AML (CS5785) HW2

Jiwon Jeong

October 2025

1 Continuation of Housing Prices

1.1 Feature Selection

In HW1, I used OLS with scikit-learn to test several models with MSE and R^2 scores. The final model ("Top 40") had MSE score of $9.035 * 10^8$ and an R^2 score of 0.8567 (Figure 1). The OLS by scikit-learn will be reperformed in this assignment, but this discussion is necessary since the preprocessing steps and decision to choose the "Top 40" feature set comes from HW1.

The preprocessing involved these following steps.

1) Impute missing values
2) Value encode high cardinality categorical values
3) One-hot encode other categorical values
4) Scale values between 0 and 1
5) Calculate mutual information scores and select top 50 features
6) Apply domain-relevant feature combinations and feature re-additions
7) Visualize and remove low correlation features
8) Evaluate models for top 10, 20, 30, 40, 50 mutual information features

Model	Num of Features	MSE	R2
Top 10	11	1212026033.2527816	0.807821997384164
Top 20	21	1095717140.471338	0.8262638543145882
Top 30	30	1007768362.6784785	0.8402089511895913
Top 40	40	903477323.2588153	0.8567452656717244
Top 50	48	880034320.3184863	0.8604623718697642

Figure 1: Scores for models from HW1 (scikit-learn)

Regularization Param	MSE	R2
0	959260870.7847818	0.8625325334327898
1e-08	959260219.035948	0.8625326268320557
1e-07	959259957.6477271	0.8625326642904547
1e-06	959264192.113335	0.8625320574677675
1e-05	959318711.7910429	0.8625242444919641
0.0001	959846235.7953337	0.8624486473414581
0.001	965186992.8390604	0.8616832868824711
0.01	1022359458.179936	0.8534901517227079
0.1	1520962905.320697	0.7820374793708758
1	3066602330.9023113	0.5605386748931382

Figure 2: Evaluation for models from various regularization parameters

1.2 OLS by gradient descent

I split the training set into training and validation sets.

Then I defined training loops. In each loop, I use the mean squared error and an L2 regularization to define the loss function.

Thus, there are three hyperparameters: the learning rate, the regularization parameter, and the number of training loops. I did not optimize the number of training loops or the learning rate. I experimented with values and used values that showed relatively fast, good convergence. For all models, 751 training loops with a learning rate of 0.1 was more than sufficient for convergence.

With a regularization hyperparameter of 0 (meaning no L2 regularization), the MSE score was $9.593 * 10^8$ and the R^2 score was 0.8625.

1.3 Regularization hyperparameter optimization

For the remaining hyperparameter of the regularization constant, I evaluated the resulting models of various regularization constants. I did this by using the gradient descent method on the training set and evaluating on the validation set. Evaluation was based on maximizing the R^2 score, but the MSE score was also observed to decrease with this optimization.

I did this optimization once to determine the best regularization hyperparameter to be between 1e-8 and 1e-6 (Figure 2).

Then I did the optimization again for regularization constants between 1e-8 and 1e-6 (Figure 3). The final best model was determined to use a regularization parameter of 1e-8. For the final model, the MSE score was $9.593 * 10^8$ and the

Regularization Param	MSE	R2
1e-08	959259666.3105445	0.8625327060407081
2.5e-08	959260731.9833118	0.8625325533238195
5e-08	959260233.9414071	0.8625326246960197
7.5e-08	959260115.6449417	0.8625326416485671
1e-07	959260330.4884459	0.8625326108602872
2.5e-07	959261241.8481456	0.8625324802573272
5e-07	959263766.3655267	0.8625321184798178
7.5e-07	959263445.8736787	0.8625321644080984
1e-06	959265621.866733	0.8625318525760811

Figure 3: Evaluation for models from various regularization parameters between 1e-8 and 1e-6

R^2 score was 0.8625.

1.4 Re-evaluation of OLS

I re-evaluated the scikit-learn analytic solution by training the model on the training set and evaluating scores on the validation set. This was necessary, because HW1 did not split the training data into training and validation sets.

The MSE score was $9.223 * 10^8$ and the R^2 score was 0.8678.

1.5 Discussion of optimization methods

There are three optimization methods to compare: OLS closed-form (OLS), gradient descent (GD), and gradient descent with L2 regularization (GDL2). Here is a summary of the evaluation metrics for the best model produced for these methods.

OLS) MSE score = $9.223 * 10^8$ and R^2 score = 0.8678.

GD) MSE score = $9.59261 * 10^8$ and R^2 score = 0.8625325.

GDL2) MSE score = $9.59260 * 10^8$ and R^2 score = 0.8625327.

First, comparing OLS to both GD and GDL2, the analytic, closed form OLS solution proves to give the best performing model in terms of both R^2 and MSE. However, the difference is not too much.

Comparing GD to GDL2, we see an extremely marginal improvement in applying L2 regularization to gradient descent. This informs that the vanilla GD method does not easily overfit for this dataset.

While OLS proves to be the dominant method for this dataset, not all linear regressions should rely on the closed-form solution. One instance is when the

closed-form solution cannot be calculated because the design matrix, X (the matrix of features and samples), is not invertible. X would be non-invertible when X is not full rank, like when there are highly collinear features. Here, we show that GD and GDL2 are decent alternatives to OLS, so one can use these gradient descent methods when X is non-invertible.

On a conceptual level, the analytic solution fails when X in non-invertible because this implies that there are multiple (infinite) critical points on the loss function. On the other hand, gradient descent methods can work around local critical points generated from co-linearity by using a weighted average gradient descent.

2 Data Generating Distribution and Convergence of Linear Regression

2.1 Generating synthetic dataset

I generated a synthetic dataset based on a linear model with additive Gaussian noise using the following data generating process:

$$Y = \alpha + \beta X + \epsilon$$

Where each realization of is a single random value drawn from a normal distribution with mean 0 and standard deviation 20.

To generate a random input space, I generated these X values from a normal distribution as well. Each realization of X is also a single random value drawn from a normal distribution with a mean 168 and standard deviation 30.

Most importantly, I set $\alpha = 20, and \beta = 0.5$.

2.2 Running linear regression on various sample sizes

I generated datasets using the data generating process in section 2.1 of sample size $n = 10^2, 10^3, 10^4, 10^5, 10^6$. Then I ran a linear regression on each of these datasets.

Figure 4 describes the calculated parameters of and and the R^2 score.

2.3 Discussion of convergence

Running linear regression on increasing sample sizes shows that the weight coefficient converges close to 20 and the bias coefficient converges to 0.5. This shows that the calculated model coefficients converge to the actual coefficients of α and β used in the true data generating process. It seems like the R^2 score converges to 0.36, not 0.

Sample Size	Alpha	Beta	R2
10	15.023893156646935	0.42225592796682204	0.555772234438908
1000	20.167148847976847	0.49408968236306944	0.3249364610696879
10000	19.60405183787664	0.5034484653773773	0.362476837289602
100000	19.721406373732975	0.5029047086285077	0.3621654201407549
1000000	19.952240800185407	0.5003491830516813	0.3607313784983075

Figure 4: Coefficients and evaluation scores of linear regression on different sample sizes

2.4 Convergence of R^2

With the assumption that the model parameters for α and β converge to the true data generating values, we can show that the R^2 score converges to $1var(\epsilon)/var(Y)$. See Figure 5 for derivation.

2.5 Alternative models for R^2 convergence

In the above section, it was shown that an appropriate linear model will converge to $1var(\epsilon)/var(Y)$. These values are only dependent on the properties of the data generating distribution itself, and no score loss is contributed by model parameters. So it is mathematically impossible to improve from this inherent noise of the data points themselves. Thus, we cannot improve the asymptotic R^2 score.

In other words, the optimized linear model got as close to the data generating distribution as possible, with the exception of random additive gaussian noise. We saw this in the R^2 score derivation, where the numerator canceled out all terms except ϵ. Trying to deviate and get a "better" model will only add more separation between the predictions and the non-additive portion of the data generating distribution.

2.6 Important characteristics of data generating distribution

As seen in this example, the R^2 score depends on $1var(\epsilon)/var(Y)$. This means that a "poor" model will emerge if the variance of the error is extremely high compared to the variance of the label; this occurs when the noise variance is high compared to the variance in the input.

Thus, a "good" model with good evaluation metrics also depends on the underlying data generating distribution having a combination of narrow noise variance and wide distribution of inputs.

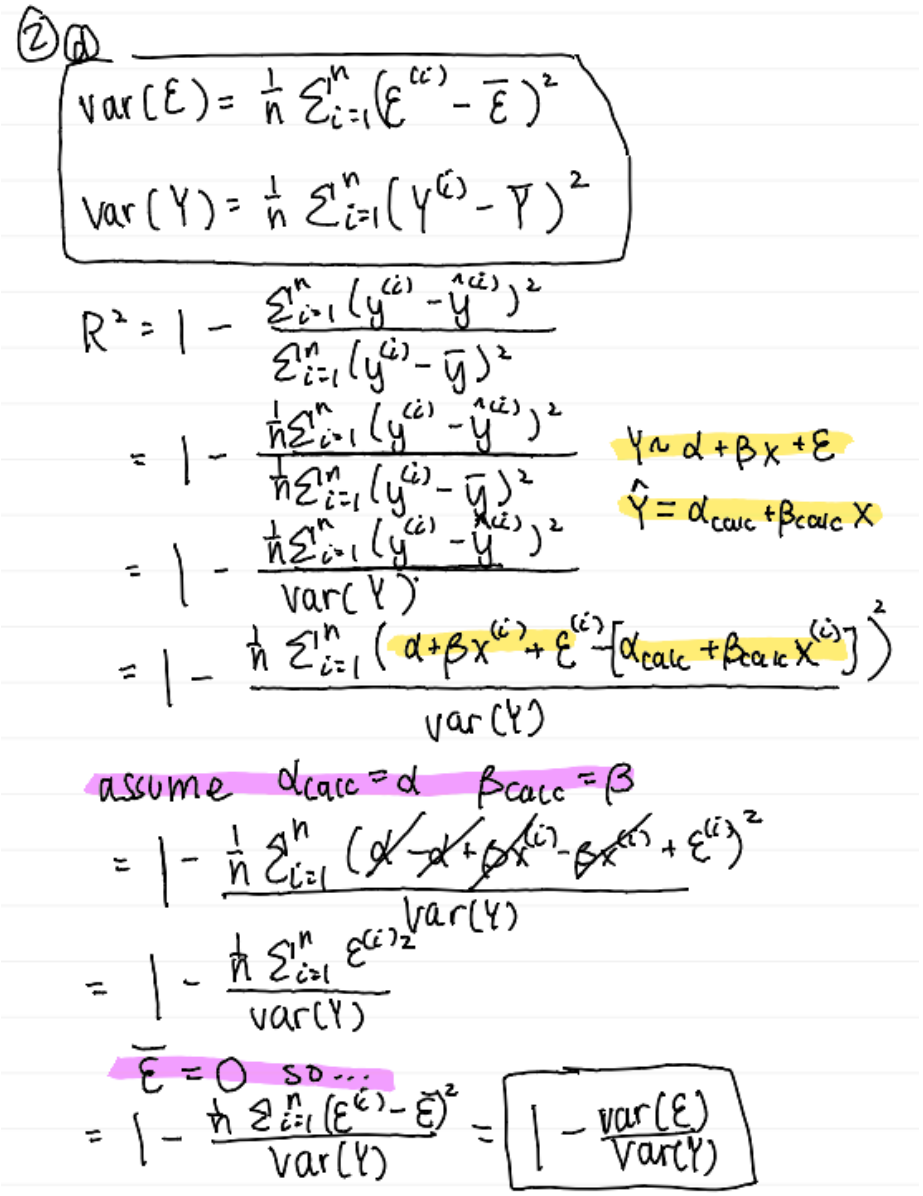

Figure 5: Convergence of R2 score

Additionally, beyond a good evaluation metric, we would ideally have a large sample size so that the model parameters can converge towards the true data generating constants.

3 Binary Classification on Text Data

3.1 Download the data

The Kaggle NLP with Disaster Tweets dataset was downloaded. There are 7613 training data samples and 3263 testing data points. Of the 7613 training data points, 3271, or 43%, are tweets about disasters. The remaining 4342, or 57%, are tweets about nondisasters (binary classification).

3.2 Split the training data

The training data was split into a training set and validation set using a random 70-30 split.

3.3 Preprocessing

The following preprocessing steps were taken on the tweet text data.

1) Convert all words to lowercase.
2) Lemmatize all words.
3) Strip punctuations (but keep @s, s)
4) Handle @username and topic tags
5) Strip stop words
6) Strip Û symbols and other non-standard characters
7) Strip link details (but keep http and https)

The following section will describe each preprocessing step.

3.3.1 Lowercase

All words were converted to lowercase. This was automatically decided because "hello" and "Hello" have the same semantic meaning. While there may be some semantic differences due to capitalization, like "Apple" and "apple", most capitalization differences are only driven by sentence start capitalizations.

3.3.2 Lemmatize

Lemmatizing means to turn all words into their root. This was done to converge words into their semantic meaning. This was done with nltk.stem. It is important to lemmatize words to properly group semantically same words. If we kept words nonlemmatized, then our feature set would be incredibly high cardinality (leading to memory issues) and less represented per feature (potential overfit issue).

3.3.3 Punctuations

Punctuations were also removed since "apple.", "apple," and "apple" all have the same semantic meaning. So we can lower cardinality and improve representation per feature like the lemmatizing step.

3.3.4 Twitter tags

Particularly with Twitter tweets, many text had @username or topic attached. I compared the frequency of @ and # tags on disaster vs nondisaster tweets. For @username tags, found that 20.7% of disaster tweets had tags and 31.4% of nondisaster tweets had # tags. For topic tags, found that 26.8% of disaster tweets had tags and 20.4% of nondisaster tweets had # tags. I thought these differences were significant enough that I kept the tags as tokens. So the data set would identify if a text contained @ or .

For @username tags, the username portion is not semantically relevant, so I removed the username. However, for #topic, the topic text is very relevant to the tweet, so I kept it (but separated it from as its own token).

3.3.5 Strip stop words

All common stop words were removed. Stop words are semantically empty and common words like "the", "a", and "me". Once again, removing semantically irrelevant words reduces feature cardinality and improves the number of samples for each feature. These positively improve issues in memory and overfitting.

3.3.6 Strip non-standard characters

There are some odd square-Û symbols in the dataset that likely correspond to emojis. However, it seems that all non-standard characters were recorded the same way in the Kaggle dataset, so all the different emoji semantics got lost. We only keep the fact that an emoji or some other non-standard character was used. Since the semantic meaning of emojis were lost, I removed these non-standard characters.

3.3.7 Strip link details

The Kaggle dataset also included links. The link details are not useful to the model, because the url is not lemmatizable to a semantic root nor is it common for the same url to be repeated across many tweets. However, I was curious if the presence of a link itself was different between tweet classes. For http: links, 62.9% of disaster tweets included a link while only 35.7% of nondisaster tweets included a link. For https: links, 4.1% of disaster tweets included a link while 6.3% of nondisaster tweets included a link. I thought these differences were significant enough to keep the http or https as a token (thus allowing the model to know if a link was included). I decided to tokenize the two link types separately due to the different distributions seen (http: links are more popular

for disaster tweets and https: links are more popular for nondisasater tweets); this also makes sense, since https: links have different rules than http: links.

3.4 Bag of words model

I built a bag of words with different threshold of M. This means that a word needs to appear in at least M different tweets to be included in the bag of words. A bag of words model uses these d words that meet the threshold criteria to create features as a d-dimensional vector. Each element of this vector would be 0 or 1, depending on if that feature did not have that word or not. For example, for 3 words, there would be 8 different features (0,0,0), (0,0,1), (0,1,0) ... (1,1,1). And a specific realization (a specific tweet) would be converted to one bag of word vector and match only one feature.

As seen with the 3 word example, it is very easy for the dimensionality of the features to grow, so it was important to choose a minimum threshold M that was as high as possible without losing model predictive power. To choose M, I counted the frequency of each word and discovered that 6084 words only appeared once, 1310 words appeared only twice, 636 words only appeared three times, and 391 words only appeared four times. There on, the frequency only continues to decrease. As seen, setting M=2 would result in a significant removal of 6084 words. This is a good idea as well, since tokens that only appear once are probabilistically unlikely to represent the data generating distribution well. Removing these single-frequency words reduces likelihood of overfitting. I also decided to remove words that only appear twice for the same reason and for even more memory efficiency. However, deciding to choose M=3,4,5,6, or 7 was unclear, so I decided to create models on all these M values to evaluate the models on a later step.

3.5 Logistic regression

3.5.1 Regression without regularization

A logistic regression without regularization was performed on each of the 5 feature sets created by M=3,4,5,6,7. For all feature sets, the F1 score on the training set was very high (¿0.89), but the F1 scores on the validation set was not very high (¡ 0.71). This is a sign that the models have good training loss, but poor generalization (also known as overfitting).

3.5.2 Regression with L1 regularization

A logistic regression with L1 regularization was performed on each of the 5 feature sets created by M=3,4,5,6,7. Once again, the F1 score on the testing set is higher than the scores on the validation set, but the difference is now much less. Specifically, we see the testing F1 scores drop and validation F1 scores rise. Thus, with regularization, the overfitting has been reduced.

Minimum M	F1 Test	F1 Validate
3	0.9763469119579501	0.686848635235732
4	0.9750219106047326	0.6807258460029426
5	0.9690857268142951	0.6548323471400395
6	0.928964152188256	0.686
7	0.8928729526339088	0.7071320182094082

Figure 6: F1 scores on training and validation sets for no regularization models

Minimum M	F1 Test	F1 Validate
3	0.8449791763072652	0.7479586281981492
4	0.841009025688498	0.7472766884531591
5	0.8351394973483975	0.7474198804997284
6	0.8271719038817006	0.737527114967462
7	0.8246993524514339	0.741304347826087

Figure 7: F1 scores on training and validation sets for L1 regularization models

3.5.3 Regression with L2 regularization

A logistic regression with L2 regularization was performed on each of the 5 feature sets created by M=3,4,5,6,7. Similar to the L1 regularization scores, there is less overfitting compared to the no regularization scores. Additionally the scores for the L2 regularizations are better than the scores for the L1 regularization, showing that the L1 regularization underfits slight more than L2 regularization.

3.5.4 Discussion of regularization

The training set score was best with no regularization. The validation set score was best on the L2 regularization. As discussed, overfitting was observed on the

Minimum M	F1 Test	F1 Validate
3	0.8846594817702362	0.7537796976241901
4	0.8709232889297198	0.7463493780421849
5	0.861807312025753	0.7486515641855448
6	0.8475356978350991	0.7415002698327037
7	0.8395118581625605	0.7431340872374798

Figure 8: F1 scores on training and validation sets for L2 regularization models

no regularization models (high training score but low validation score). For all feature sets, the F1 score on the testing set was very high (¿0.89), but the F1 scores on the validation set was not very high (¡ 0.71).

Adding either L1 or L2 regularization helped reduce overfitting. This was observed in both the L1 and L2 regularization as an increase in the validation (generalization) score from around 0.68 to around 0.74. As a byproduct, the training score also dips a little.

To investigate the effect of regularization further, I tested various regularization coefficient hyperparameters on each of the feature sets. A high coefficient means less regularization. As expected, adding more and more regularization increases the validation F1 score while decreasing training F1 score, showing a reduction in overfitting. However at some point with very high regularization, the F1 validation score also begins to drop and the F1 training score is low. This is a sign of underfitting due to a too-extreme regularization constant. Surprisingly the optimal regularization hyperparameter was 1, the default. The best model of all of these hyperparameters was for the M=3 feature set using L2 regularization with C=1. The F1 validation score was 0.754.

3.5.5 Inspecting weights of L1 regularization

I observed the weights of the best L1 regularization model. As expected with the sparsity of L1, the weight vector had a lot of 0 values, meaning many features had zero effect on a positive classification. I investigated the top few weights and decomposed the bag of words feature to get what words are in that feature. Some of these words are expected to be correlated with disaster tweets like "crash", "bomb", and "deton". However, it is important to keep in mind that the bag of words approach does not treat each word independently. So "crash" is only a strong indicator of a disaster tweet when together with "http", "mansehra", "major" and "pilot".

4 N-gram model

I trained a model using the 2-gram model as well. Some examples of 2-grams were "abandon aircraft", "abbswinston zionist", "abc news", "about cabl", "about trap", "access secret", "accid expert", "accid http", "accid indian", and "accid man". Based on the feature set study on the bag of words model, the threshold M=3 was the best feature set for the F1 validation score. Thus, I expected a similar result, but I still chose to test a few different feature sets with threshold M=3,4,5 to confirm.

For threshold M=3, there are 1779 2-grams. These only include the two word tokens. Combined with the 1-grams, the threshold M=3 has 4663 grams. For M=4, there are 1101 2-grams and 3353 total 1- and 2-grams. For M=5, there are 762 2-grams and 2632 total 1- and 2-grams. As seen, there is a choice to train only on the 2-grams or also include the 1-grams. Thus I created another feature set to evaluate. With 3 different threshold values and two gram-related

Minimum M	Regularization Type	Regularization Constant	F1 Test	F1 Validate
3	None	N/A	0.9763469119579501	0.68684835235732
3	L1	100	0.9763469119579501	0.6783391695847923
3	L1	10	0.9671878440872055	0.7024586051179127
3	L1	1	0.8449791763072652	0.7479586281981492
3	L1	0.1	0.635489043292357	0.6227544910179641
3	L1	0.01	0.5946053153510512	0.5968194574368568
3	L2	100	0.9719298245614035	0.687624750499002
3	L2	10	0.9506309497454063	0.72
3	L2	1	0.8846594817702362	0.7537796976241901
3	L2	0.1	0.7890794877989853	0.7407407407407407
3	L2	0.01	0.6855952070851784	0.6701631701631702
4	None	N/A	0.9750219106047326	0.6807258460029426
4	L1	100	0.9723562966213252	0.6760140210315475
4	L1	10	0.9555604687154543	0.7060020345879959
4	L1	1	0.841009025688498	0.7472766884531591
4	L1	0.1	0.635489043292357	0.6227544910179641
4	L1	0.01	0.5946053153510512	0.5968194574368568
4	L2	100	0.965941551307405	0.6919575113808801
4	L2	10	0.931934835974113	0.7182952182952183
4	L2	1	0.8709232889297198	0.7463493780421849
4	L2	0.1	0.7836398838334947	0.7407407407407407
4	L2	0.01	0.6838810641627543	0.6666666666666666
5	None	N/A	0.9690857268142951	0.6548323471400395
5	L1	100	0.9588377723970944	0.6780684104627767
5	L1	10	0.934984520123839	0.6998982706002035
5	L1	1	0.8351394973483975	0.7474198804997284
5	L1	0.1	0.635489043292357	0.6227544910179641
5	L1	0.01	0.5946053153510512	0.5968194574368568
5	L2	100	0.9464875578066505	0.6950354609929078
5	L2	10	0.9127426913635349	0.7217030114226376
5	L2	1	0.861807312025753	0.7486515641855448
5	L2	0.1	0.7786666666666666	0.7374929735806833
5	L2	0.01	0.6809287764153404	0.667054714784633
6	None	N/A	0.928964152188256	0.686
6	L1	100	0.9244031830238727	0.6961770623742455
6	L1	10	0.9077951002227171	0.7155963302752294
6	L1	1	0.8271719038817006	0.737527114967462
6	L1	0.1	0.635489043292357	0.6227544910179641
6	L1	0.01	0.5946053153510512	0.5968194574368568
6	L2	100	0.91672218520986	0.7115384615384616
6	L2	10	0.8922732362821949	0.7238883143743536
6	L2	1	0.8475356978350991	0.7415002698327037
6	L2	0.1	0.7746786320640311	0.7393258426966293
6	L2	0.01	0.6798748696558915	0.6662783925451369
7	None	N/A	0.8928729526339088	0.7071320182094082
7	L1	100	0.8920888888888888	0.7065989847715736
7	L1	10	0.8824843610366399	0.7142124166239097
7	L1	1	0.8246993524514339	0.741304347826087
7	L1	0.1	0.635489043292357	0.6227544910179641
7	L1	0.01	0.5946053153510512	0.5968194574368568
7	L2	100	0.8869565217391304	0.7123287671232876
7	L2	10	0.8760703019378098	0.7254800207576544
7	L2	1	0.8395118581625605	0.7431340872374798
7	L2	0.1	0.7709443099273607	0.7365470852017937
7	L2	0.01	0.6784876140808345	0.6658905704307334

Figure 9: F1 training and validation scores across hyperparameters

Weight	Bag of Words Feature
2.8366367976830786	{'bodi', 'danc', 'cours', 'themselv', 'collis', 'rather', 'find', 'space', 'sometim'}
2.7727953216726107	{'http', 'crash', 'mansehra', 'major', 'pilot'}
3.2846664209958076	{'galact', 'http', 'structur', 'crash', 'investig', 'copilot', 'said', 'failur', 'spaceship', 'virgin', 'caus', 'after'}
3.4443852636588868	{'16yr', 'bomb', 'http', 'armi', 'pkk', 'releas', 'bomber', 'suicid', 'who', 'turkey', 'pic', 'old', 'trench', 'deton'}
3.2040133064468885	{'http', 'apollo', 'feat', 'brown', 'deton'}

Figure 10: Decomposed bag of words for top weights in L1 regularized model

chocies for each, there are 6 feature sets to test. However, due to memory constraints I decided to only test L2 regularizations. L1 regularizations required much more computational power (likely due to non-differentiability of absolute values). This choice likely is not significant, because L2 regularization models consistently performed better than L1 regularizations on the bag of word models.

The results were surprising, because models trained only on 2-grams had an F1 score of 0 on validation sets while having decent F1 scores on training sets. This means that they had terrible generalization and overfitting was seen. I believe that this happened, because 2-grams like "about trap" frequencies are much rarer than just "about" or "trap", which leads to each feature having one or few representatives. With poor representation on each feature, it is very easy to overfit. However, this alone does not explain the whole picture, since we observe F1 scores of 0 even when the regularization constant is extreme and training F1 scores drop. I believe that a feature set on 2-grams alone cannot work well on generalized samples, because the features created from 2-gram tokens are too specific.

For the models trained on both 1- and 2-grams, they behaved similar to the bag of words models. Similarly, overfitting was observed on weak or no regularization models and underfitting was observed on extreme regularization models. The best N-gram model was with the feature set using M=4, a combination of both 1 and 2-grams, and trained with an L2 regularization with constant of 1. The validation set F1 score is 0.743. This model is slightly worse than the best bag of words model. Thus, we see that the addition of 2-grams does not significantly help the classification of disaster tweets. This suggests that sequence-dependent words (at least for adjacent words) do not affect the semantic understanding of a tweet as disaster or not.

4.1 Determine performance with test set

The best model of overall was with the M=3 feature set using L2 regularization with C=1 with the bag of words model. The model was rebuilt using the entire training data. Recall that when this model was trained on the partial testing set, the F1 validation score was 0.75378.

When trained with the whole training set and submitted to Kaggle, the F1 score was 0.78577. So the F1 testing score was very similar to the validation score as expected. It was slightly higher. This very slight increase is likely because the validation set we used was only 2284 samples, but the testing set is 3263 samples. It is possible that our smaller validation set was a worse representation of the true distribution, but this slight variance could have been in either direction (so it is coincidence that the final testing score is higher). A more likely reason for the increase in F1 score is that the F1 validation score is based off predictions from the model trained from only 70% of the training data. However, the F1 testing score from Kaggle is based off predictions from the model trained from 100% of the training data. Greater number of sample helps the model determine the parameters closer to the true data generating model.

Minimum M	Grams	Regularization Type	Regularization Constant	F1 Test	F1 Validate
3	2	None	N/A	0.7143985259278758	0.0
3	2	L2	100	0.7135306553911205	0.0
3	2	L2	10	0.7119723918237324	0.0
3	2	L2	1	0.6829402084476138	0.0
3	2	L2	0.1	0.5481390341433405	0.0
3	2	L2	0.01	0.23100537221795855	0.0
3	1 & 2	None	N/A	0.9794490599038042	0.6827021494370522
3	1 & 2	L2	100	0.9739092304319228	0.6839110191412312
3	1 & 2	L2	10	0.962422634836428	0.7139852786540484
3	1 & 2	L2	1	0.8973418881759854	0.7409440175631175
3	1 & 2	L2	0.1	0.7995163240628779	0.7374429223744292
3	1 & 2	L2	0.01	0.6885072655217965	0.6556213017751479
4	2	None	N/A	0.6532302595251243	0.0
4	2	L2	100	0.6524861878453039	0.0
4	2	L2	10	0.6490545050055617	0.0
4	2	L2	1	0.6299346776483954	0.0
4	2	L2	0.1	0.5214397496087637	0.0
4	2	L2	0.01	0.2228218966846569	0.0
4	1 & 2	None	N/A	0.9752681111840665	0.6812627291242362
4	1 & 2	L2	100	0.9710526315789474	0.688911704312115
4	1 & 2	L2	10	0.9496339028178389	0.7175974710221286
4	1 & 2	L2	1	0.8814338235294118	0.743139407244786
4	1 & 2	L2	0.1	0.7932203389830509	0.7310108509423187
4	1 & 2	L2	0.01	0.6876487701666226	0.6548463356973995
5	2	None	N/A	0.6153846153846154	0.0
5	2	L2	100	0.6147308781869688	0.0
5	2	L2	10	0.6109839816933639	0.0
5	2	L2	1	0.5970322956066337	0.0
5	2	L2	0.1	0.5031725888324873	0.0
5	2	L2	0.01	0.2187017001545595	0.0
5	1 & 2	None	N/A	0.9710144927536232	0.6601842374616171
5	1 & 2	L2	100	0.9574983483814138	0.6933744221879815
5	1 & 2	L2	10	0.9298480786416443	0.7238295633876907
5	1 & 2	L2	1	0.8710495963091118	0.7413509060955519
5	1 & 2	L2	0.1	0.7858183584264206	0.7297605473204105
5	1 & 2	L2	0.01	0.6841269841269841	0.6548463356973995

Figure 11: F1 scores for N-gram models

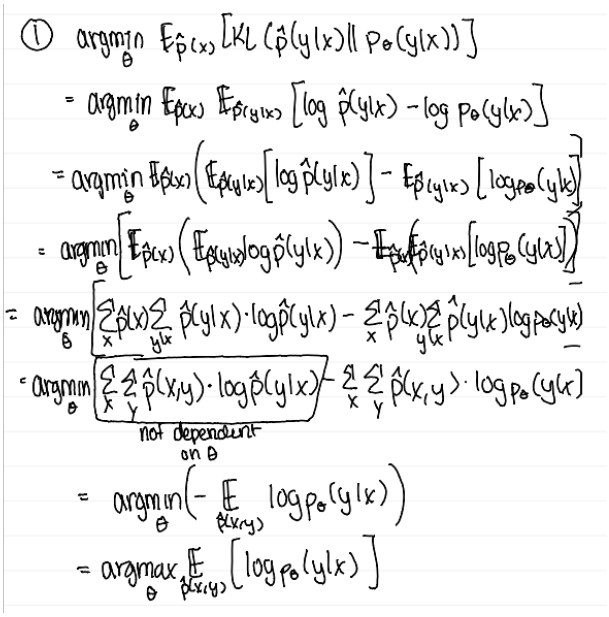

Figure 12: Derivation for equality of minimizing KL divergence and maximum likelihood estimation

5 Written Exercises

5.1 Maximum likelihood and KL divergence

Maximizing the likelihood that the model generates the data distribution and minimizing the KL divergence between the data distribution and the model predictions are the same. See Figure 12 for derivation.

5.2 Gradient and log-likelihood for logistic regression

See Figure 13 and 14 for derivation on gradient of the log likelihood loss function.

5.3 Problem with single learning rate

Using a single learning rate for all components in a gradient descent means that all dimensions will use the same learning rate. This can be an issue because a too-high learning rate can break convergence, causing either divergence or jittering. Especially with higher dimensions, a single learning rate might mean that some dimensions converge well, but other dimensions cannot converge. One naive solution would be to just lower the single learning rate until all these dimensions converge, but that could mean that some dimensions now converge too slowly and will not reach convergence in a reasonable time.

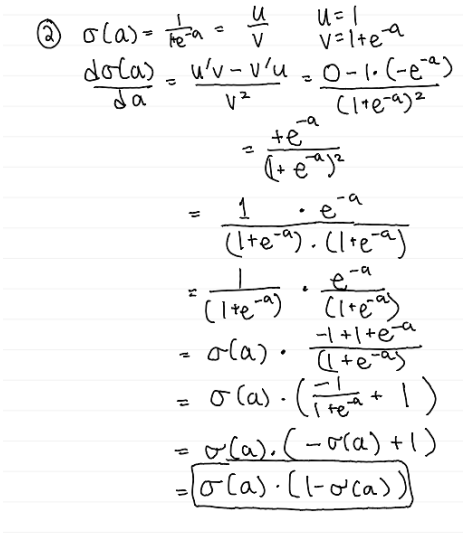

Figure 13: Derivative of sigmoid function

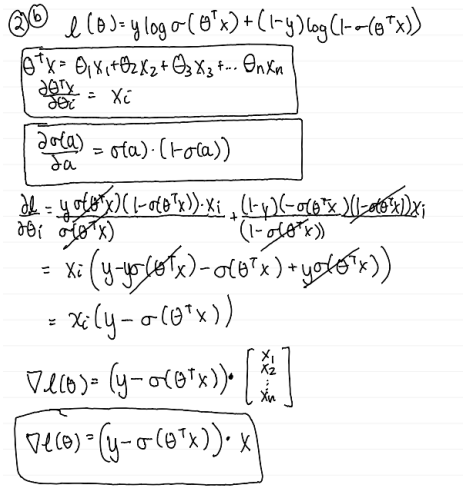

Figure 14: Gradient of log likelihood

5.4 Gradient descent can fail to reach global minimum

Previously alluded, gradient descent can fail to converge if the learning rate for that dimension is too high. There are two versions. If step size is extremely large, then the global minimum cannot be reached, because the steps will cause the solution to overstep and shoot out of convex loss surfaces. This is called divergence. Alternatively, the step size can still be large, causing the solution to step across the minimum to the other size of the convex surface and keep stepping around the minimum in the convex surface. While it may eventually converge, it will take a very long time. This is called jittering.

However, gradient descent can also fail to reach global minima under other conditions. If the learning rate is too low and/or the loss surface is very flat in regions away from the minimum, then the solution will step so slowly that the minimum is never reached in realistic time. Particularly concerning this idea of the loss surface being very flat, the loss surface can have flat points or stationary points if there are many collinear features.

Thus, ideally the step size is optimized to not be too large for divergence/jittering or too small for unrealistic convergence times. In fact for convex loss surfaces, there is an optimal step size for which we can reach the minimum in one step.

However, even gradient descents with optimized step sizes can still fail to reach a global minimum. This can happen for two main reasons. First, if there are local minima, it is possible that the specific initialization brings the solution to converge to a local minimum. This is the most concern for nonconvex loss surfaces. Second, if parts of the loss surface is nondifferentiable, the vanilla computation for gradient descent can get stuck at nondifferentiable points.

5.5 Decaying learning rate

When getting close to a (hopefully global) minimum and within the local convex surface, we need a small step size to guarantee convergence. Otherwise, a large step size could cause divergence or very slow convergence through jittering.

However, having a small step size throughout the whole optimization can be a bad idea, because we want to quickly approach the minima. The loss surface might be relatively flat when farther away from the minima, so a larger step size is necessary to reach the minima quickly. In addition, for complex non-convex loss surfaces, a large initial step size can help escape local minima.

Thus, we initially set a high learning rate to take faster steps towards the minima and possibly escape local minima. Then we gradually decrease the learning rate, so that it is eventually possible to converge in a local convex surface towards the (hopefully global) minimum. This change in learning rate is typically performed through 3 decay schedules. Starting from a large learning rate of greater than 2, the learning rate is decreased with each training loop iteration. The linear decay schedule decreases rate based on number of iterations in a $1/x$ rational decay. The quadratic decay schedule decreases rate based on number of iterations in a $1/x^2$ rational decay. The exponential decay schedule decreases rate based on number of iterations in a $1/e^x$ exponential decay.

Using these decaying learning rates means that convergence will initially not occur. There will be divergence away from (local) minima and large step sizes. With iterations, the steps will get smaller and hopefully allow the solution to approach a minima. It is important to choose the right decay schedule so that the step size does not decay too quickly (leading to slow optimization from too small steps) or decay too slowly (leading to slow optimization from unnecessary divergence/jittering).