proj

May 9, 2015

1 Predicting Anime Ratings: Linear Regression vs. Neural Nets

- Name: Austin Chen
- SID: 23826762

- Repro: Open up proj.ipynb in IPython Notebook.

The goal of my 289 project is to analyze and predict what makes for a good anime, by drawing on hundreds of thousands of user votes. Just like movies, TV shows, and other forms of media, anime can be categorized into different genres. In addition, there are many common themes found in anime, some of which are quite unique to the medium (for example, magical girls). I'll be leveraging linear regression and neural networks as two different ways to predict anime scores.

In [254]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET
import urllib.request
import time
import numpy as np
import collections
import random
```

1.1 Featurizing

For each anime, I included as features: the number of votes, the number of views, and a binary indicator for each of the genres and the 50 most common themes. These were used to predict anime ratings, which were provided as averages across a 10-point scale.

Example genres included action, adventure, and comedy; example themes include aliens, assassins, and bishounen.

All anime data was made available courtesy of Anime News Network.

In []:
```
# Download the list of all anime, sorted by ratings
urllib.request.urlretrieve('http://www.animenewsnetwork.com/encyclopedia/reports.xml?id=172&nlis
tree = ET.parse('anime_by_ratings.xml')
root = tree.getroot()

# Download details for blocks of 40 anime at a time
for i in range(0, 4680, 40):
    print(i)
    ids = [item.attrib['id'] for item in root[i:i+40]]
    query = 'http://cdn.animenewsnetwork.com/encyclopedia/api.xml?title=' + '/'.join(ids)
    urllib.request.urlretrieve(query, 'anime/{0}.xml'.format(i))
    time.sleep(60)
```

```
In [353]: # nb_votes, nb_seen, genres, themes
          # TODO Maybe add: # episodes? running time? animation studio/prodcution? year?

          animes, scores = np.zeros((4679, 68)).astype(int), np.zeros(4679)
          for i, item in enumerate(root[:4679]):
              votes = int(item.find('nb_votes').text)
              seen = int(item.find('nb_seen').text)
              average = float(item.find('weighted_average').text)

              animes[i][0:2] = np.array([votes, seen])
              scores[i] = average

In [199]: # Count occurences of genres and themes
          themes, genres = collections.Counter(), collections.Counter()
          for index in range(0, 4680, 40):
              ann = ET.parse('anime/{0}.xml'.format(index)).getroot()
              genres.update(genre.text.lower() for genre in ann.findall("./*/info[@type='Genres']"))
              themes.update(theme.text.lower() for theme in ann.findall("./*/info[@type='Themes']"))


          # Choose most common genres and themes as features
          all_themes = sorted([theme for theme, count in themes.most_common(50)])
          all_genres = sorted([genre for genre, count in genres.most_common(16)])

          def featurize(features, all_features):
              return np.array([1 if a in features else 0 for a in all_features])

In [355]: for index in range(0, 4680, 40):
              ann = ET.parse('anime/{0}.xml'.format(index)).getroot()
              for offset, anime in enumerate(ann):
                  i = index + offset
                  if i >= 4679:
                      break

                  # Build data matrix
                  g = [genre.text.lower() for genre in anime.findall("./info[@type='Genres']")]
                  animes[i][2:18] = featurize(g, all_genres)

                  t = [theme.text.lower() for theme in anime.findall("./info[@type='Themes']")]
                  animes[i][18:68] = featurize(t, all_themes)
```

1.2 Linear Regression

Linear regression on the average user rating for each anime provided a baseline prediction strategy, which had rather large errors (between -2 and 2 on a 10 point scale). However, it was quite useful for drawing human-interpretable conclusions for the various effects of the different features.

Genres that were highly rated included drama, thriller, slice of life, and mystery, and highly rated themes included aliens, space, sports, and crime. Meanwhile, erotica and horror were heavily penalized as genres, as were the themes of male harem, samurai, and shounen-ai.

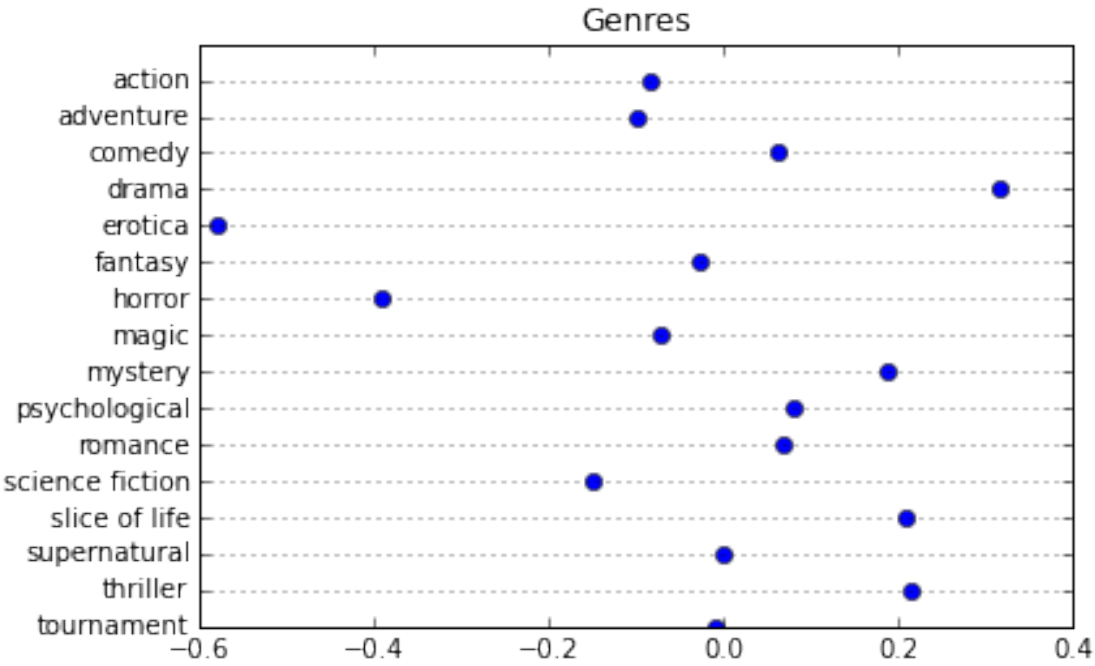
Interestingly, while the rating was positively correlated with the number of votes, it was negatively correlated with the number of views, leading to the tentative conclusion is that popular anime is terrible.

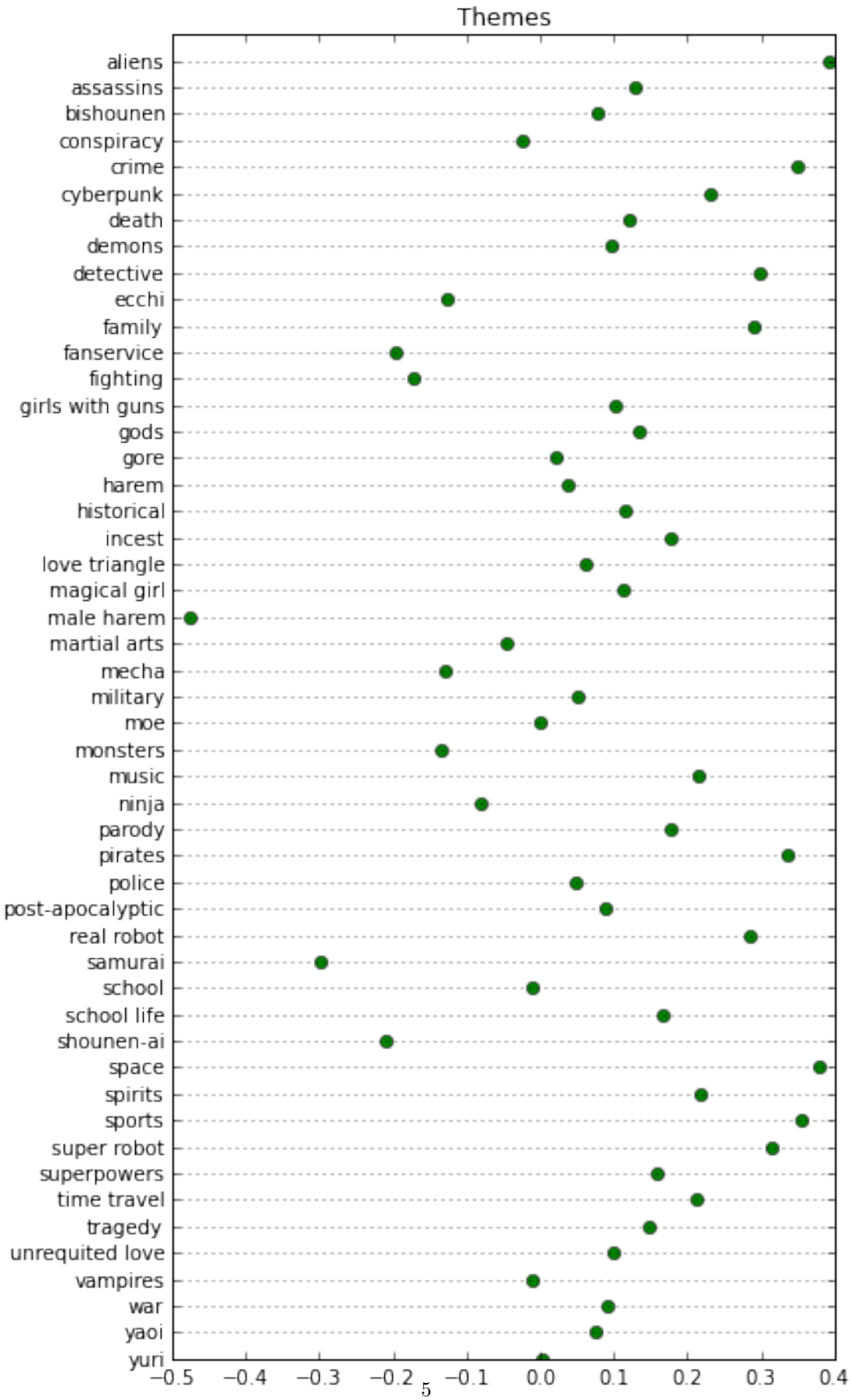
```
In [360]: # Regress scores based on anime information

          # Introduce extra row for constant term
```

```
        X = np.hstack((animes, np.ones((len(animes), 1))))
        y = scores

        # Solve for B, the weights vector
        B = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y))
```

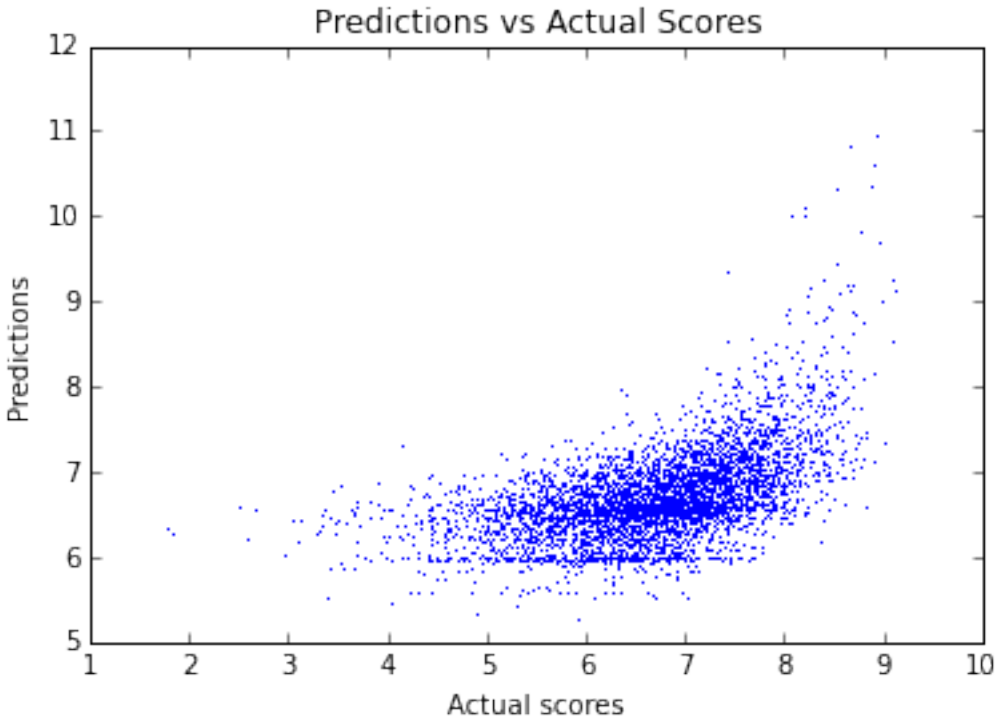
In [396]:
```
print("B0 (the constant term) is", B[-1])

        # Plot the various weights used in regression
        plt.title("Other")
        yaxis = range(2)[::-1]
        plt.plot(B[0:2]*1000, yaxis, 'ro')
        plt.yticks(yaxis, ["1000 Votes", "1000 Seen"])
        plt.grid(axis='y')
        plt.margins(y=0.5)
        plt.gcf().set_size_inches(6, 0.6)
        plt.show()

        plt.title("Genres")
        yaxis = range(16)[::-1]
        plt.plot(Bm[2:18], yaxis, 'o')
        plt.yticks(yaxis, all_genres)
        plt.grid(axis='y')
        plt.show()

        plt.title("Themes")
        yaxis = range(50)[::-1]
        plt.plot(B[18:68], yaxis, 'go')
        plt.yticks(yaxis, all_themes)
        plt.grid(axis='y')
        plt.gcf().set_size_inches(6,12)
        plt.show()
```

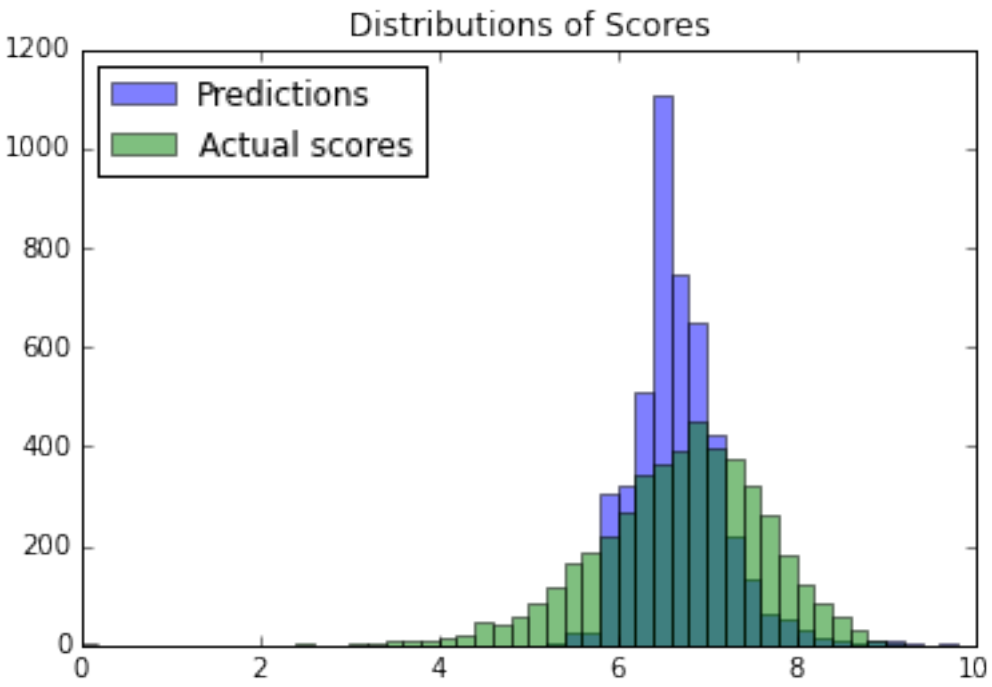
```
B0 (the constant term) is 6.53347904931
```

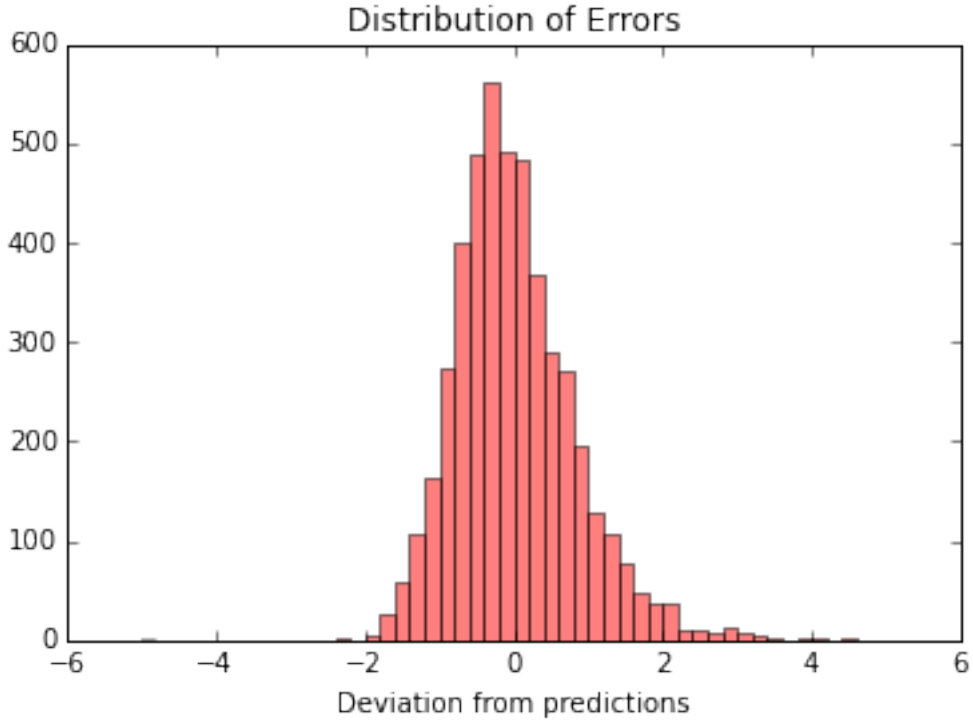

Genres
action
adventure
comedy
drama
erotica
fantasy
horror
magic
mystery
psychological
romance
science fiction
slice of life
supernatural
thriller
tournament
−0.6
−0.4
−0.2
0.0
0.2
0.4


Themes
aliens
assassins
bishounen
conspiracy
crime
cyberpunk
death
demons
detective
ecchi
family
fanservice
fighting
girls with guns
gods
gore
harem
historical
incest
love triangle
magical girl
male harem
martial arts
mecha
military
moe
monsters
music
ninja
parody
pirates
police
post-apocalyptic
real robot
samurai
school
school life
shounen-ai
space
spirits
sports
super robot
superpowers
time travel
tragedy
unrequited love
vampires
war
yaoi
yuri
−0.5
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4

In [387]:
```
predictions = np.dot(X, B)

# Some interesting visualizations
plt.title("Predictions vs Actual Scores")
plt.plot(y, predictions, 'b,')
plt.xlabel("Actual scores")
plt.ylabel("Predictions")
plt.show()

plt.title("Distributions of Scores")
plt.hist(predictions, bins=np.arange(0, 10, 0.2), alpha=0.5, label="Predictions")
plt.hist(y, bins=np.arange(0, 10, 0.2), alpha = 0.5, label="Actual scores")
plt.legend(loc="upper left")
plt.show()

plt.title("Distribution of Errors")
plt.hist(predictions - y, bins=np.arange(-5, 5, 0.2), color='r', alpha=0.5)
plt.xlabel("Deviation from predictions")
plt.show()
```







In [341]: *# Calculate regression sum of squares*

```
        RSS = sum((np.dot(X, B) - y) ** 2)
        print("RSS is", RSS)
```

```
RSS is 3027.11535137
```

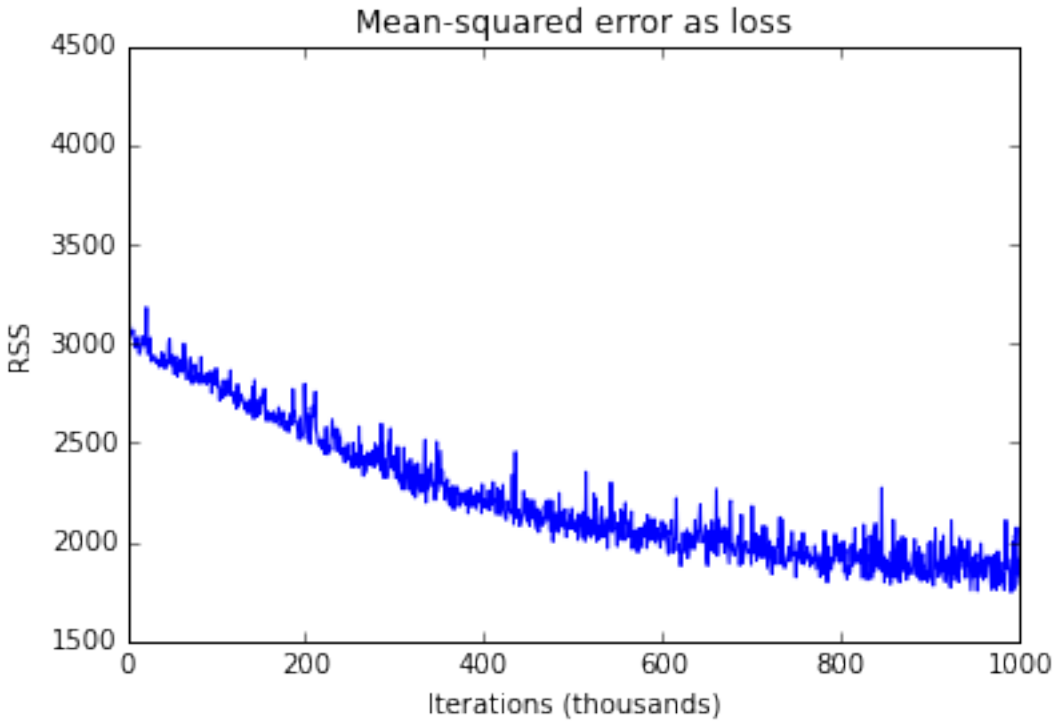
1.3 Neural Nets

Neural networks proved to provide better predictions. I used a neural net with 1 hidden layer with an activation of tanh, and 1 output node with no activation function. Over 1 million training iterations, it had a regression sum of squares of 1790, com

```
In [381]: class NeuralNet(object):
              def __init__(self):
                  e = 10**-5
                  self.W1 = np.random.normal(0, e, (69, 40))
                  self.W2 = np.random.normal(0, e, (41, 1))

              def forward(self, x):
                  a2 = np.tanh(x.dot(self.W1))
                  a2 = np.c_[a2, np.ones(len(a2))]
                  hx = a2.dot(self.W2)
                  return hx

              def train(self, X, Y, iterations, step, error):
                  for i in range(iterations):
                      i = random.randrange(0, len(X))
                      x, y = X[i], Y[i] # x and y are both row vectors

                      a2 = np.tanh(x.dot(self.W1))
                      a2 = np.append(a2, 1)
                      hx = a2.dot(self.W2)
                      d3 = error(y, hx)

                      dJdW2 = - np.outer(a2.T, d3)
                      dJdW1 = - np.outer(x.T, d3.dot(self.W2.T) * (1 - a2**2))

                      self.W2 -= step * dJdW2
                      self.W1 -= step * dJdW1[:,:-1]

              def predict(self, images):
                  return self.forward(images).flatten()

              def score(self, data, labels):
                  return sum((self.predict(data) - labels) ** 2)

              def save(self, filename='default'):
                  np.savez(filename, self.W1, self.W2)

              def load(self, filename='default'):
                  arrs = np.load(filename)
                  self.W1, self.W2 = arrs['arr_0'], arrs['arr_1']

          def mean_squared_error(y, hx):
              return y - hx
```

```
        def cross_entropy_error(y, hx):
            return np.divide(y, hx) - np.divide(1 - y, 1 - hx)
```

In [392]:
```
        # Train the neural net
        scores = []
        Xp = sklearn.preprocessing.scale(X)
        nn = NeuralNet()
        for _ in range(1000):
            nn.train(Xp, y, 1000, 0.01, mean_squared_error)
            scores.append(nn.score(Xp, y))
        # nn.save()

        # Plot RSS over iterations
        plt.plot(scores)
        plt.title("Mean-squared error as loss")
        plt.ylabel("RSS")
        plt.xlabel("Iterations (thousands)")
        plt.show()
```

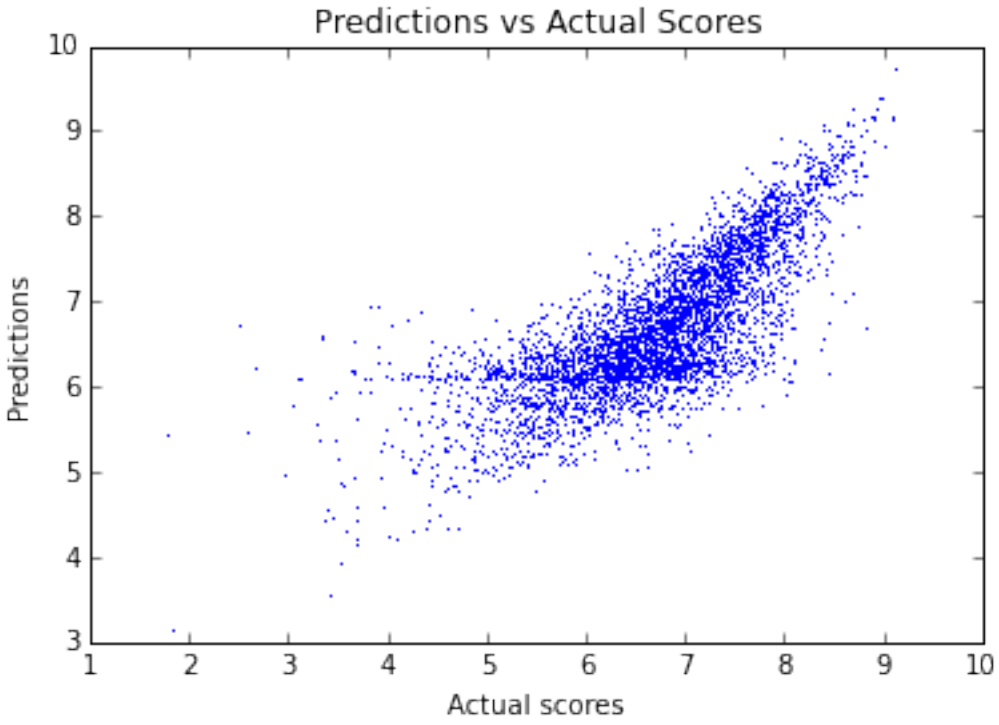


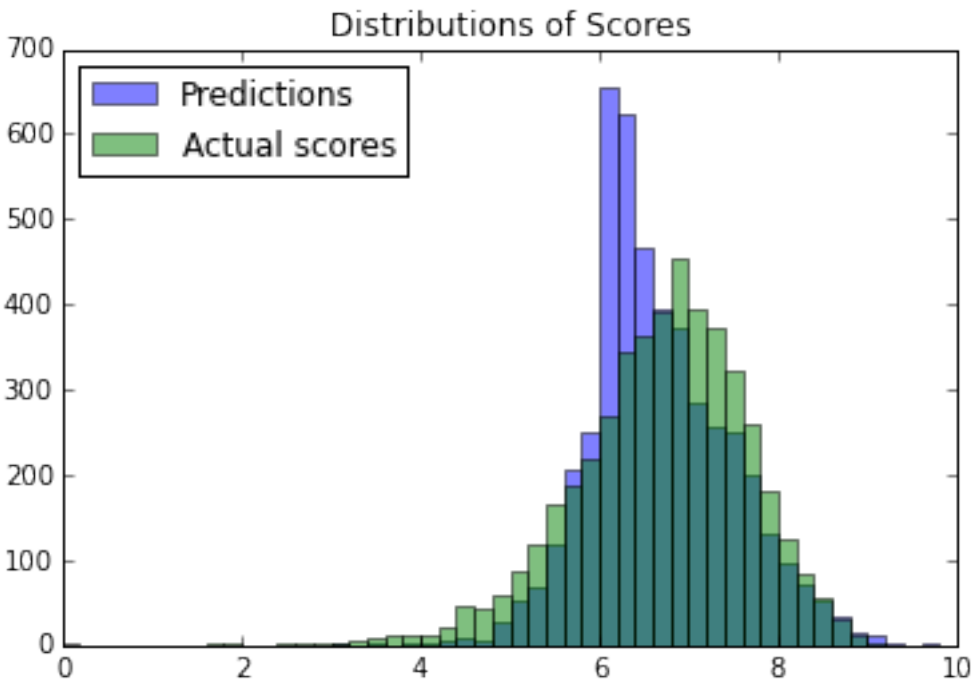
In [394]:
```
        # Calculate regression sum of squares
        RSS = nn.score(Xp, y)
        print("RSS is", RSS)
```

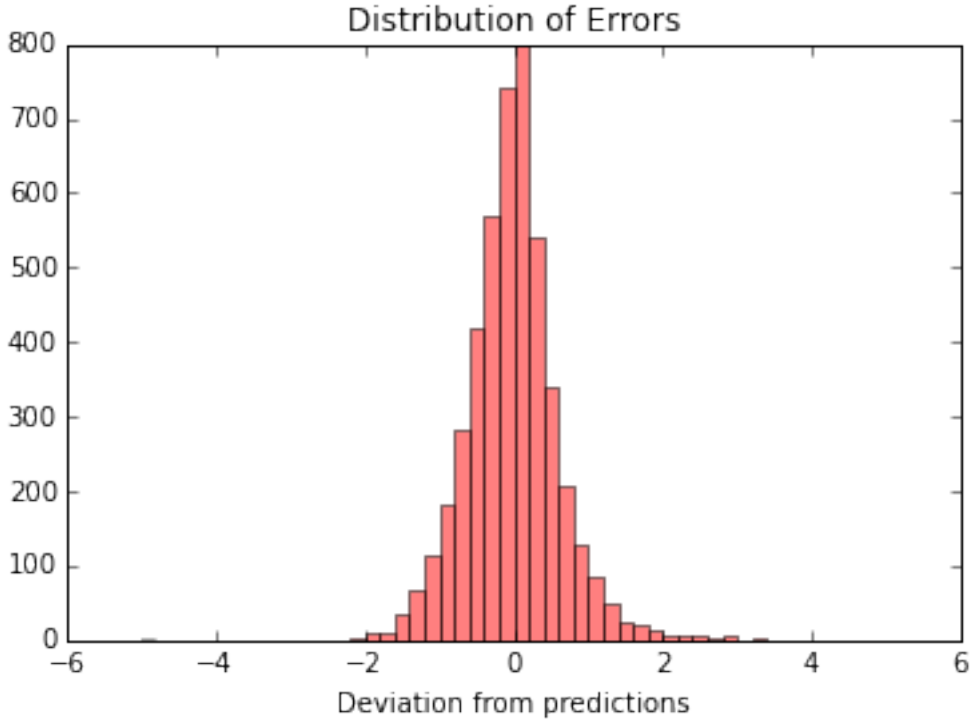
```
RSS is 1790.37986983
```

In [395]:
```
        predictions = nn.predict(Xp)

        # Some interesting visualizations
```

```
plt.title("Predictions vs Actual Scores")
plt.plot(y, predictions, 'b,')
plt.xlabel("Actual scores")
plt.ylabel("Predictions")
plt.show()

plt.title("Distributions of Scores")
plt.hist(predictions, bins=np.arange(0, 10, 0.2), alpha=0.5, label="Predictions")
plt.hist(y, bins=np.arange(0, 10, 0.2), alpha = 0.5, label="Actual scores")
plt.legend(loc="upper left")
plt.show()

plt.title("Distribution of Errors")
plt.hist(predictions - y, bins=np.arange(-5, 5, 0.2), color='r', alpha=0.5)
plt.xlabel("Deviation from predictions")
plt.show()
```




Distributions of Scores
Predictions
Actual scores
700
600
500
400
300
200
100
0
0
2
4
6
8
10


Distribution of Errors
800
700
600
500
400
300
200
100
0
−6
−4
−2
0
2
4
6
Deviation from predictions