ISYE 6740 Homework 4

Nick DiNapoli, ndinapoli6@gatech.edu

Due: October 17, 2021

1 Optimization

I consider a simple logistic regression problem with m training samples (x^i, y^i), $i = 1$, ...,m where each sample has only one feature and y^i is binary. I aim to solve the optimization problem shown in Equation 1, where θ is the parameter I wish to find and $\ell(\theta)$ is the log-likelihood function shown in Equation 2.

$$\max_{\theta} \ell(\theta) \tag{1}$$

$$\ell(\theta) = \sum_{i=1}^{m} \{-log(1 + e^{-\theta x^i}) + (y^i - 1)\theta x^i\}. \tag{2}$$

1.1 Derivation

In this section I will show how to derive the gradient of the cost function $\ell(\theta)$. Following Equation 1, because I aim to find the θ that maximizes the log-likelihood function, I take the derivative of Equation 2 with respect to θ,

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{i=1}^{m} \{-log(1 + e^{-\theta x^i}) + (y^i - 1)\theta x^i\}. \tag{3}$$

I can move the partial derivative term inside the sum and distribute to each term,

$$\frac{\partial \ell(\theta)}{\partial \theta} = \sum_{i=1}^{m} \frac{\partial}{\partial \theta}(-log(1 + e^{-\theta x^i})) + \frac{\partial}{\partial \theta}(y^i - 1)\theta x^i. \tag{4}$$

I use the chain rule to to evaluate the derivative of the log term with respect to θ but the second term is much more simple to evaluate as it is linear in θ,

$$\begin{aligned} \frac{\partial \ell(\theta)}{\partial \theta} &= \sum_{i=1}^{m} -\frac{1}{1 + e^{-\theta x^i}} \frac{\partial}{\partial \theta}(1 + e^{-\theta x^i}) + (y^i - 1)x^i \\ &= \sum_{i=1}^{m} \frac{x^i e^{-\theta x^i}}{1 + e^{-\theta x^i}} + (y^i - 1)x^i. \quad \checkmark \end{aligned} \tag{5}$$

1.2 Gradient descent

Gradient descent is used to find the optimizer θ^* because setting the gradient in Equation 5 to zero does not lead to a closed form solution. The following is pseudo-code for performing gradient descent:

```
1. Randomly initialize parameter $\theta^0$ (could draw from normal distribution)
2. Set the tolerance, $\epsilon$, and step size/learning rate, $\eta$
3. Initialize how to store previous $\theta$ ($\theta^t$) or all previous $\theta$'s ($t$ is iteration number)
4. Loop:
   (a) $\theta^{t+1} \leftarrow \theta^t + \eta_t \sum_{i=1}^{m} \frac{x^i e^{-\theta x^i}}{1+e^{-\theta x^i}} + (y^i - 1)x^i$ ("+" sign because in this problem I aim to maximize $\ell(\theta)$ or minimize the negative log likelihood)
   (b) Store $\theta^{t+1}$ following Step 3.
   (c) Check $||\theta^{t+1} - \theta^t|| < \epsilon$
       i. If the inequality is true, terminate and return $\theta^{t+1}$
```

1.3 Stochastic gradient Descent

Another option is to use the stochastic gradient descent algorithm (SGD) to solve the optimization problem in Equation 1. In gradient descent, all of the training data, (x^i, y^i), is used to estimate the gradient at every iteration. This can be computationally expensive as the training set size increases. But in SGD, it exploits the fact that the gradient de-couples, each iteration is a sum over individual samples. Now in SGD, at each iteration, a different, small subset (S_k) of the data is used to evaluate the gradient. Additionally, if the convergence condition is not satisfied over one loop of the training data, one can reduce the size of S_k and re-loop over the data. When the gradient is evaluated using a subset of the data in SGD, noise will be introduced so to counter this fact, a decreasing step size is used and the algorithm has converged when $||\nabla\ell(\theta)||$ is small. The following is pseudo-code for performing SGD:

```
1. Randomly initialize parameter $\theta^0$
2. Set the tolerance, $\epsilon$, and step size constant, C, such that $\eta$ = C/t
3. Initialize how to store previous $\theta$ ($\theta^t$)
4. Randomly sample k subsets of points from training data without replacement.
5. Loop (through $S_k$'s):
   (a) $\theta^{t+1} \leftarrow \theta^t + \eta_t \sum_{i \in S_k} \sum_i \frac{x^i e^{-\theta x^i}}{1+e^{-\theta x^i}} + (y^i - 1)x^i$
   (b) Store $\theta^{t+1}$ following Step 3.
   (c) Check $||\nabla\ell(\theta)|| < \epsilon$
       i. If the inequality is true, terminate and return $\theta^{t+1}$
6. If termination condition not met, return to step 4 using a different sampling or adjust C or $\epsilon$ and re-loop.
```

1.4 Hessian matrix

In this simple logistic regression problem there is only one feature for each sample so the Hessian matrix, H_θ, is just a scalar. The Hessian matrix,

$$H_\ell = \left(\frac{\partial^2 \ell}{\partial \theta^2}\right) \tag{6}$$

can be found by using the result in Equation 5. Now I take the second derivative of the log likelihood with respect to θ. I start with Equation 5 and take the derivative of both sides with respect to θ to find the second derivative,

$$\frac{\partial^2 \ell(\theta)}{\partial \theta^2} = \frac{\partial}{\partial \theta} \sum_{i=1}^{m} \frac{x^i e^{-\theta x^i}}{1 + e^{-\theta x^i}} + (y^i - 1)x^i \tag{7}$$

and now I pull the derivative inside the sum and distribute to each term,

$$\frac{\partial^2 \ell(\theta)}{\partial \theta^2} = \sum_{i=1}^{m} \frac{\partial}{\partial \theta} \frac{x^i e^{-\theta x^i}}{1 + e^{-\theta x^i}} + \frac{\partial}{\partial \theta}(y^i - 1)x^i. \tag{8}$$

The second term in Equation 8 has no dependence on θ and I used the quotient rule on the first term to compute the derivative,

$$\begin{aligned}
\frac{\partial^2 \ell(\theta)}{\partial \theta^2} &= \sum_{i=1}^{m} \frac{\partial}{\partial \theta} \frac{x^i e^{-\theta x^i}}{1 + e^{-\theta x^i}} \\
&= \sum_{i=1}^{m} \frac{(-(x^i)^2 e^{-\theta x^i})(1 + e^{-\theta x^i}) - (x^i e^{-\theta x^i})(-x^i e^{-\theta x^i})}{(1 + e^{-\theta x^i})^2} \\
&= \sum_{i=1}^{m} \frac{(-(x^i)^2 e^{-\theta x^i}) - ((x^i)^2 e^{-2\theta x^i}) + ((x^i)^2 e^{-2\theta x^i})}{(1 + e^{-\theta x^i})^2} \\
&= -\sum_{i=1}^{m} \frac{(x^i)^2 e^{-\theta x^i}}{(1 + e^{-\theta x^i})^2}.
\end{aligned} \tag{9}$$

Because of the squared terms and the nature of the exponential function, what is after the summation will always be positive and therefore the second derivative is negative because of the negative sign out front. Lastly, the reason why this problem can be efficiently solved and using gradient descent will achieve a global optimizer is because I just proved this is a convex optimization problem and for convex optimization, any local optimum is also a global optimum.

2 Comparing classifiers

In this section, I use two classic datasets, the divorce classification dataset and the MNIST handwritten digit dataset, to compare the quality and effectiveness of several different classifier models. I focus on classifiers: Naive Bayes, Logistic Regression, k-Nearest Neighbors (KNN), Support Vectors Machine (SVM), and Neural Networks.

2.1 Divorce classification/prediction

I use the divorce dataset, consisting of 170 participants who documented personal information, which includes 54 predictors in the attempt to predict if the subject got divorced. This dataset does have added noise such that it is not identical to the publicly available version and I have the ground truth values for each data point. For this dataset I compare three classifiers: Naive Bayes, Logistic Regression, and KNN. I first train the three classifiers using the first 80% of the data and use the trained models to predict the labels for the remaining 20% of the data (test set). I compare these model predictions to the ground truth values provided to obtain the overall accuracy of each classifier. It is worth noting that the training set consists of data points from each class but the test set consists solely of points with label = 0 (no divorce). I wasn't sure if this was the intended task, although language in the prompt suggested so, so as an additional layer of comparison, I also report the accuracies for when the data is shuffled before drawing the training and test sets. The results of these task are summarized in Table 1.

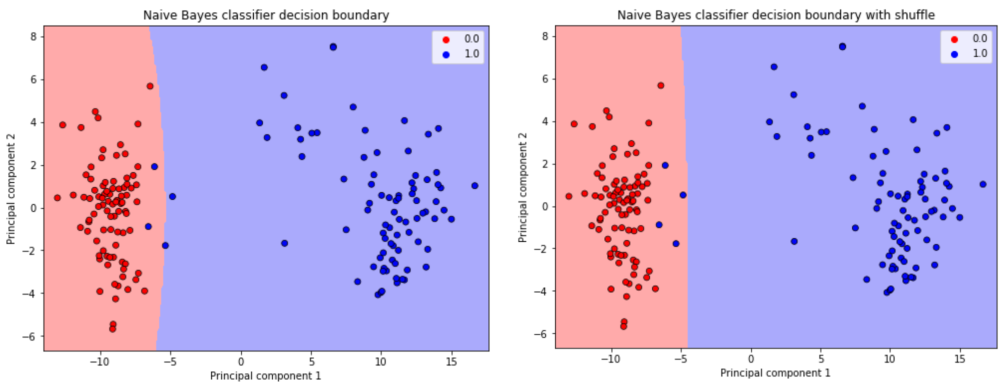
Classifier	Data shuffle before train/test split?	Accuracy
Naive Bayes	No	1.0
Naive Bayes	Yes	0.9412
Logistic Regression	No	0.8529
Logistic Regression	Yes	0.8824
KNN	No	1.0
KNN	Yes	0.9412

Table 1: Accuracy comparisons between the classifiers with and without shuffling.

It is clear from Table 1 that overall (with and without shuffling), the Naive Bayes classifier and KNN have the same performance based on this metric and are both "better" than the Logistic Regression classifier. I say "better" because accuracy is not the tell-all performance metric when comprehensively analyzing a classifier. I suspect and intuit that the reason Logistic Regression performs the worst is because the fully dimensional data is not linearly separable and the Logistic Regression decision boundary is linear whereas the decision boundary for Naive Bayes and KNN is soft.

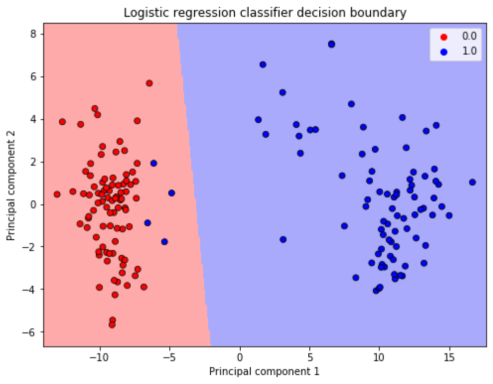
2.2 Decision boundaries

I now perform PCA on the dataset to project the data into 2D space. After doing so, it is much easier to visualize what the decision boundary for each classifier looks like in this lower dimensional space. I can use these boundaries to attempt to verify my performance-related hypothesis from the previous section. Figures 1-3 show these decision boundaries for each of the three classifiers being considered, with and without data shuffling. Additionally, I include the reduced data (full dataset) with ground truth labels. Recall that label = 0 means no divorce and label = 1 means divorced.

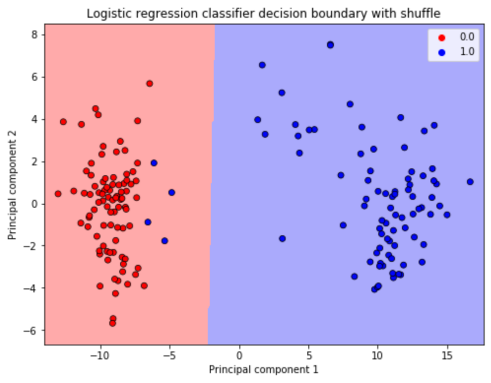


(a) shuffle = False (b) shuffle = True

Figure 1: Decision boundaries for Naive Bayes.

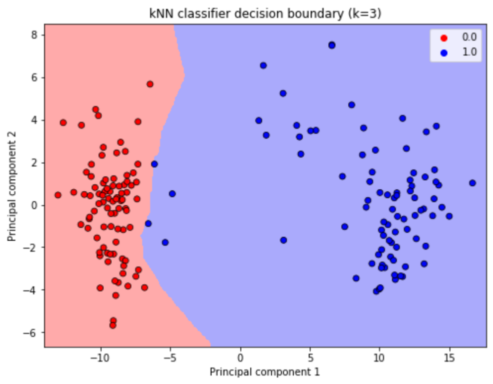


(a) shuffle = False

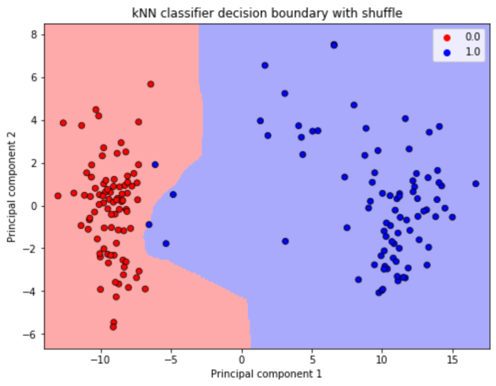


(b) shuffle = True

Figure 2: Decision boundaries for Logistic Regression.



(a) shuffle = False



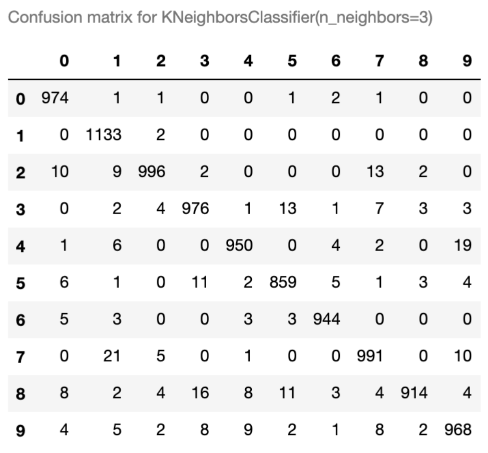
(b) shuffle = True

Figure 3: Decision boundaries for KNN.

It does in fact appear that the data is not linearly separable in 2D so my hypothesis from the previous section is likely true. The non-linearity of the Naive Bayes and KNN decision boundaries show that a slightly more "complex" or overfit model performs better on the test sets for this dataset both with and without shuffling. Based on the nature of this dataset, and the few points that lie close to one another but from different classes (in 2D), the KNN classifier might perform best for a dataset of this size but it is likely that the Naive Bayes (or even Logistic Regression) classifier could generalize better to additional unseen test data. It appears, in 2D at least, that both of these classifiers generally increase the margin even if they have miss-classification errors.

2.3 Handwritten digit classification & performance

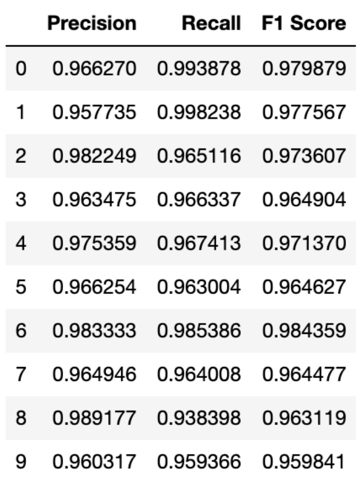
I once again turn to the MNIST handwritten digit dataset to train five different classifiers using 60,000 samples. I train KNN, Logistic Regression, SVM, kernel SVM, and Neural Network classifiers and seek to compare their performance using additional metrics and not solely accuracy. As a note, I standardize the features before training and for the KNN classifier, I tuned the parameter k via cross-validation. I first tested $k = 1...60$ taking every 5^{th} k and discovered that the accuracy monotonically decreased with increasing k. I then tested $k = 1...6$ and found $k = 3$ to be optimal (97.05% accuracy on test set). Additionally, as suggested I used hidden layer sizes (20, 10) for the neural network classifier and the radial basis function kernel for kernel SVM. After testing on the trained classifiers, I report the confusion matrices and the precision, recall, and F1 scores for each digit in each classifier for analysis. Figures 4-XXX summarize all of these results.



Confusion matrix for KNeighborsClassifier(n_neighbors=3)

	0	1	2	3	4	5	6	7	8	9
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	10	9	996	2	0	0	0	13	2	0
3	0	2	4	976	1	13	1	7	3	3
4	1	6	0	0	950	0	4	2	0	19
5	6	1	0	11	2	859	5	1	3	4
6	5	3	0	0	3	3	944	0	0	0
7	0	21	5	0	1	0	0	991	0	10
8	8	2	4	16	8	11	3	4	914	4
9	4	5	2	8	9	2	1	8	2	968

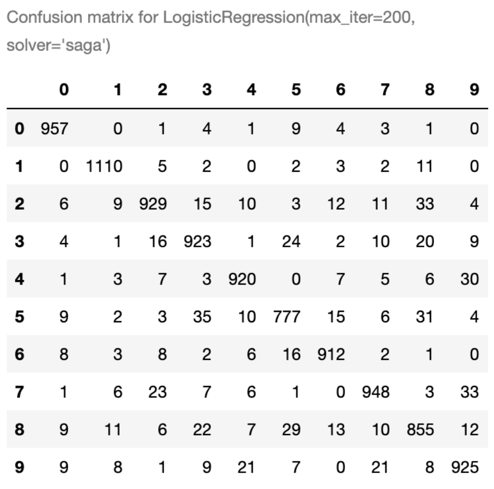
(a) Confusion matrix



	Precision	Recall	F1 Score
0	0.966270	0.993878	0.979879
1	0.957735	0.998238	0.977567
2	0.982249	0.965116	0.973607
3	0.963475	0.966337	0.964904
4	0.975359	0.967413	0.971370
5	0.966254	0.963004	0.964627
6	0.983333	0.985386	0.984359
7	0.964946	0.964008	0.964477
8	0.989177	0.938398	0.963119
9	0.960317	0.959366	0.959841

(b) Performance metrics

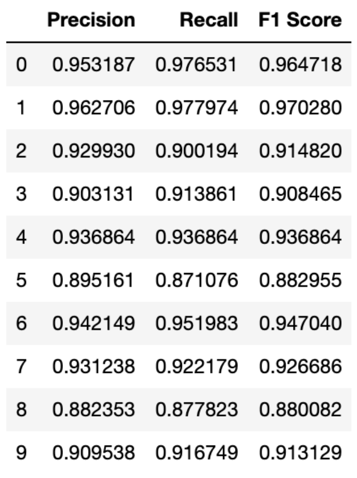
Figure 4: Per-digit performance for KNN.



Confusion matrix for LogisticRegression(max_iter=200, solver='saga')

	0	1	2	3	4	5	6	7	8	9
0	957	0	1	4	1	9	4	3	1	0
1	0	1110	5	2	0	2	3	2	11	0
2	6	9	929	15	10	3	12	11	33	4
3	4	1	16	923	1	24	2	10	20	9
4	1	3	7	3	920	0	7	5	6	30
5	9	2	3	35	10	777	15	6	31	4
6	8	3	8	2	6	16	912	2	1	0
7	1	6	23	7	6	1	0	948	3	33
8	9	11	6	22	7	29	13	10	855	12
9	9	8	1	9	21	7	0	21	8	925

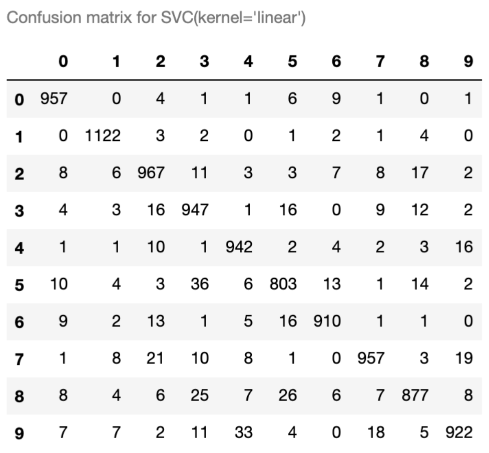
(a) Confusion matrix



	Precision	Recall	F1 Score
0	0.953187	0.976531	0.964718
1	0.962706	0.977974	0.970280
2	0.929930	0.900194	0.914820
3	0.903131	0.913861	0.908465
4	0.936864	0.936864	0.936864
5	0.895161	0.871076	0.882955
6	0.942149	0.951983	0.947040
7	0.931238	0.922179	0.926686
8	0.882353	0.877823	0.880082
9	0.909538	0.916749	0.913129

(b) Performance metrics

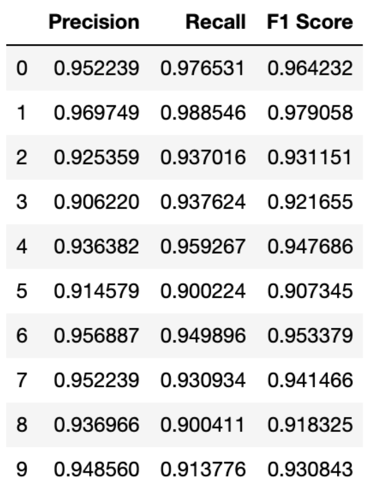
Figure 5: Per-digit performance for Logistic Regression.



Confusion matrix for SVC(kernel='linear')

	0	1	2	3	4	5	6	7	8	9
0	957	0	4	1	1	6	9	1	0	1
1	0	1122	3	2	0	1	2	1	4	0
2	8	6	967	11	3	3	7	8	17	2
3	4	3	16	947	1	16	0	9	12	2
4	1	1	10	1	942	2	4	2	3	16
5	10	4	3	36	6	803	13	1	14	2
6	9	2	13	1	5	16	910	1	1	0
7	1	8	21	10	8	1	0	957	3	19
8	8	4	6	25	7	26	6	7	877	8
9	7	7	2	11	33	4	0	18	5	922

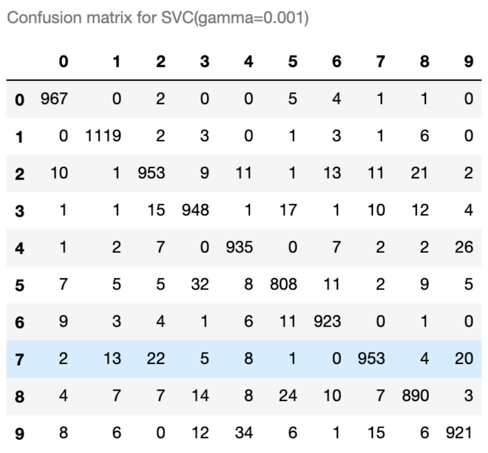
(a) Confusion matrix



	Precision	Recall	F1 Score
0	0.952239	0.976531	0.964232
1	0.969749	0.988546	0.979058
2	0.925359	0.937016	0.931151
3	0.906220	0.937624	0.921655
4	0.936382	0.959267	0.947686
5	0.914579	0.900224	0.907345
6	0.956887	0.949896	0.953379
7	0.952239	0.930934	0.941466
8	0.936966	0.900411	0.918325
9	0.948560	0.913776	0.930843

(b) Performance metrics

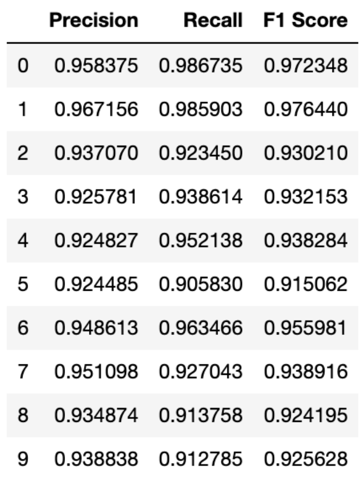
Figure 6: Per-digit performance for SVM.



Confusion matrix for SVC(gamma=0.001)

	0	1	2	3	4	5	6	7	8	9
0	967	0	2	0	0	5	4	1	1	0
1	0	1119	2	3	0	1	3	1	6	0
2	10	1	953	9	11	1	13	11	21	2
3	1	1	15	948	1	17	1	10	12	4
4	1	2	7	0	935	0	7	2	2	26
5	7	5	5	32	8	808	11	2	9	5
6	9	3	4	1	6	11	923	0	1	0
7	2	13	22	5	8	1	0	953	4	20
8	4	7	7	14	8	24	10	7	890	3
9	8	6	0	12	34	6	1	15	6	921

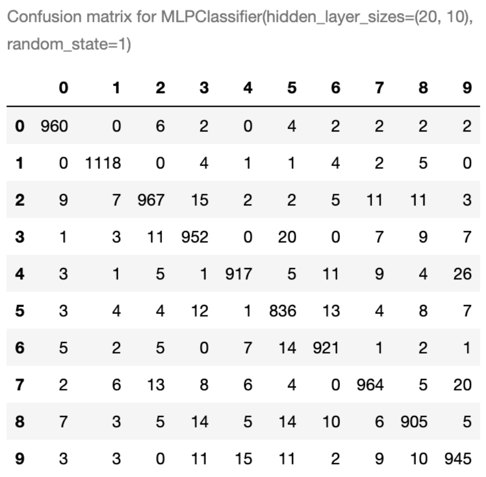
(a) Confusion matrix



	Precision	Recall	F1 Score
0	0.958375	0.986735	0.972348
1	0.967156	0.985903	0.976440
2	0.937070	0.923450	0.930210
3	0.925781	0.938614	0.932153
4	0.924827	0.952138	0.938284
5	0.924485	0.905830	0.915062
6	0.948613	0.963466	0.955981
7	0.951098	0.927043	0.938916
8	0.934874	0.913758	0.924195
9	0.938838	0.912785	0.925628

(b) Performance metrics

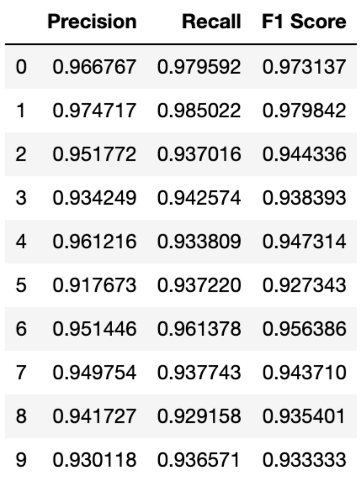
Figure 7: Per-digit performance for kernel SVM.



Confusion matrix for MLPClassifier(hidden_layer_sizes=(20, 10), random_state=1)

	0	1	2	3	4	5	6	7	8	9
0	960	0	6	2	0	4	2	2	2	2
1	0	1118	0	4	1	1	4	2	5	0
2	9	7	967	15	2	2	5	11	11	3
3	1	3	11	952	0	20	0	7	9	7
4	3	1	5	1	917	5	11	9	4	26
5	3	4	4	12	1	836	13	4	8	7
6	5	2	5	0	7	14	921	1	2	1
7	2	6	13	8	6	4	0	964	5	20
8	7	3	5	14	5	14	10	6	905	5
9	3	3	0	11	15	11	2	9	10	945

(a) Confusion matrix



	Precision	Recall	F1 Score
0	0.966767	0.979592	0.973137
1	0.974717	0.985022	0.979842
2	0.951772	0.937016	0.944336
3	0.934249	0.942574	0.938393
4	0.961216	0.933809	0.947314
5	0.917673	0.937220	0.927343
6	0.951446	0.961378	0.956386
7	0.949754	0.937743	0.943710
8	0.941727	0.929158	0.935401
9	0.930118	0.936571	0.933333

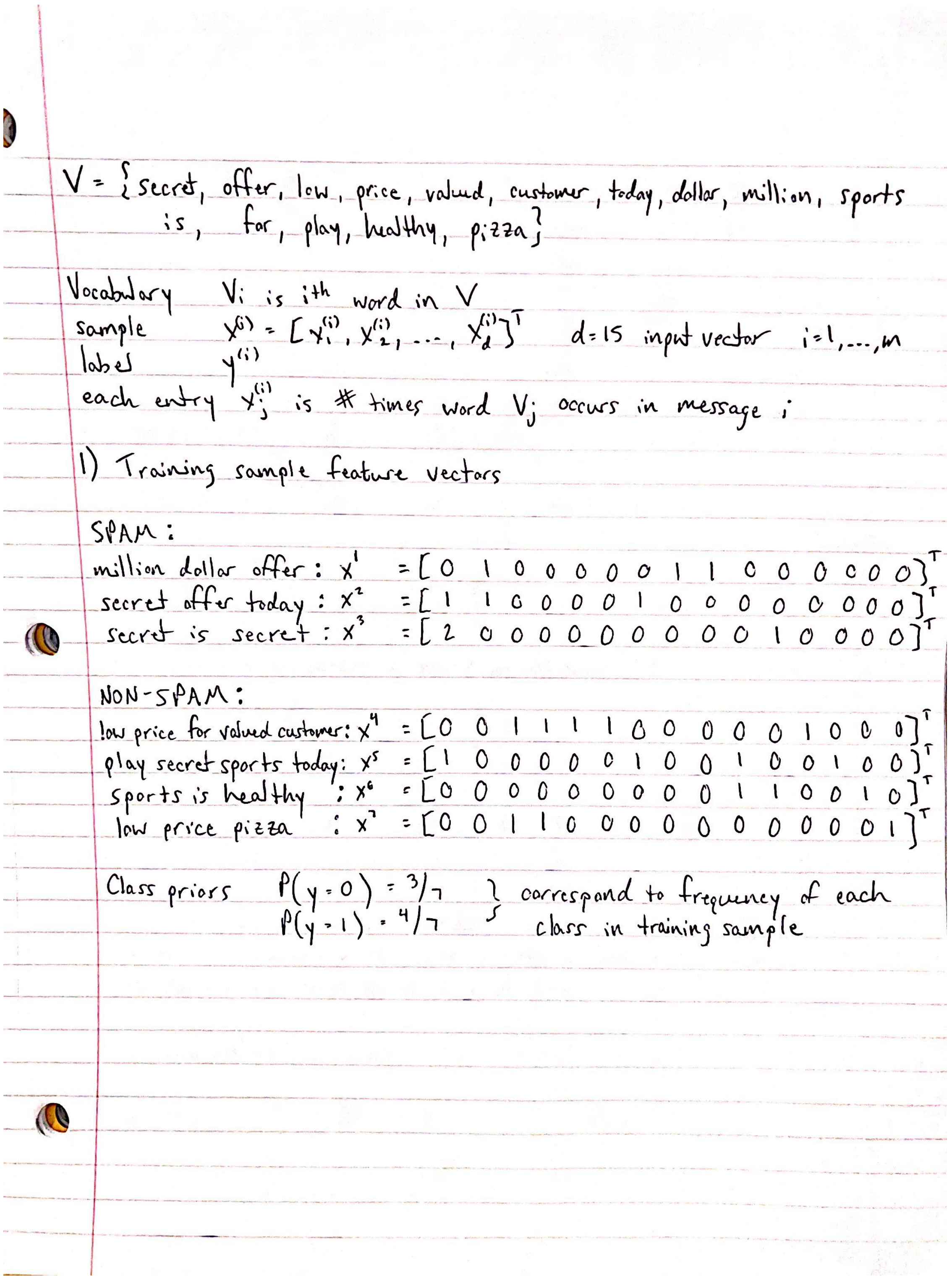
(b) Performance metrics

Figure 8: Per-digit performance for Neural Network.

Relatively speaking, all classifiers analyzed perform extremely well. The F1 score, the harmonic mean between the precision and recall, is likely the best all-encompassing metric for this analysis. The only classifier that deviates a bit more than others is Logistic Regression. Logistic Regression in the only classifier that has any F1 scores below 90% and it in fact has two, for the digits 5 and 8. First, it is worth noting that it is very plausible that 5 and 8 have low F1 scores because 5 arguably has the most amounts of types of segments to complete the digit and therefore might have high variance. For the 8, this digit if written slightly different could falsely be noted as a 0, 1 or even 2. Also, by comparing across classifiers and digits, its fairly consistent that the 5 and 8 digit have amongst the lowest F1 scores. I will again attribute the worse metrics of the Logistic Regression classifier to the fact that it employs a linear separator. In a similar light, that is also why linear SVM joins Logistic Regression as one of the lower-performers. SVM appears to actually have the best overall F1 scores and this makes sense for a dataset with a large number of data points and a small k because it is very likely that a point close to the test point in feature space is the correct label considering the conditional distributions of each digit.

3 Naive Bayes for spam filtering

All math derivations and explanations are hand-written for this section below.



$V = \{$secret, offer, low, price, valued, customer, today, dollar, million, sports is, for, play, healthy, pizza$\}$

Vocabulary V_i is ith word in V

sample $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^T$ $d = 15$ input vector $i = 1, \ldots, m$

label $y^{(i)}$

each entry $x_j^{(i)}$ is # times word V_j occurs in message i

1) Training sample feature vectors

SPAM:

million dollar offer : $x^1 = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$

secret offer today : $x^2 = [1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$

secret is secret : $x^3 = [2\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$

NON-SPAM:

low price for valued customer: $x^4 = [0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$

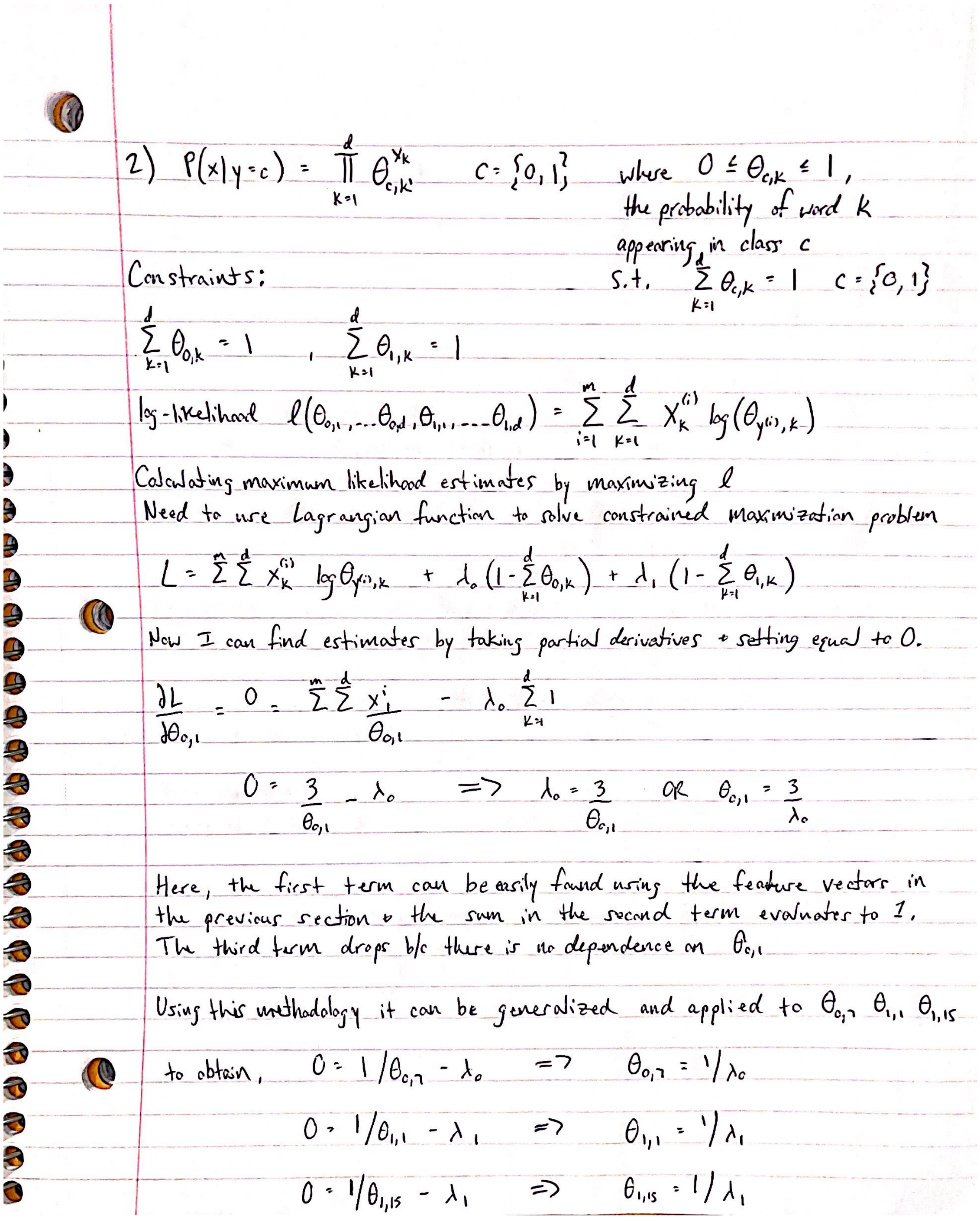
play secret sports today: $x^5 = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0]^T$

sports is healthy : $x^6 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0]^T$

low price pizza : $x^7 = [0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$

Class priors $P(y=0) = 3/7$, $P(y=1) = 4/7$ $\}$ correspond to frequency of each class in training sample

Figure 9



2) $P(x|y=c) = \prod_{k=1}^{d} \theta_{c,k}^{x_k}$ $\quad c=\{0,1\}$ where $0 \le \theta_{c,k} \le 1$, the probability of word k appearing in class c s.t. $\sum_{k=1}^{d} \theta_{c,k} = 1 \quad c=\{0,1\}$

Constraints:

$$\sum_{k=1}^{d} \theta_{0,k} = 1 \quad , \quad \sum_{k=1}^{d} \theta_{1,k} = 1$$

log-likelihood $\ell(\theta_{0,1}, \ldots \theta_{0,d}, \theta_{1,1}, \ldots \theta_{1,d}) = \sum_{i=1}^{m} \sum_{k=1}^{d} X_k^{(i)} \log(\theta_{y^{(i)},k})$

Calculating maximum likelihood estimates by maximizing ℓ

Need to use Lagrangian function to solve constrained maximization problem

$$L = \sum^{m} \sum^{d} X_k^{(i)} \log \theta_{y^{(i)},k} + \lambda_0 \left(1 - \sum_{k=1}^{d} \theta_{0,k}\right) + \lambda_1 \left(1 - \sum_{k=1}^{d} \theta_{1,k}\right)$$

Now I can find estimates by taking partial derivatives + setting equal to 0.

$$\frac{\partial L}{\partial \theta_{0,1}} = 0 = \sum^{m} \sum^{d} \frac{x_1^i}{\theta_{0,1}} - \lambda_0 \sum_{k=1}^{d} 1$$

$$0 = \frac{3}{\theta_{0,1}} - \lambda_0 \implies \lambda_0 = \frac{3}{\theta_{0,1}} \quad \text{OR} \quad \theta_{0,1} = \frac{3}{\lambda_0}$$

Here, the first term can be easily found using the feature vectors in the previous section + the sum in the second term evaluates to 1. The third term drops b/c there is no dependence on $\theta_{0,1}$

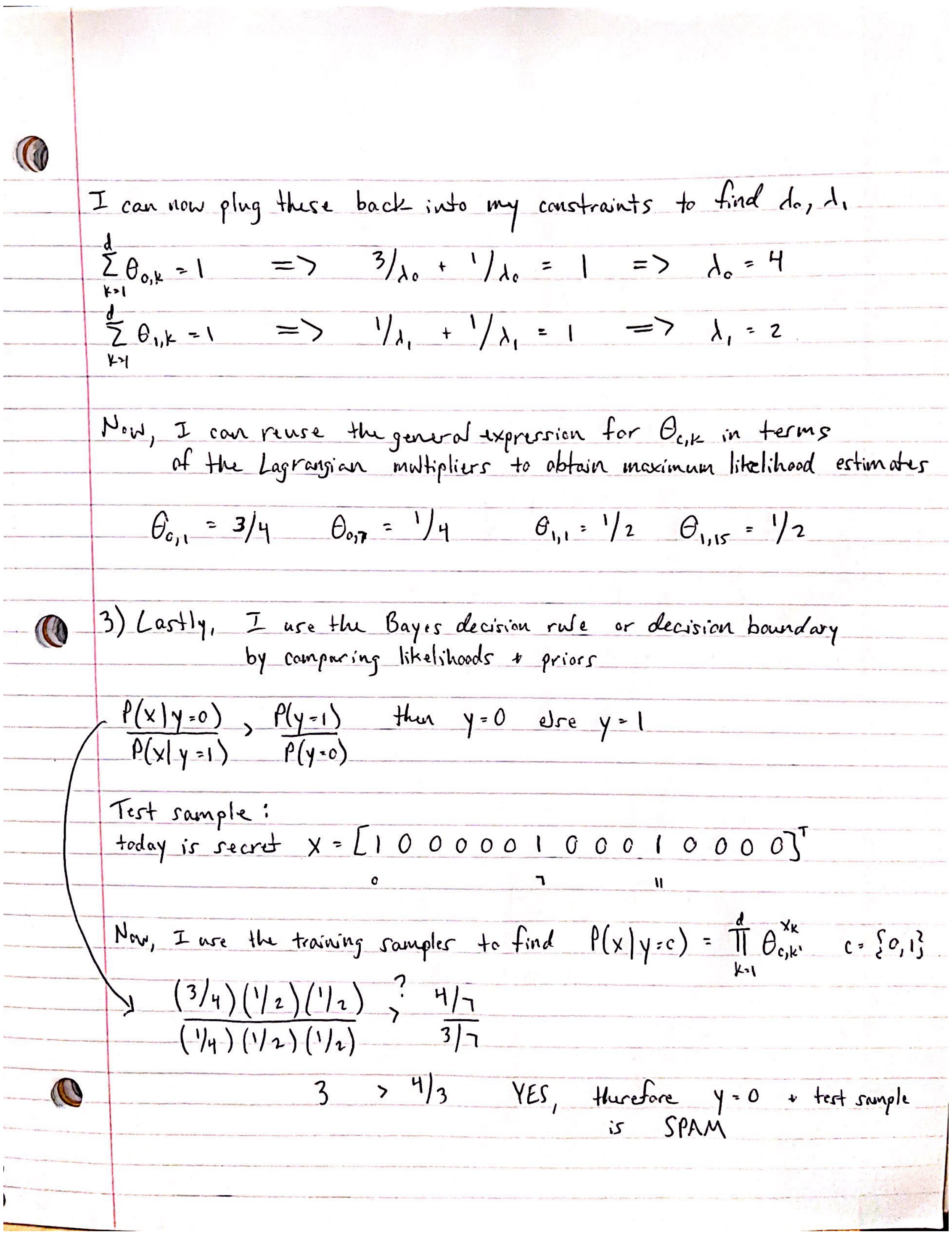
Using this methodology it can be generalized and applied to $\theta_{0,7}$ $\theta_{1,1}$ $\theta_{1,15}$

to obtain, $0 = 1/\theta_{0,7} - \lambda_0 \implies \theta_{0,7} = 1/\lambda_0$

$$0 = 1/\theta_{1,1} - \lambda_1 \implies \theta_{1,1} = 1/\lambda_1$$

$$0 = 1/\theta_{1,15} - \lambda_1 \implies \theta_{1,15} = 1/\lambda_1$$

Figure 10



I can now plug these back into my constraints to find λ_0, λ_1

$$\sum_{k=1}^{d} \theta_{0,k} = 1 \quad \Rightarrow \quad 3/\lambda_0 + 1/\lambda_0 = 1 \quad \Rightarrow \quad \lambda_0 = 4$$

$$\sum_{k=1}^{d} \theta_{1,k} = 1 \quad \Rightarrow \quad 1/\lambda_1 + 1/\lambda_1 = 1 \quad \Rightarrow \quad \lambda_1 = 2$$

Now, I can reuse the general expression for $\theta_{c,k}$ in terms of the Lagrangian multipliers to obtain maximum likelihood estimates

$$\theta_{0,1} = 3/4 \qquad \theta_{0,7} = 1/4 \qquad \theta_{1,1} = 1/2 \qquad \theta_{1,15} = 1/2$$

3) Lastly, I use the Bayes decision rule or decision boundary by comparing likelihoods + priors

$$\frac{P(x|y=0)}{P(x|y=1)} > \frac{P(y=1)}{P(y=0)} \quad \text{then } y=0 \quad \text{else } y=1$$

Test sample:

today is secret $x = [1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$

0 7 11

Now, I use the training samples to find $P(x|y=c) = \prod_{k=1}^{d} \theta_{c,k}^{x_k}$ $\quad c = \{0,1\}$

$$\frac{(3/4)(1/2)(1/2)}{(1/4)(1/2)(1/2)} \overset{?}{>} \frac{4/7}{3/7}$$

$3 > 4/3$ YES, therefore $y = 0$ + test sample is SPAM

Figure 11

References

[1] https://towardsdatascience.com/easily-visualize-scikit-learn-models-decision-boundaries-dd0fb3747508

[2] https://stackoverflow.com/questions/45075638/graph-k-nn-decision-boundaries-in-matplotlib