Assignment 3

Benjamin Jakubowski

February 29, 2016

2. Calculating Subgradients

2.1 Subgradients for pointwise maximum of functions

Suppose $f_1, \ldots, f_m : \mathbb{R}^d \to \mathbb{R}$ are convex functions. Now let

$$f(x) := \max_{i=1,\ldots,m} f_i(x)$$

Let k be any index for which $f_k(x) = f(x)$ and choose $g \in \partial f_k(x)$. We now show $g \in \partial f(x)$.
Since $g \in \partial f_k(x)$, for all z

$$f_k(z) \geq f_k(x) + g^T(z - x)$$

We must show

$$f(z) \geq f(x) + g^T(z - x)$$

Well, since $f_k(x) = f(x)$, we have

$$f_k(x) + g^T(z - x) = f(x) + g^T(z - x) \leq f_k(z)$$

But $f(z) \geq f_k(z)$ (by $f(x) := \max_{i=1,\ldots,m} f_i(x)$), so

$$f(x) + g^T(z - x) \leq f_k(z) \leq f(z)$$

and

$$f(x) + g^T(z - x) \leq f(z)$$

Thus we conclude $g \in \partial f(x)$.

2.2 Subgradients for hinge loss for linear prediction

First, recall $J(w) = \max\{0, 1 - yw^T x\}$.
We want to find a subgradient of J- let's just find the subdifferential. There are three cases:

- Case 1: If $yw^T x > 1$, then $J(w) = \max\{0, 1 - yw^T x\} = 0$. Thus, J is differentiable, and $\nabla J(w) = 0$.

- Case 2: If $yw^Tx < 1$, then $J(w) = \max\{0, 1 - yw^Tx\} = 1 - yw^Tx$. Again, J is differentiable, and $\nabla J(w) = -yx$.
- Case 3: If $yw^Tx = 1$, then $0 = 1 - yw^Tx$. J is not differentiable, so $\partial J(w)$ is the convex hull of $\{0\} \cup \{-yx\}$, which is

$$\{\alpha_1 \cdot 0 + \alpha_2(-yx) | \alpha_1, \alpha_2 \geq 0, \alpha_1 + \alpha_2 = 1\} = \{\alpha(-yx) | 0 \leq \alpha \leq 1\}$$

3. Perceptron

3.1 Perceptron loss on linearly separable $\mathcal{D}$

Assume $\{x | w^Tx = 0\}$ is a separating hyperplane for a training set $\mathcal{D}$. We show

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}, y) = 0.$$

First, since we have a separating hyperplane, for all $i, y_i w^T x_i > 0$ (i.e. each training data observation is correctly classified).
Since $\hat{y} = w^T x_i$, for all i

$$-\hat{y}_i y_i = -w^T x_i y_i = -y_i w^T x_i < 0$$

So for all i, $\max\{0, -\hat{y}_i y_i\} = 0$. Therefore (recalling $\ell(\hat{y}, y) = \max\{0, -\hat{y}y\}$)

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}, y) = 0$$

3.2 Comparing SGD and the Perceptron algorithm

If we use SGD to minimize the empirical risk with perceptron loss, with fixed $\alpha = 1$, then our update rule is

$$w^{(k+1)} \leftarrow w^{(k)} - \alpha g = w^{(k)} - g$$

where g is a subgradient of our SGD empirical risk function- specifically,

$$g = \begin{cases} 0 & \text{if } -y_i {w^{(k)}}^T x_i < 0 \\ -y_i x_i & \text{if not} \end{cases}$$

For reference, note that for $-y_i {w^{(k)}}^T x_i > 0, \nabla_w(-y_i {w^{(k)}}^T x_i) = -y_i x_i$, so (by the same argument as Case 3 in 2.2), $-y_i x_i$ is also a subgradient when $-y_i {w^{(k)}}^T x_i = 0$.
Thus, we can rewrite our update rule as

$$w^{(k+1)} \leftarrow \begin{cases} w^{(k)} - g = w^{(k)} - 0 = w^{(k)} & \text{if } -y_i {w^{(k)}}^T x_i < 0 \\ w^{(k)} - g = w^{(k)} - -y_i x_i = w^{(k)} + y_i x_i & \text{if not} \end{cases}$$

This is just the Perceptron algorithm given as Algorithm 1 in the assignment (if we cycle through our data points and terminate when training loss is 0, i.e. we've constructed a separating hyperplane).

3.3 Show $w = \sum_{i=1}^{n} \alpha_i x_i$

We show this with a simple inductive argument: At $k = 0$, $w^{(0)} = (0, \ldots, 0) \in \mathbb{R}^d$. Then $(y_1 x_1^T w(k)) = (y_1 x_1^T 0) = 0 \leq 0$, so

$$w^{(1)} = w^{(0)} + y_i x_i = 0 + y_i x_i = y_i x_i$$

Now consider the $(k+1)^{st}$ step (made on the j^{th} data point). Assume $w^{(k)} = \sum_{i=1}^{n} \beta_i x_i$. Consider our update:

$$w^{(k+1)} \leftarrow \begin{cases} w^{(k)} & \text{if } -y_j {w^{(k)}}^T x_j < 0 \\ w^{(k)} + y_j x_j & \text{if not} \end{cases}$$

Since $w^{(k)}$ is a linear combination of the x_i's, it's obvious $w^{(k+1)}$ is as well.
Next, assume the algorithm terminates and returns $w = \sum_{i=1}^{n} \alpha_i x_i$.
By construction, it is apparent

$$\alpha_i \begin{cases} = 0 & \text{if } x_i \text{ was correctly classified by every } w^k \text{ such that } k \bmod n = i - 1 \\ \neq 0 & \text{if } x_i \text{ was incorrectly classified by some } w^k \text{ such that } k \bmod n = i - 1 \end{cases}$$

4. The Data

4.1 Load and randomly split the data (75-25 training-test)

See attached iPython notebook.

5. Sparse Representations

5.1 Write a function that sparsifies an input list of words

See attached iPython notebook.

6. Support Vector Machine via Pegasos

6.1 Compute a subgradient for the "stochastic" SVM objective

The "stochastic" SVM objective is

$$h_i(w) = \frac{\lambda}{2} ||w||^2 + (1 - y_i w^T x_i)_+$$

Now, note we have three cases (either side or at the non-differentiable point):

- For $y_i w^T x_i < 1, h_i(w) = \frac{\lambda}{2}||w||^2 + (1 - y_i w^T x_i)$. This is differentiable with respect to w, so $\partial h_i(w) = \{\nabla_w h_i\} = \{\lambda w - y_i x_i\}$. Thus, the update rule (with $\eta_t = 1/(\lambda t)$) is

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_t g \\ &\leftarrow w_t - \eta_t(\lambda w - y_i x_i) \\ &\leftarrow (1 - \eta_t \lambda) w_t + \eta_t y_i x_i \end{aligned}$$

- For $y_i w^T x_i > 1, h_i(w) = \frac{\lambda}{2}||w||^2$, so $\partial h_i(w) = \{\nabla_w h_i\} = \{\lambda w\}$. Thus, the update rule is

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_t g \\ &\leftarrow w_t - \eta_t \lambda w \\ &\leftarrow (1 - \eta_t \lambda) w_t \end{aligned}$$

- For $y_i w^T x_i = 1$, we have $\partial h_i(w) = \mathbf{Conv}\{\lambda w\} \cup \{\lambda w - y_i x_i\}$. Thus, λw is itself a subgradient, and

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_t g \\ &\leftarrow (1 - \eta_t \lambda) w_t \end{aligned}$$

6.2 Implement the Pegasos algorithm

See attached iPython notebook.

6.3 Implement the improved Pegasos algorithm

The initial implementation of the Pegasos algorithm is very slow, since at each stochastic step we must update every entry in the weight vector (which is a large dictionary). Thus, to decrease runtime, we can represent w as $w = sW$, where $s \in \mathbb{R}$ and $w \in \mathbb{R}^d$. First, we verify this- note it is obvious that the update $w_{t+1} = (1 - \eta_t \lambda) w_t$ is equivalent to

$$\begin{aligned} W_{t+1} &= W_t \\ s_{t+1} &= (1 - \eta_t \lambda) s_t \end{aligned}$$

Thus, we show the update $w_{t+1} = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$ is equivalent to

$$\begin{aligned} s_{t+1} &= (1 - \eta_t \lambda) s_t \\ W_{t+1} &= W_t + \frac{1}{s_t + 1} \eta_t y_j x_j \end{aligned}$$

First, note

$$\begin{aligned} w_{t+1} &= s_{t+1} W_{t+1} \\ \frac{w_{t+1}}{s_{t+1}} &= W_{t+1} \\ \frac{(1-\eta_t\lambda)w_t + \eta_t y_i x_i}{s_{t+1}} &= W_{t+1} \end{aligned}$$

Now, letting $s_{t+1} = (1-\eta_t\lambda)s_t$ yields

$$\begin{aligned} \frac{(1-\eta_t\lambda)w_t + \eta_t y_i x_i}{(1-\eta_t\lambda)s_t} &= W_{t+1} \\ \frac{(1-\eta_t\lambda)w_t}{(1-\eta_t\lambda)s_t} + \frac{\eta_t y_i x_i}{(1-\eta_t\lambda)s_t} &= W_{t+1} \\ W_t + \frac{1}{s_{t+1}}\eta_t y_j x_j &= W_{t+1} \end{aligned}$$

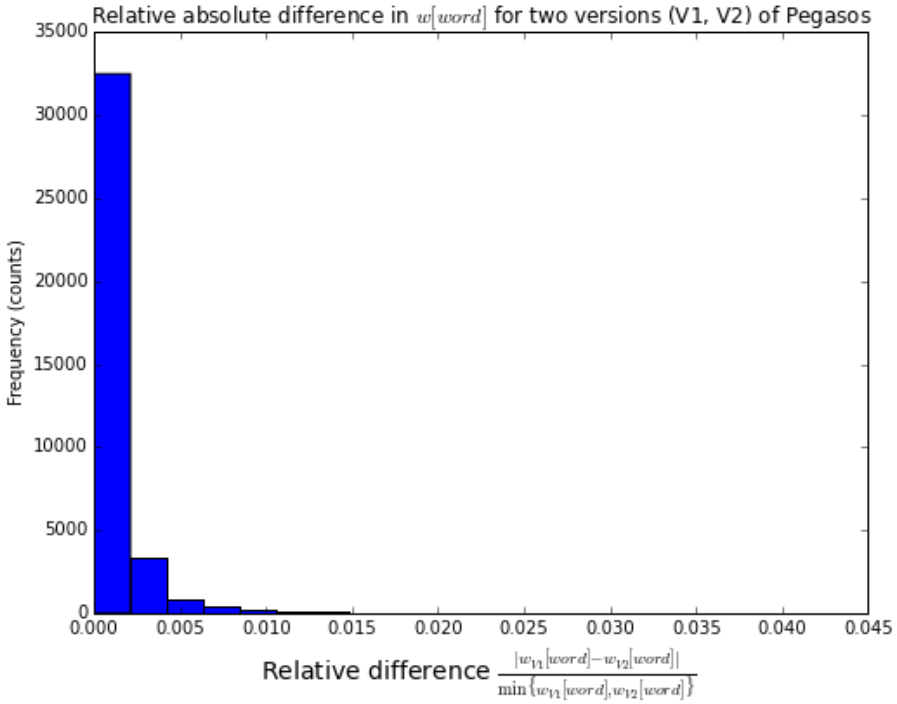
Thus, representing $w = sW$ allows us to update much more quickly (on the order of the number of non-zero entries of x_j).

6.4 Running and comparing both versions of the Pegasos algorithm

Both versions of the Pegasos algorithm were run for 10 iterations (with λ set to 0.1), and the final weight vectors w and runtimes were compared. First, the runtimes are given below:

Version of Pegasos	**Runtime**
First version (naive representation of w) including check for convergence	`1 loops, best of 3: 9min 4s per loop`
Second version (representing $w = sW$) not including check for convergence	`1 loops, best of 3: 4.12 s per loop`
Second version (representing $w = sW$) including check for convergence	`1 loops, best of 3: 7.18 s per loop`

Obviously the second version of the algorithm is a dramatic improvement over on the first. Next, to confirm the two results return essentially the same weight vector, the relative differences in weights are reported.



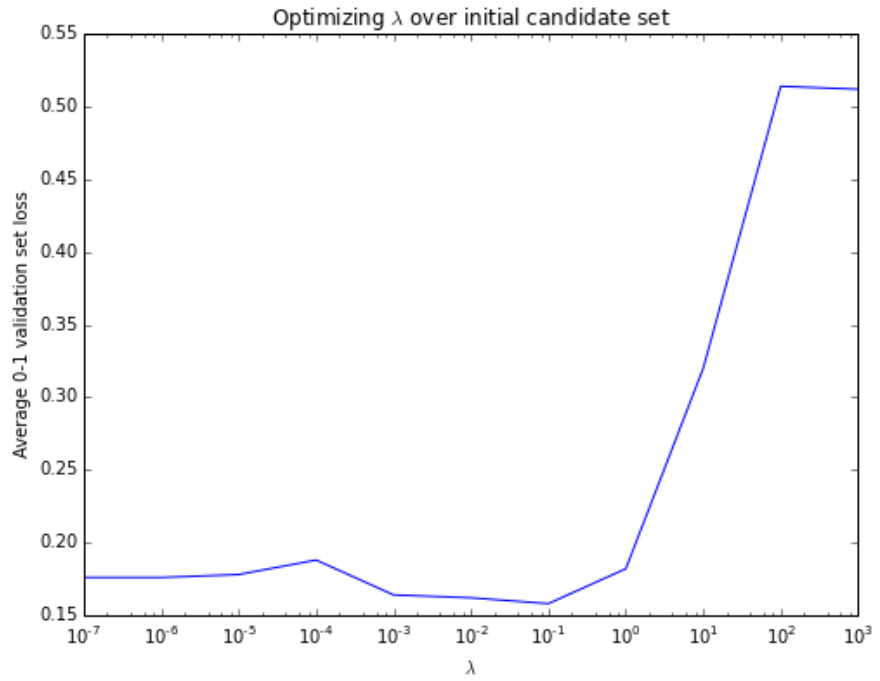
It is apparent from the small relative differences that these two implementations are returning essentially the same weight vector (up to differences errors attributable to floating point precision on $1500 \times 10 = 15000$ updates to the weight vector w).

6.5 0-1 loss of linear predictor $x \mapsto w^T x$

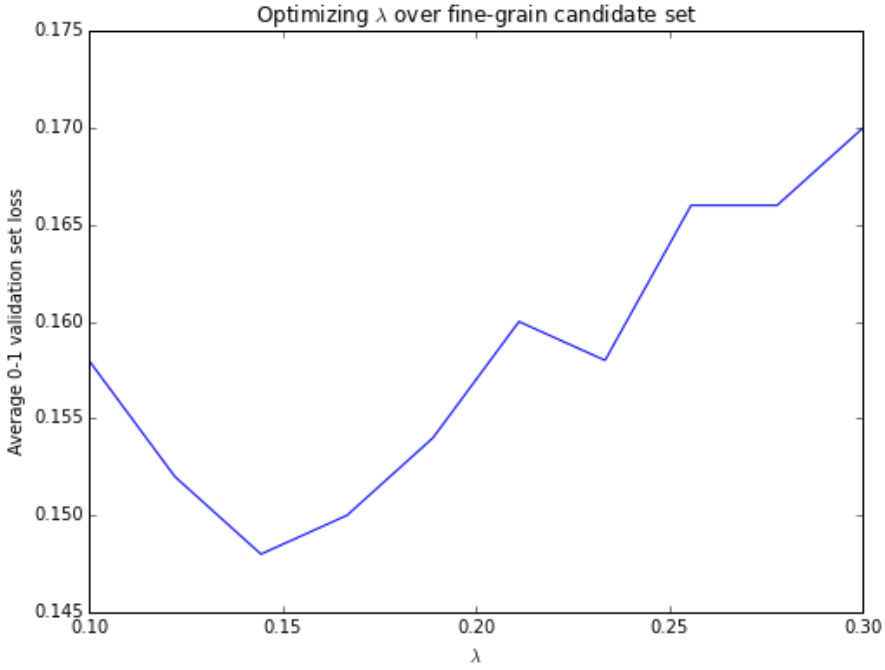
See attached iPython notebook

6.6 Optimizing λ

To find the optimal λ, the average validation set 0-1 loss was calculate for the SVM model fit with λ in $\{10^{-7}, 10^{-6}, \ldots, 10^{3}\}$.

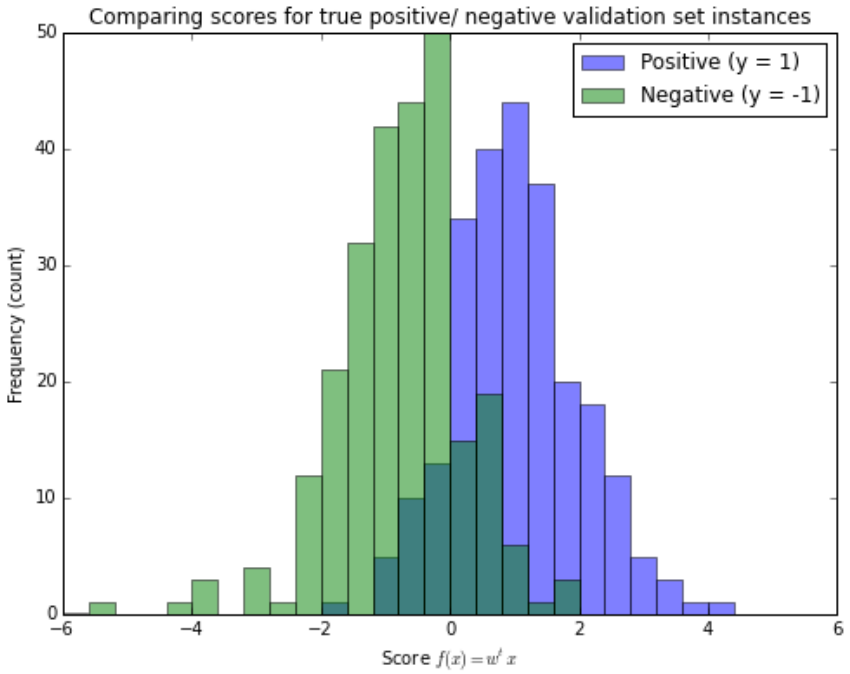


Based on model performance across this range, further exploration was conducted over the subsets of this range. Ultimately, the optimal λ value was found to be 0.14.

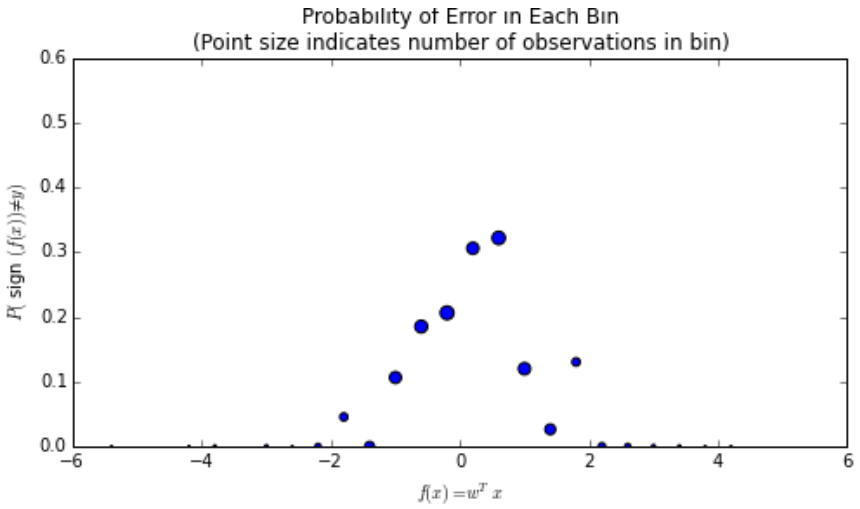


6.7 Comparing scores (i.e. $f(x) = w^T x$)

In SVM, we assume the score corresponds to the confidence of our prediction. To test this assumption, we compare the scores of true positive and negative instances from our validation set. First, histograms showing the scores for these subsets are shown below:



Next, the estimated probability of error was determined for each bin (using the relative frequency of misclassified instances). These probability estimates are shown in the plot below- note the size of the point corresponds to the number of instances in the bin.



From this plots, it's apparent that the score does in fact correspond to the confidence of our prediction (since the conditional probability of error given a large absolute scores is low).

6.8 Investigating the case $y_i w^T x_i = 1$

Our hinge loss function is not differentiable when $y_i w^T x_i = 1$. However, it is worth asking how often and when we hit this point. In our training data, we find 0 training set points with $y_i w^T x_i = 1$ This makes sense- w is a vector in $\mathbb{R}^{38793}$, with floating point entries. Thus, the probability of constructing a text string such that $y_i w^T x_i = 1$ is extremely low.

7. Error Analysis

The follow approach was taken to analyze errors made by our model:

1. First, incorrectly classified instances were ranked by the absolute score to identify the 10 "biggest" mistakes.
2. Then, for each of these 10 mistakes, the 10 features that contributed most heavily (i.e. largest $|w_i x_i|$) to the decision were identified and listed.

These mistakes are listed below:

```
Mistake 1- actual ranking = -1 , score = 1.31645450259

Word = the,  x = 90.0,  w = 0.0031,  xw = 0.2758
Word = both,  x = 5.0,  w = 0.0564,  xw = 0.2822
Word = with,  x = 13.0,  w = 0.0223,  xw = 0.2893
Word = any,  x = 4.0,  w = -0.0748,  xw = -0.2991
Word = an,  x = 8.0,  w = -0.0377,  xw = -0.3017
Word = best,  x = 5.0,  w = 0.0616,  xw = 0.3081
Word = that,  x = 20.0,  w = 0.0182,  xw = 0.3649
Word = is,  x = 25.0,  w = 0.0186,  xw = 0.4644
Word = have,  x = 8.0,  w = -0.0592,  xw = -0.4737


Mistake 2- actual ranking = 1 , score = -1.10188590288

Word = about,  x = 4.0,  w = -0.0352,  xw = -0.1407
Word = this,  x = 6.0,  w = -0.0249,  xw = -0.1494
Word = an,  x = 4.0,  w = -0.0377,  xw = -0.1509
Word = no,  x = 3.0,  w = -0.0547,  xw = -0.1642
Word = woman,  x = 3.0,  w = -0.0597,  xw = -0.1792
Word = is,  x = 10.0,  w = 0.0186,  xw = 0.1858
Word = have,  x = 4.0,  w = -0.0592,  xw = -0.2369
Word = her,  x = 11.0,  w = -0.0217,  xw = -0.2391
Word = and,  x = 11.0,  w = 0.024,  xw = 0.264
```

```
Mistake 3- actual ranking = 1 , score = -1.15931164545

Word = an,  x = 3.0,  w = -0.0377,  xw = -0.1132
Word = things,  x = 3.0,  w = 0.0435,  xw = 0.1306
Word = bad,  x = 1.0,  w = -0.1391,  xw = -0.1391
Word = on,  x = 4.0,  w = -0.0368,  xw = -0.1471
Word = with,  x = 7.0,  w = 0.0223,  xw = 0.1558
Word = and,  x = 7.0,  w = 0.024,  xw = 0.168
Word = more,  x = 8.0,  w = -0.0212,  xw = -0.1694
Word = is,  x = 12.0,  w = 0.0186,  xw = 0.2229
Word = this,  x = 9.0,  w = -0.0249,  xw = -0.224


Mistake 4- actual ranking = 1 , score = -1.17769919849

Word = performance,  x = 2.0,  w = 0.0506,  xw = 0.1012
Word = an,  x = 3.0,  w = -0.0377,  xw = -0.1132
Word = most,  x = 2.0,  w = 0.0567,  xw = 0.1133
Word = with,  x = 6.0,  w = 0.0223,  xw = 0.1335
Word = to,  x = 15.0,  w = -0.009,  xw = -0.1344
Word = acting,  x = 2.0,  w = -0.0683,  xw = -0.1365
Word = bad,  x = 1.0,  w = -0.1391,  xw = -0.1391
Word = and,  x = 6.0,  w = 0.024,  xw = 0.144
Word = this,  x = 6.0,  w = -0.0249,  xw = -0.1494


Mistake 5- actual ranking = -1 , score = 1.06030174446

Word = about,  x = 4.0,  w = -0.0352,  xw = -0.1407
Word = well,  x = 2.0,  w = 0.0738,  xw = 0.1476
Word = this,  x = 7.0,  w = -0.0249,  xw = -0.1743
Word = plot,  x = 2.0,  w = -0.0958,  xw = -0.1915
Word = computers,  x = 8.0,  w = 0.0248,  xw = 0.198
Word = is,  x = 11.0,  w = 0.0186,  xw = 0.2043
Word = what,  x = 6.0,  w = 0.0383,  xw = 0.23
Word = too,  x = 5.0,  w = -0.0465,  xw = -0.2324
Word = and,  x = 11.0,  w = 0.024,  xw = 0.264


Mistake 6- actual ranking = 1 , score = -1.70089580387

Word = off,  x = 2.0,  w = -0.056,  xw = -0.112
Word = action,  x = 5.0,  w = -0.0266,  xw = -0.133
Word = you,  x = 3.0,  w = 0.045,  xw = 0.1349
Word = bad,  x = 1.0,  w = -0.1391,  xw = -0.1391
Word = and,  x = 6.0,  w = 0.024,  xw = 0.144
Word = big,  x = 4.0,  w = -0.0401,  xw = -0.1603
```

```
Word = is,  x = 9.0,  w = 0.0186,  xw = 0.1672
Word = have,  x = 3.0,  w = -0.0592,  xw = -0.1777
Word = hit,  x = 4.0,  w = -0.0499,  xw = -0.1997


Mistake 7- actual ranking = -1 , score = 1.79500235738

Word = to,  x = 19.0,  w = -0.009,  xw = -0.1702
Word = on,  x = 5.0,  w = -0.0368,  xw = -0.1839
Word = may,  x = 3.0,  w = 0.0658,  xw = 0.1973
Word = good,  x = 4.0,  w = 0.0555,  xw = 0.2222
Word = also,  x = 4.0,  w = 0.0573,  xw = 0.2293
Word = only,  x = 2.0,  w = -0.1165,  xw = -0.2329
Word = very,  x = 4.0,  w = 0.0585,  xw = 0.234
Word = plot,  x = 3.0,  w = -0.0958,  xw = -0.2873
Word = is,  x = 20.0,  w = 0.0186,  xw = 0.3715


Mistake 8- actual ranking = -1 , score = 1.80678925035

Word = is,  x = 10.0,  w = 0.0186,  xw = 0.1858
Word = i,  x = 18.0,  w = 0.0106,  xw = 0.1901
Word = on,  x = 6.0,  w = -0.0368,  xw = -0.2207
Word = any,  x = 3.0,  w = -0.0748,  xw = -0.2243
Word = very,  x = 4.0,  w = 0.0585,  xw = 0.234
Word = have,  x = 4.0,  w = -0.0592,  xw = -0.2369
Word = sense,  x = 7.0,  w = 0.0361,  xw = 0.2525
Word = that,  x = 17.0,  w = 0.0182,  xw = 0.3102
Word = ending,  x = 6.0,  w = 0.0552,  xw = 0.3313


Mistake 9- actual ranking = -1 , score = 1.13484205563

Word = better,  x = 2.0,  w = -0.0595,  xw = -0.1191
Word = others,  x = 2.0,  w = 0.068,  xw = 0.1361
Word = to,  x = 17.0,  w = -0.009,  xw = -0.1523
Word = is,  x = 9.0,  w = 0.0186,  xw = 0.1672
Word = things,  x = 4.0,  w = 0.0435,  xw = 0.1741
Word = have,  x = 3.0,  w = -0.0592,  xw = -0.1777
Word = best,  x = 3.0,  w = 0.0616,  xw = 0.1849
Word = american,  x = 4.0,  w = 0.0818,  xw = 0.3274
Word = and,  x = 15.0,  w = 0.024,  xw = 0.36


Mistake 10- actual ranking = -1 , score = 1.85025931165

Word = an,  x = 4.0,  w = -0.0377,  xw = -0.1509
Word = to,  x = 17.0,  w = -0.009,  xw = -0.1523
Word = 2,  x = 4.0,  w = -0.0385,  xw = -0.1541
```

```
Word = the,  x = 53.0,  w = 0.0031,  xw = 0.1624
Word = there's,  x = 2.0,  w = -0.0842,  xw = -0.1683
Word = with,  x = 8.0,  w = 0.0223,  xw = 0.178
Word = of,  x = 35.0,  w = 0.0055,  xw = 0.1914
Word = is,  x = 13.0,  w = 0.0186,  xw = 0.2415
Word = and,  x = 20.0,  w = 0.024,  xw = 0.48
```

There are two observations we can make from these errors.

1. First, it's apparent frequent words are decreasing the performance of the model. Specifically, a number of high-frequency words are assigned small weights (example: `the` is assigned the weight 0.0031). While the weights are small, the frequency of these words is high in many documents, and as a result they can produce errors.

2. In addition, the model is unable to account for context-dependent differences in the meaning of words like `bad` and `good`. These words can be negated (or otherwised used in contexts that change meaning), and (given these words are assigned large negative weights, i.e. w`[bad]` $= -0.1391$) this can lead to errors.

To try to address these errors, two feature engineering approaches will be tested. First, we will test model performance after introducing bigrams. This is hypothesized to reduce error due to negation of heavily weighted terms (i.e. account for `not good`). Second, we will test model performance using a TF-IDF feature set (for the bag-of-words and bigram representation). This is hypothesized to reduce the impact (via IDF weighting) of high frequency terms (like `the`).

8. Features

8.1 and 8.2 New features

As described above, we will attempted to improve performance using the following feature:

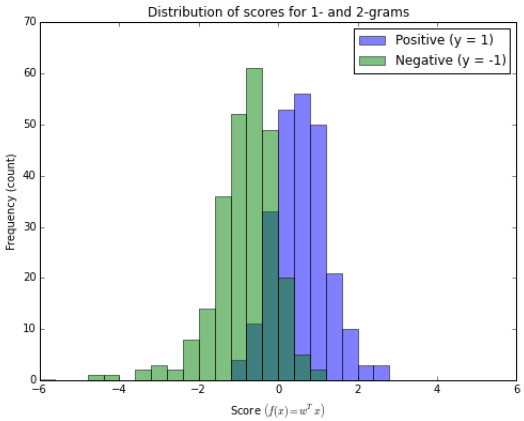
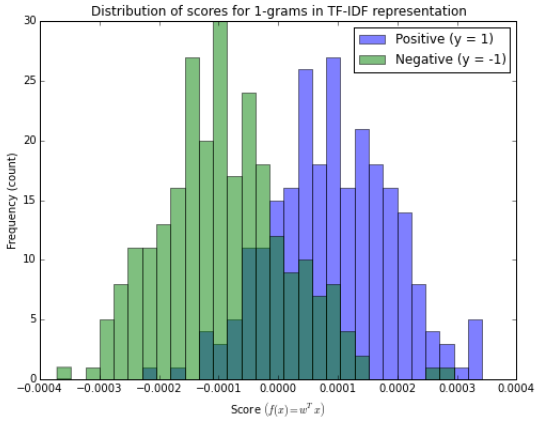
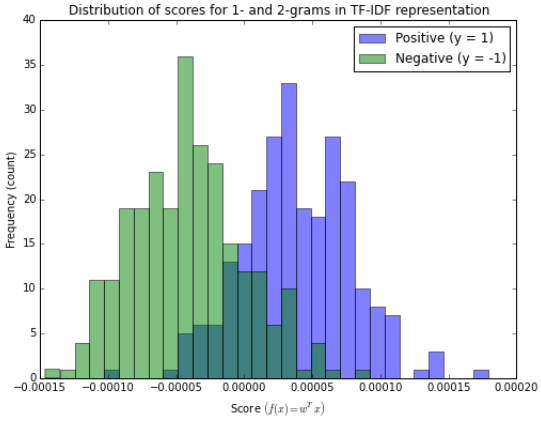
1. 1-grams (previously conducted)

2. 1-grams and bigrams.

3. TF-IDF for 1-grams

4. TF-IDF for 1-grams and bigrams

We present histograms showing the distribution of scores for the true positive and true negative ratings, as well as the estimated probabilities of errors (including 95% confidence intervals on this probability).

Feature set	**Distribution**	**Average 0-1 loss** $(\hat{p}, \hat{p} \pm 1.96 \cdot SE)$
1- and 2-grams (counts)		0.15 (0.1187, 0.1813)
1-grams (TF-IDF)		0.188 (0.1538, 0.2222)
1- and 2-grams (TF-IDF)		0.166 (0.1334, 0.1986)

Unfortunately, it does not seem that any of these alternate feature sets or representations

produced an improved model (i.e. model with lower out-of-sample average 0-1 loss). Note, however, λ was not optimized for these alternate models. If additional time was available, I would have first searched for an optimal lambda value for each of these feature spaces, then determined the validation set 0-1 loss for this optimized model.
Still, given the results available, it appears the best model is the original 1-gram model (both in terms of runtime and average 0-1 loss).