1000.0 gamma: 1.0 Score: 0.0204022220242
c: 1000.0 gamma: 10.0 Score: 0.0204051926447
c: 1000.0 gamma: 100.0 Score: 0.0204030707729
c: 1000.0 gamma: 1000.0 Score: 0.0204017976498
```

```
NameError                                 Traceback (most recent call last)

<ipython-input-8-88ed7ab49b37> in <module>()
     43 ovr_svc_rbf_score = ovr_svc_rbf.score(x_test, y_test)
     44 ovr_svc_rbf_predicted = ovr_svc_rbf.predict(x_test)
---> 45 ovr_svc_rbf_match = compute_match_ratio(y_test, ovr_svc_rbf_predicted)
     46
     47 print 'Chosen c:', best_c


<ipython-input-7-ea78e425dc08> in compute_match_ratio(y_true, y_pred)
      4     test_similarity = []
      5     for i in range(0,len(y_true)):
----> 6         sm = difflib.SequenceMatcher(None, a = y_true[i], b = y_pred[i]
      7         # ratio() returns a measure of the sequences' similarity as a f
      8         # Where T is the total number of elements in both sequences, an


NameError: global name 'difflib' is not defined
```

Please ignore this error message. We did not have enough time to rerun this portion to get rid of the error (took about 6 hours to run this block), so we are keeping this in the notebook. This error has nothing to do with the cross-validation. Based on the results of the cross validation, the best gamma is 0.001 and the best C is 1.0.

```
In [8]: best_c = 1.0
        best_gamma = 0.001
        ovr_svc_rbf = OneVsRestClassifier(SVC(kernel = 'rbf', C = best_c, gamma = b
        ovr_svc_rbf.fit(x_train, y_train)
        ovr_svc_rbf_score = ovr_svc_rbf.score(x_test, y_test)
        ovr_svc_rbf_predicted = ovr_svc_rbf.predict(x_test)
        ovr_svc_rbf_match = compute_match_ratio(y_test, ovr_svc_rbf_predicted)
```




```
In [9]: print 'Chosen c:', best_c
        print 'Chosen gamma:', best_gamma
        print 'Test accuracy:', ovr_svc_rbf_score
        print 'Match ratio: ', ovr_svc_rbf_match
```

```
Chosen c: 1.0
Chosen gamma: 0.001
Test accuracy: 0.0414683888511
Match ratio:  (0.75186947654656699, 0.78947368421052633)
```

```
In [ ]: # # Parameters for tuning
        # C = np.power(10., range(-1, 2))
        # polys = range(2, 5)

        # # To keep track of the best model
        # best_score = 0

        # # Run grid search for model with 5-fold cross validation
        # print '5-fold cross validation:'


        # for c in C:
```

```
#     for poly in polys:
#         # Cross validation for every experiment
#         k_folds = KFold(x_train.shape[0], n_folds=5, shuffle=True)
#         scores = []
#         for train_indices, validation_indices in k_folds:
#             # Generate training data
#             x_train_cv = x_train.values[train_indices]
#             y_train_cv = y_train[train_indices]
#             # Generate validation data
#             x_validate = x_train.values[validation_indices]
#             y_validate = y_train[validation_indices]

#             # Fit random forest on training data
#             model = OneVsRestClassifier(SVC(kernel = 'poly', C = c, degre
#             model.fit(x_train_cv, y_train_cv)
#             # Score on validation data
#             scores += [model.score(x_validate, y_validate)]
#         # Record and report accuracy
#         average_score = np.mean(scores)
#         print "c:", c, "poly:", poly, "Score:", average_score

#         # Update our record of the best parameters see so far
#         if average_score > best_score:
#             best_score = average_score
#             best_c = c
#             best_poly = poly

# # Fit model on entire train set using chosen C
# ovr_svc_poly = OneVsRestClassifier(SVC(kernel = 'poly', C = best_c, degre
# ovr_svc_poly.fit(x_train, y_train)
# ovr_svc_poly_score = ovr_svc_poly.score(x_test, y_test)
# ovr_svc_poly_predicted = ovr_svc_poly.predict(x_test)
# ovr_svc_poly_match = compute_match_ratio(y_test, ovr_svc_poly_predicted)
```

```
5-fold cross validation:
```

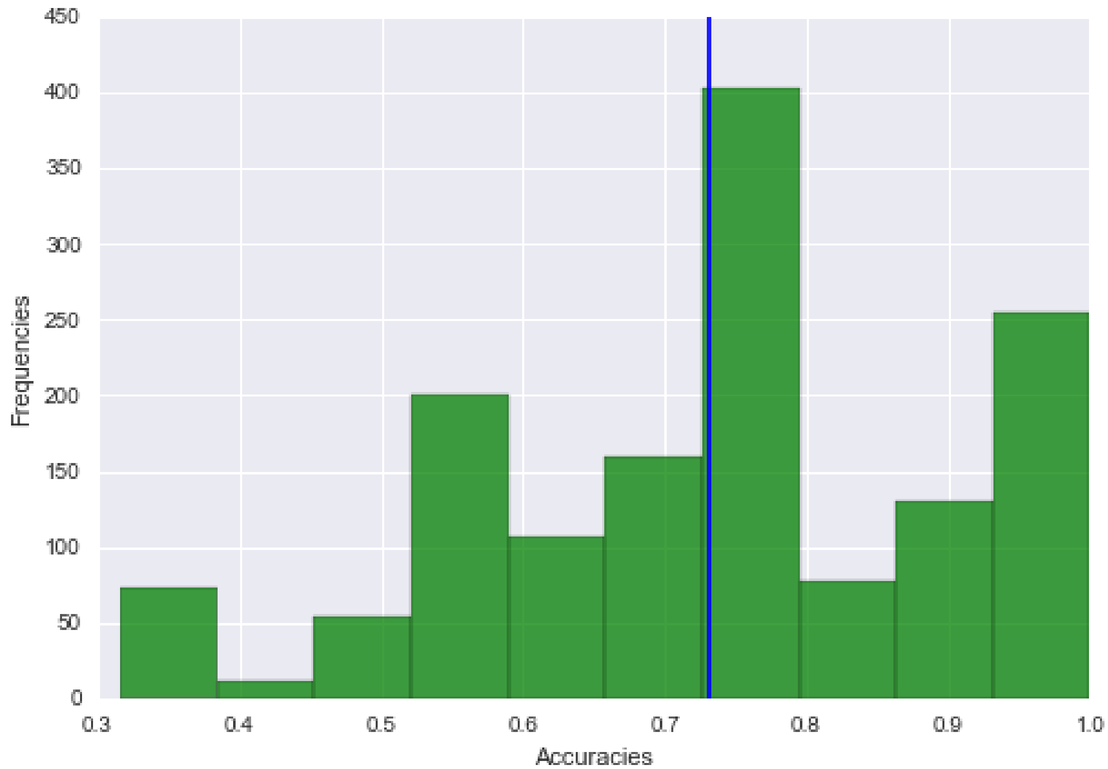
0.2 Logistic Regression

In [9]:
```
# Parameters for tuning
C = np.power(10., range(-7, 8))

# To keep track of the best model
best_score = 0

# Run grid search for model with 5-fold cross validation
print '5-fold cross validation:'
```

```python
    for c in C:
        # Cross validation for every experiment
        k_folds = KFold(x_train.shape[0], n_folds=5, shuffle=True)
        scores = []
        for train_indices, validation_indices in k_folds:
            # Generate training data
            x_train_cv = x_train.values[train_indices]
            y_train_cv = y_train[train_indices]
            # Generate validation data
            x_validate = x_train.values[validation_indices]
            y_validate = y_train[validation_indices]

            # Fit random forest on training data
            model = OneVsRestClassifier(LogisticRegression(C = c))
            model.fit(x_train_cv, y_train_cv)
            # Score on validation data
            scores += [model.score(x_validate, y_validate)]
        # Record and report accuracy
        average_score = np.mean(scores)
        print "c:", c, "Score:", average_score

        # Update our record of the best parameters see so far
        if average_score > best_score:
            best_score = average_score
            best_c = c

    # Fit model on entire train set using chosen C
    ovr_logistic = OneVsRestClassifier(LogisticRegression(C = best_c))
    ovr_logistic.fit(x_train, y_train)
    ovr_logistic_score = ovr_logistic.score(x_test, y_test)
    ovr_logistic_predicted = ovr_logistic.predict(x_test)
    ovr_logistic_match = compute_match_ratio(y_test, ovr_logistic_predicted)
```

```
5-fold cross validation:
c: 1e-07 Score: 0.0177774665699
c: 1e-06 Score: 0.017778739693
c: 1e-05 Score: 0.0177791640674
c: 0.0001 Score: 0.0177795884417
c: 0.001 Score: 0.0177795884417
c: 0.01 Score: 0.0177757690724
c: 0.1 Score: 0.0177778909443
c: 1.0 Score: 0.0177770421955
c: 10.0 Score: 0.0177804371905
c: 100.0 Score: 0.0177791640674
c: 1000.0 Score: 0.0177795884417
c: 10000.0 Score: 0.0177804371905
c: 100000.0 Score: 0.017778739693
```

```
c: 1000000.0 Score: 0.0177791640674
c: 10000000.0 Score: 0.0177804371905
```




```
In [10]: print 'Chosen c:', best_c
         print 'Test accuracy:', ovr_logistic_score
         print 'Match ratio: ', ovr_logistic_match
```

```
Chosen c: 10.0
Test accuracy: 0.0169952413324
Match ratio:  (0.73033024437368055, 0.73684210526315785)
```

```
In [ ]:
```

poster data generation

April 19, 2017

1 Code to generate 20000-movie poster data to prepare for deep learning

In [3]:
```
import json
import urllib
import cStringIO
from PIL import Image
from imdb import IMDb
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
import time
import ast
```

In [3]:
```
# 20000 dataset creation

random1 = urllib.urlopen("https://api.themoviedb.org/3/discover/movie?api_key=2dc6c9f1d1
random1_json = json.loads(random1.read())
random_movie_data_json = random1_json["results"]

pages = range(2,1001)
# np.random.shuffle(pages)
# sampled_pages = pages[:500]


# need to sleep in order to not return an error: limitation 40 requests per 10s
for i in range(len(pages)):
    if i%39 == 0:
        time.sleep(7)

    tmp_url = "https://api.themoviedb.org/3/discover/movie?api_key=2dc6c9f1d17bd39dcbaef
    tmp_page = urllib.urlopen(tmp_url)
    tmp_json = json.loads(tmp_page.read())
    for movie in tmp_json["results"]:
```

```
                random_movie_data_json.append(movie)

            if i% 100 == 0:
                print i
```

```
0
100
200
300
400
500
600
700
800
900
```

In [4]:
```
genre_ids, overview, popularity, poster_path, title, vote_average, vote_count, release_d
for movie in random_movie_data_json:
    genre_ids.append(movie["genre_ids"])
    overview.append(movie["overview"])
    popularity.append(movie["popularity"])
    poster_path.append(movie["poster_path"])
    title.append(movie["title"])
    vote_average.append(movie["vote_average"])
    vote_count.append(movie["vote_count"])
    release_date.append(movie["release_date"])
    movie_id.append(movie["id"])

data = {'title': title, 'overview': overview, 'popularity': popularity, 'release_date':
ran_df = pd.DataFrame(data = data)
```

In [7]:
```
# get genre list
genre_list = urllib.urlopen("https://api.themoviedb.org/3/genre/movie/list?api_key=2dc6c

genre_list_json = json.loads(genre_list.read())

genre_lst = {}
for i in genre_list_json['genres']:
    genre_lst[i['id']] = str(i['name'])
```

In [12]:
```
genre_lst.keys()
```

Out[12]:
```
[10752,
 80,
 10402,
 35,
 36,
 37,
```

```
 53,
 9648,
 12,
 10770,
 14,
 16,
 18,
 99,
 878,
 27,
 28,
 10749,
 10751]
```

```
In [29]: a = str(movie_20000_df.release_date[0])
         f= a.split('-')
         b = datetime.date(int(f[0]), int(f[1]), int(f[2]))
         to_integer(b)
```

```
Out[29]: 20170316
```

```
In [31]: # process new features
         movie_20000_df = ran_df.dropna()

         labels = []
         for i in movie_20000_df.genre_ids:
             label_matrix = np.zeros(len(genre_lst.keys()), dtype=int)
             for j in i:
                 if j in genre_lst.keys():
                     label_matrix[genre_lst.keys().index(j)] = 1
             labels.append(label_matrix)
         movie_20000_df['labels'] = labels

         # convert dates
         import datetime
         def to_integer(dt_time):
             return 10000*dt_time.year + 100*dt_time.month + dt_time.day

         int_dates =[]

         for i in movie_20000_df.release_date:
             f = str(i).split('-')
             try:
                 ff = (int(f[0]), int(f[1]), int(f[2]))
             except:
                 print i
             a = datetime.date(ff[0], ff[1], ff[2])
             int_dates.append(to_integer(a))
```

```
movie_20000_df['int_dates'] = int_dates
```

```
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

In [21]:
```
# to locate buffer overflow error
import csv
with open(r"20000_movie_meta2.csv", 'rb') as f:
    reader = csv.reader(f)
    linenumber = 1
```

```
        try:
            for row in reader:
                linenumber += 1
        except Exception as e:
            print (("Error line %d: %s %s" % (linenumber, str(type(e)), e.message)))
    # movie_20000_df.to_csv('20000_movie_meta.csv', encoding = 'utf-8')
```

```
Error line 1: <class '_csv.Error'> line contains NULL byte


/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:11: DeprecationWarning: BaseExceptio
```

In [70]:
```
# after correcting, we have a good csv file
movie_20000_df = pd.read_csv('20000_movie_meta_good.csv')
```

In [71]:
```
# now we download all the posters and put them into a df
# this process takes 2.5 hours
imgs = []
for i in range(len(movie_20000_df.poster_path[:10])):
    if i%39 == 0:
        # sleep
        time.sleep(7)
    try:
        url = "https://image.tmdb.org/t/p/w500" + movie_20000_df.poster_path[i]
    except:
        print "error"
        url = "https://image.tmdb.org/t/p/w500"+ '/ylXCdC106IKiarftHkcacasaAcb.jpg'
    tmp_poster = cStringIO.StringIO(urllib.urlopen(url).read())
    img = Image.open(tmp_poster)
    imgs.append(img)
#     if i %100 == 0:
#         print i
```

In [72]:
```
# take a peek
imgs[:10]
```

Out[72]:
```
[<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277C90>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277A10>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277BD0>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277E10>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277E90>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277910>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277FD0>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277C50>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277DD0>,
 <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x750 at 0x2C1277C10>]
```

In [73]:
```
# create rgb arrays for these images
# 0 padding to those with less than 500 width
```

```
# so now all rgb pixels will have the same format
RGB = []
for img in imgs:
    tmp = img.load()
    pixels = []
    # crop if larger than 750
    if (img.size[1] > 750):
        for i in range(img.size[0]):
            for j in range(750):
                pixels.append(tmp[i,j])
    else:
        for i in range(img.size[0]):
            for j in range(img.size[1]):
                pixels.append(tmp[i,j])
    # add 0 paddings if less than 750
    if (img.size[1] < 750):
        for p in range(img.size[0]):
            for q in range(img.size[1], 750):
                pixels.append((0,0,0))
    RGB.append(pixels)
```

In [76]:
```
data_img = {'movie_id': movie_20000_df.movie_id,'genre_ids': movie_20000_df.genre_ids,
img_df = pd.DataFrame(data = data_img)
img_df.head()
```

Out[76]:
```
                                                 RGB  \
0  [(12, 32, 65), (11, 31, 64), (21, 41, 74), (28...
1  [(7, 9, 8), (5, 7, 6), (7, 9, 8), (7, 9, 8), (...
2  [(92, 79, 107), (89, 76, 104), (95, 80, 109), ...
3  [(140, 51, 17), (135, 46, 12), (133, 44, 10), ...
4  [(255, 255, 255), (255, 255, 255), (255, 255, ...

                    genre_ids  \
0          [14, 10402, 10749]
1               [28, 18, 878]
2  [16, 35, 18, 10751, 10402]
3                [28, 12, 14]
4                [28, 80, 53]


                                                imgs  movie_id
0  <PIL.JpegImagePlugin.JpegImageFile image mode=...    321612
1  <PIL.JpegImagePlugin.JpegImageFile image mode=...    263115
2  <PIL.JpegImagePlugin.JpegImageFile image mode=...    335797
3  <PIL.JpegImagePlugin.JpegImageFile image mode=...    293167
4  <PIL.JpegImagePlugin.JpegImageFile image mode=...    337339
```

In [77]:
```
# produce a csv
img_df.to_csv('imgs.csv')
```

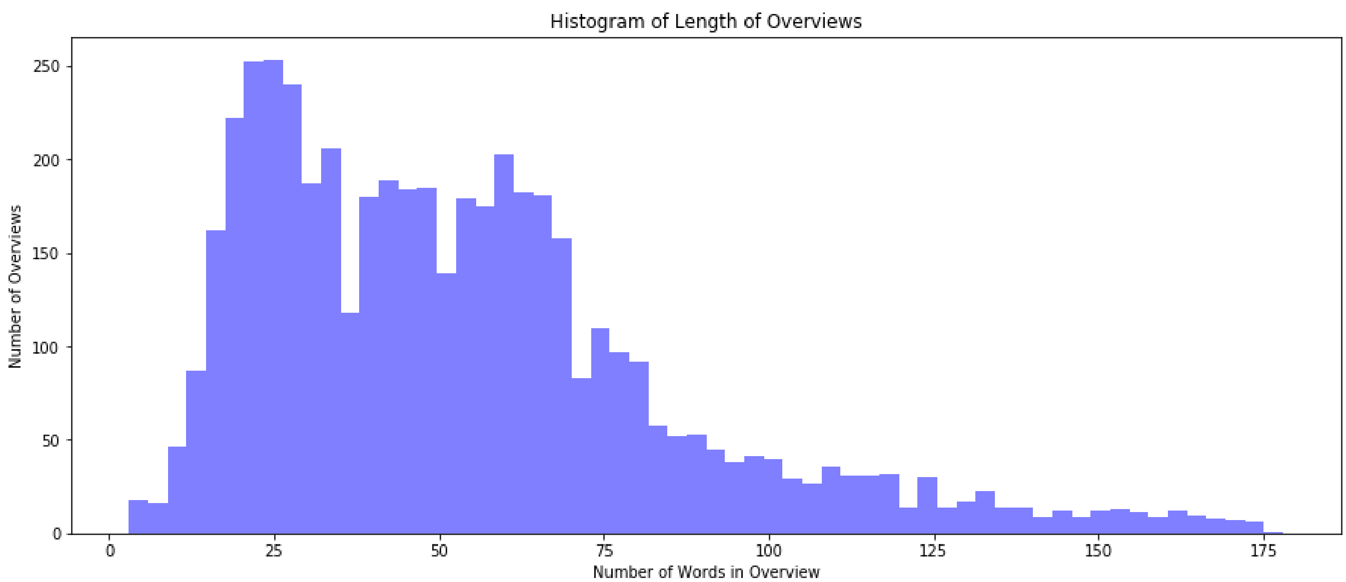
Detailed description and implementation of two different models

We decided to adopt a multilabel classification to our modeling approach. For each movie in our data set, we create a 1x19 row vector in the form of $[1, 0, ..., 0]$ where a 1 indicates that the movie is in the given genre and a 0 indicates that the movie is not in the given genre. We assume that one movie could have multiple genres. Our predictions of the test data also follow the same format.

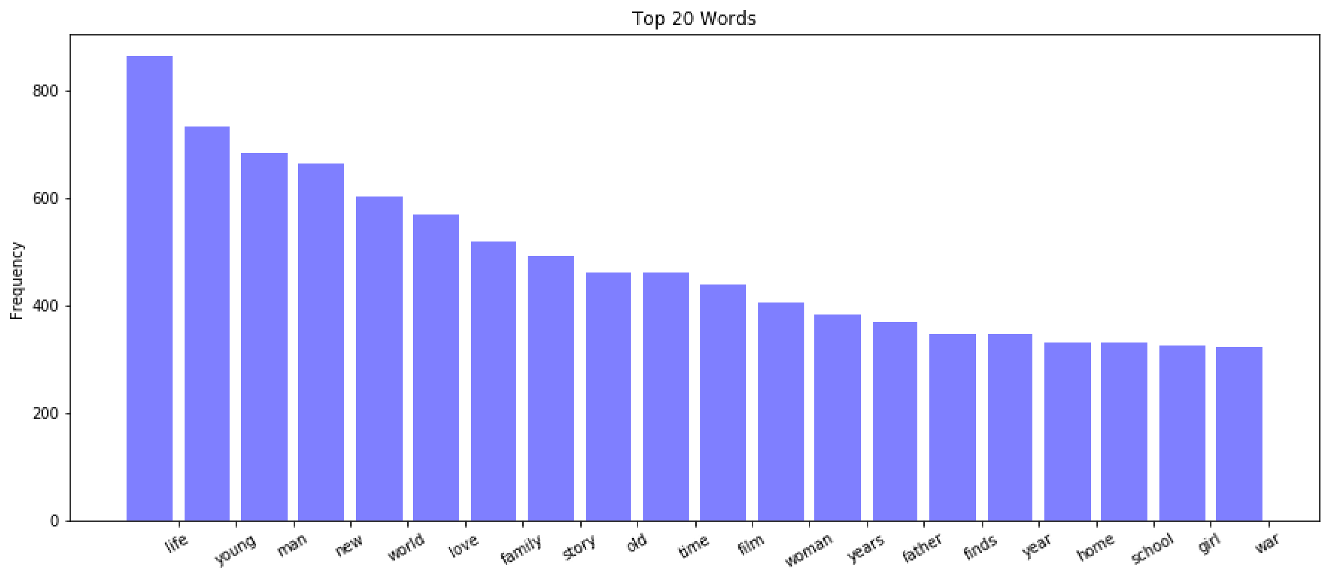
The features for our model include `overview`, `popularity`, `vote_average`, `vote_count`, `int_dates` and the processed text features.

<u>Our approach for text analysis:</u>
Our data has features `title` and `overview` that are the titles and plot overviews of the movies. We did these analysis on the text features: (code can be found in TextVectorizing.ipynb and TextVectorizing.pdf)

1. Histogram of the lengths of overviews



2. Top 20 words in the titles and overviews



3. Extracting top 100 words from each of the 19 genres, for a total of 1900 words
4. Vectorizing the text features of the movies in our dataset to the 1900 words
5. PCA on the text vectors.

For tasks 2,3,4, we used the CountVectorizer from the sklearn.feature_extraction.text library, which excludes stop words and counts the frequency of words in a text.

Task 3: To have a balanced representation of top words in each genre for vectorizing the movies' text features, we extracted 100 top words from each genre. For each genre, we looked for the subset of movies in our data that has this genre in its metadata, then we count the frequency of words from this subset. After this process, we have 1900 top words.

Task 4: For each movie in our dataset, we ran the CountVectorizer on its title and overview, then store the frequency of each of the 1900 top words in this movie's text features. After this process, we have a dataframe of ~4900 rows and 1900 columns.

Task 5: We ran PCA to reduce the dimensions of our text vectors. Our text vectors were reduced to 300 components, which captured 90% of the variance in the data.

<u>Our models:</u>

One model that we tried is the `OneVsRestClassifier` model from the sklearn library (http://scikit-learn.org/stable/modules/multiclass.html). This model uses a one-vs-all approach to make multi-label predictions. Specifically, the model fits a classification approach for each class and allows the movies to have multiple classes in the prediction phase.

We attempted two different approaches for the `OneVsRestClassifier` model, SVM and Logistic Regression. For SVM, we decided to use the RBF kernel (we also tried the linear and polynomial kernels, but the code took too long to run and we were not sure if the code was running at all) and cross validated to obtain the proper values for C and gamma. For logistic regression, we also cross validated to find the best C.

- Description of your performance metrics
We used two different performance metrics.

The first one is the complete-match accuracy rate. This metric is very harsh in a multi-label classification with 19 different labels, and our best model (random forest) results in an accuracy rate of 10%.

The second metric we used is match ratio, which is the fraction of labels that are correctly predicted. This metric is much more lenient, giving us a result of 85% for our best model.

- Careful performance evaluations for both models
Model 1: Random Forest for text features post pca and numerical data before pca

- Chosen number of trees, depth: 60 , 0. Test accuracy: 10.24. This is a 7% improvement on a single decision tree classifier model.
- The training accuracy is 0.31.
- The mean and median match ratios on the test set are (0.85, 0.84).
- The mean and median match ratios on the training set are (0.91, 0.98).

Model 2: Random Forest for text features post pca and numerical data before pca

- Overall, Model 2 did a little better than model 1.
- Chosen number of trees, depth: 120, 14. Test accuracy: 0.96.
- The training accuracy is 0.66, 30% higher than the result from Model 1.
- The mean and median match ratios on the test set are (0.85, 0.84).
- The mean and median match ratios on the training set are (0.94, 1.0).

Model 3: One-Vs-All Classifier using a SVM with a RBF kernel

- Chosen gamma: 0.001, C: 1.0, Test accuracy: 0.96.
- Match ratio (0.75, 0.79)

Model 4: One-Vs-All Classifier using a Logistic Regression

- Chosen C 10^-6, , Test accuracy: 0.015
- Match ratio: (0.73, 0.74)

- Visualizations of the metrics for performance evaluation
See figures produced after tuning.

- **Discussion of the differences between the models, their strengths, weaknesses, etc.**
Model 1 vs. Model 2 (Random Forest)
Model 2 is somewhat faster to train than model 1 because we did pca on all model 1 inputs. This allows us to try deeper trees. However, best performance accuracy suffered because we didn’t have a larger sample and gave up some predictive power by keeping the principle components that explain 90% of the variance.

Model 3 vs. Model 4 (One-Vs-All Classifier)
Both models use the one-vs-all classifier from the sklearn library, but with different classification models for each label. Model 3 uses a SVM with a RBF kernel; Model 4 uses a Logistic Regression. Both models had similar levels of performance. SVM is not a probabilistic model, so it does not allow for predicting the probabilities for each label. However, logistic regression is a probabilistic model, so it allows for predicting probabilities for each label.

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

Ideas on Improving Performance

- Add more text data. The current text data is very sparse because overviews are short for most entries.
- Add more numerical data. Budget and revenue might be predictive of genre.
- Process the text data differently.
    - N-Gram Approach: We can break down the overviews and turn the words into combinations. This method preserves the sequence of the words and retains more information than a regular bag-of-words method does.
- Different models:
    - Clustering
    - LDA, QDA
    - KNN
    - GAM
- Assign weights to the classes so that we can handle the imbalanced classes better.