UNIVERSITY OF MICHIGAN
Department of Electrical Engineering and Computer Science
EECS 445 — Introduction to Machine Learning
Fall 2022 Project 1 - Sachchit's Social SVMs

Due: Wednesday, 10/5 at 10:00pm

Section	Points	Recommended Completion Date
2. Dataset Considerations	11	Friday, 9/23
3. Feature Extraction	12	Friday, 9/23
4. Hyperparameter and Model Selection	35	Wednesday, 9/28
5. Asymmetric Cost Functions and Class Imbalance	20	Friday, 9/30
6. Challenge	14	Tuesday, 10/4
Code Appendix	8	Tuesday, 10/4

This spec contains 17 pages, including Appendices with approximate run-times for programming problems, as well as a list of topics, concepts, and further reading.

Include your entire project code (copy/paste or screenshot) as an appendix in your report submission, which should be submitted to assignment named Project 1 on Gradescope. Please try to format lines of code so they are visible within the pages.

Upload your file `uniqname.csv` containing the label predictions for the held-out data to the Canvas assignment named Project 1 Challenge Submissions.

1 Introduction

Sachchit has spent a lot of time on online boards and forums lately, and he wants to be able to have better conversations. Specifically, he wants to know the sentiment that an online forum has about a certain topic, so that he can respond appropriately. Thankfully, Sachchit has a group of EECS 445 students willing to help him, who have become well-versed in solving complex supervised Machine Learning problems. He plans to solve the task of quickly identifying the mood of an online forum by training a model to deduce the sentiment of the text speech (i.e determine what emotion the speaker felt based on their diction).

Since the historical log of EECS 445 Piazza posts does not result in a large enough dataset, Sachchit will test the feasibility of this classification task on Reddit data. The Reddit comments contain thousands of different phrases across many users, and the main two emotions you will identify are **gratitude** (here used to describe any **positive** emotion) and **sadness** (used for **negative** emotion). You will work with this dataset to train various Support Vector Machines (SVMs) to classify the emotion of a comment. In this process, you will also explore some very useful `scikit-learn` modules and data science techniques.

1.1 Requirements:

1. Python (https://www.python.org/downloads/), with a Python 3.10 virtual environment.

2. `scikit-learn` (1.1.2): https://scikit-learn.org/stable/index.html
3. `numpy` (1.23.0): http://www.numpy.org/
4. `pandas` (1.4.4): https://pandas.pydata.org/
5. `matplotlib` (3.5.2): https://matplotlib.org/

See the section below for more information on installing these packages onto your system.

1.2 Getting Started

To get started, download `Project1` from Canvas. It should contain the following files:

- `data/dataset.csv`
- `data/heldout.csv`
- `data/debug.csv`
- `debug_output.txt`
- `project1.py`
- `helper.py`
- `test_output.py`
- `requirements.txt`

The csv files in `data/` have Reddit comments. If you open `dataset.csv`, you will find that it has 4 columns: *text*, *created_utc*, *label*, and *emotion*. Each row in the file corresponds to one comment. The *text* column contains the text of the comment. The *label* column is a multiclass label: 1 if the emotion of the comment was **gratitude**, 0 if **neutral**, and -1 if the emotion of the comment was **sadness**. The *created_utc* column contains the UTC timestamp when the comment was posted.

You will use the *text* and *label* columns for most of the project (we will ignore the 0 label comments in order to make the label binary). The final challenge portion, however, will utilize all -1, 0, and 1 labeled comments.

The helper file `helper.py` provides functions that allow you to read in the data from `csv` files. The file `project1.py` contains skeleton code for the project. The file `test_output.py` allows you to test your output csv file before submission to make sure the format is correct.

You can run your code quickly on `data/debug.csv` and compare against `debug_output.txt` to make sure your code is implemented correctly before running it on `data/dataset.csv`. **Please note that this is not an exhaustive test case suite.** Due to variance in numerical precision across processors, and some inherent randomness, some of your results may be off. As a rule of thumb, if your results are within ± 0.01 for debug, you should feel confident running your code on the actual dataset. **Further, don't use the debug output to answer any analytical questions in the project.** Because of its small

size, the model being learned will not be useful and many interesting trends won't be present. The format of your output doesn't need to match the provided file exactly, but the numerical values should be reasonably close. To most closely match the output results, make sure 1) your package versions match the `requirements.txt` and 2) use the `random_state=445` keyword argument when instantiating your SVMs. You can help to ensure that your package versions match those specified with the command `pip install -r requirements.txt` while your virtual environment is activated.

The data for each part of the project has already been read in for you in the main function of the skeleton code. Please do not change how the data is read in; doing so may affect your results.

The skeleton code `project1.py` provides specifications for functions that you will implement. You may choose to implement additional helper functions as you see fit.

- `extract_word`
- `extract_dictionary`
- `generate_feature_matrix`
- `cv_performance`
- `select_param_linear`
- `plot_weight`
- `select_param_quadratic`
- Optional: `performance`

1.3 Submitting Your Work

This project contains questions that involve coding as well as others where you'll be asked to write up an answer and submit to Gradescope. **Coding questions will be highlighted green, and questions that require written answers will be highlighted blue.** Please make sure to complete both and attach your code to your report. Additionally, remember to submit your challenge predictions to Canvas (see section 6 for more details).

2 Dataset Considerations [11 pts]

Machine learning methods can be powerful tools that drive decision making. However, they need to be used carefully as they have the potential to reinforce biases and cause harm. It is therefore imperative that we take care while constructing or choosing datasets for prediction tasks, as the outcome of a machine learning algorithm is entirely dependent on the dataset used to train it. Recent work proposes a series of questions to critically evaluate datasets.

(a) (9 pts) Answer the following questions about the dataset we provided. If you make any assumptions about the source of the data, please state them in your answer.

i) Is it possible to identify individuals (i.e., one or more natural persons), either directly or indirectly (i.e., in combination with other data) from the dataset?

ii) What tasks could the dataset be used for?

iii) Are there tasks for which the dataset should not be used?

(b) (2 pts) Supervised learning, as we have seen, requires a labeled training dataset. One way to obtain labels is through crowdsourcing; effectively having human beings provide labels for each datapoint. This could lead to errors in the training data itself; in fact, recent work has demonstrated that many commonly used datasets suffer from this issue.

i) What is one potential source of inaccuracy in the labels in these datasets?

ii) How could inaccurate labels harm our ability to construct an effective model?

3 Feature Extraction [12 pts]

Given a dictionary containing d unique words, we can transform a number of n comments into n feature vectors of length d, by setting the i^{th} element of the j^{th} feature vector to 1 if the i^{th} word is in the j^{th} comment, and 0 otherwise. If the four words {`'book':0, 'was':1, 'the':2, 'best':3`} are the only words we encountered in the training data, the comment *"BEST book ever!!"* would map to the feature vector $[1, 0, 0, 1]$.

Note that we do not consider case ("Best" and "beSt" is the same). Also, note that since the word "ever" was not in the original dictionary, it is ignored as a feature. Because we use the training data to construct our dictionary for testing data set, there may be words in test data that you do not encounter in training data. There are many interesting methods for dealing with this that you may explore in part 6.

(a) (2 pts) Start by implementing the `extract_word(input_string)` function. This function should return an array of words, in lowercase, that were separated by white space or punctuation in the input string.

Include in your write-up the result of `extract_word(input_string)` on the sentence:

```
'It's a test sentence! Does it look CORRECT?'
```

Hint: You should be implementing this function with `string.punctuation` and `string.replace()`. No external library is needed.

(b) (4 pts) Now, implement the `extract_dictionary(df)` function. You will use this function to read all unique words contained in the training data by running `get_split_binary_data()` on `dataset.csv` after `extract_dictionary(df)` is correctly implemented. You will thus construct the dictionary described at the beginning of this section. Make use of the `extract_word(input_string)` function you just implemented! Your function should return a dictionary of length d, the number of unique words.

Include in your write-up the value of d.

(c) (6 pt) Next, implement the `generate_feature_matrix(df, word_dict)` function. Assuming that there are n comments total, return the feature vectors as an (n, d) feature matrix, where each row represents a comment, and each column represents whether or not a specific word appeared in that comment. In your write-up, include the following:

- The average number of non-zero *features* per comment in the training data. You will need to calculate this on `X_train`.
- The word appearing in the greatest number of comments. You will need to make use of the dictionary returned by `get_split_binary_data`.

4 Hyperparameter and Model Selection [35 pts]

In section 3, you implemented functions that can transform the comments into a feature matrix `X_train` and a label vector `y_train`. Test data `X_test, y_test` has also been read in for you. **You will use this data for all of question 4.**

We will learn a classifier to separate the *training* data into positive and negative labels. The labels in `y_train` are transformed into binary labels in $\{-1, 1\}$, where -1 means "sad" and 1 means "gratitude."

For the classifier, we will explore SVMs with two different kernels (linear and quadratic), penalties (L1 and L2), and loss functions (hinge and squared hinge). In sections 4.1 and 4.2, we will make use of the `sklearn.svm.LinearSVC` class. In 4.3 we will make use the `sklearn.svm.SVC` class.

In addition, we will use the following methods in both the classes: `fit(X,y)`, `predict(X)` and `decision_function(X)` in the LinearSVC class– please use `predict(X)` when measuring for any performance metric that is not AUROC and `decision_function(X)` for AUROC. `predict(X)` produces labels [-1,1], whereas `decision_function(X)` produce confidence scores (distance to hyperplane in SVM case).

As discussed in lecture, SVMs have hyperparameters, which must be set by the user. For both linear-kernel and quadratic-kernel SVMs, we will select hyperparameters using 5-fold cross-validation (CV) on the training data. We will select the hyperparameters that lead to the 'best' mean performance across all five folds. The result of hyperparameter selection often depends upon the choice of performance measure. Here, we will consider the following performance measures: **Accuracy**, **F1-Score**, **AUROC**, **Precision**, **Sensitivity**, and **Specificity**.

Note: When calculating some metrics, it is possible that a divide-by-zero may occur which throws a warning. Consider how these metrics are calculated, perhaps by reviewing the relevant `scikit-learn` documentation.

Some of these measures are already implemented as functions in the `sklearn.metrics` submodule. Please use `sklearn.metrics.roc_auc_score` for AUROC. You can use the values from `sklearn.metrics.confusion_matrix` to calculate the others, or methods from the submodule where applicable (**Note:** the confusion matrix is just the table of predicted vs. actual label counts. That is, the true positive, false positive, true negative, and false negative counts for binary classification). Make sure to read the documentation carefully, as when calling this function you will want to set `labels=[1,-1]` for a deterministic ordering of your confusion matrix output.

4.1 Hyperparameter Selection for a Linear-Kernel SVM [18 pts]

(a) (2 pts) You may have noticed that the proportion of the two classes (positive and non-positive) are equal in the training data. When dividing the data into folds for CV, you should try to keep the class proportions roughly the same across folds. In our case, the class proportions should be roughly equal across folds since the original training data has equal class proportions, so you will not have to worry about this. In your write-up, briefly describe why it might be beneficial to maintain class proportions across folds.

(b) (8 pts) Next, you will need to implement some functions.

- Implement the function `cv_performance` as defined in the skeleton code. The function returns the mean k-fold CV performance for the performance metric passed into the function. The default metric is `"accuracy"`, but your function should work for all of the metrics listed above. It may be useful to have a helper function that calculates each performance metric. For instance: `performance(y_true, y_pred, metric='accuracy')`. You will make use of the `fit(X,y)`, `predict(X)`, and `decision_function(X)` methods in the LinearSVC class. You must implement this function without using the `scikit_learn` implementation of CV. You will need to employ the `sklearn.model_selection.StratifiedKFold()` class for splitting the data. Do not shuffle points (i.e., do not set `shuffle=True`). This is so the generated folds are consistent for the same dataset across runs of the entire program.
- Now implement the `select_param_linear` function to choose a value of C for a linear SVM based on the training data and the specified metric. Below is the scikit-learn formulation of SVM:

$$\begin{aligned} &\underset{\bar{\theta},b,\xi_i}{\text{minimize}} \ \frac{||\bar{\theta}||^2}{2} + C\sum_{i=1}^{n}\xi_i \\ &\text{subject to } y^{(i)}(\bar{\theta}\cdot\bar{x}^{(i)} + b) \geq 1 - \xi_i \\ &\xi_i \geq 0, \forall i = 1, 2, ..., n \end{aligned}$$

  Your function should call your cross-validation function (implemented earlier), passing in instances of `LinearSVC()` with a range of values for C chosen in powers of 10 between 10^{-3} and 10^3 (i.e. $10^{-3}, 10^{-2}, \ldots, 10^3$) and the appropriate parameters from `select_param_linear()`. **Remember** to set the `random_state` parameter of `LinearSVC()` as 445.

After you have implemented the two functions, use the training data from question 2 and the functions implemented here to find the best setting for C for each performance measure (if there is a tie, choose the smaller C value). Report your findings in tabular format with three columns: names of the performance measures, along with the corresponding values of C and the mean cross-validation performance. The table should follow the format given below:

Performance Measures	C	CV Performance
Accuracy		
F1-Score		
AUROC		
Precision		
Sensitivity		
Specificity		

Your `select_param_linear` function returns the 'best' value of C given a range of values. Note: as we are working with a fairly large feature matrix, this may take a few minutes.

Also, in your write-up, describe how the 5-fold CV performance varies with C. If you have to train a final model, which performance measure would you optimize for when choosing C? Explain your choice. This value of C will be used in part c.

(c) (3 pt) Now, using the value of C that maximizes your chosen performance measure in (b), create an SVM as in the previous question. Train this SVM on the training data `X_train, y_train`. Report the performance of this SVM on the test data `X_test, y_test` for each metric below.

Performance Measures	Performance
Accuracy	
F1-Score	
AUROC	
Precision	
Sensitvity	
Specificity	

(d) (2 pt) Finish the implementation of the `plot_weight(X, y, penalty, C_range, dual)` function. In this function, you need to find the L0-norm of $\bar{\theta}$, the parameter vector learned by the SVM, for each value of C in the given range. Finding out how to get the vector $\bar{\theta}$ from a `LinearSVC` object may require you to dig into the documentation. The L0-norm is given as follows, for $\bar{\theta} \in \mathbb{R}^d$:

$$\|\bar{\theta}\|_0 = \sum_{i=1}^{d} \mathbb{I}\{\theta_i \neq 0\}$$

where $\mathbb{I}\{\theta_i \neq 0\}$ is 0 if θ_i is 0 and 1 otherwise.

Use the complete training data `X_train, Y_train`, i.e, don't use cross-validation for this part. Once you implement the function, the existing code will plot L0-norm $\|\bar{\theta}\|_0$ against C and save it to a file.

In your write-up, include the produced plot from the question above.

(e) (2 pt) Recall that each element of $\bar{\theta}$ is associated with a word. The more positive the value of the element, the more the presence of the associated word indicates that the comment is positive, and similarly with negative coefficients. In this way, we can use these coefficients to find out what word-rating associations our SVM has learned.

Using $C = 0.1$, train an SVM on `X_train, Y_train`. In your report, include a bar chart (coefficient vs each word) for both the five most positive and five most negative coefficients from the trained SVM. All the words on the bar chart should be sorted by coefficient value in ascending order.

(f) (1 pt) It is noteworthy that the word-rating association learned can be misleading. To illustrate this, formulate a comment that is conveying sadness yet contains three of the five words with the most positive coefficients (from your answer to (e)).

4.2 Linear-Kernel SVM with L1 Penalty and Squared Hinge Loss [4 pts]

In this part of the project, you will explore the use of a different penalty (i.e., regularization term) and a different loss function. In particular, we will use the L1 penalty and squared hinge loss which corresponds to the following optimization problem.

$$\underset{\bar{\theta},b}{\text{minimize}} \; ||\bar{\theta}||_1 + C \sum_{i=1}^{n} \text{loss}(y^{(i)}(\bar{\theta} \cdot \bar{x}^{(i)} + b))$$

where $\text{loss}(z) = \max\{0, (1-z)\}^2$. We will consider only a linear-kernel SVM and the original (untransformed) features. When calling `LinearSVC` please use the following settings:

```
LinearSVC(penalty='l1', loss = 'squared_hinge', C=c, dual=False).
```

(a) (1 pt) Using the training data from question 2 and 5-fold CV, find the best setting for C that maximizes the mean AUROC given that $C \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ In the case of ties, report the lower C value. Then using the C value you found, train a SVM with squared hinge loss on the entirety of the training data and report its AUROC on the test set. Report the C value, the mean CV AUROC score, and the AUROC score on the test set.

(b) (1 pt) Similar to 4.1(d), plot the L0-norm of the learned parameter $\bar{\theta}$ against C using complete training data and no cross-validation. You should be able to re-use the function `plot_weight` with different input parameters without writing additional code. Include the plot in your write-up.

(c) (1 pt) Compare and contrast the graphs that you generated for both the L1 and L2 penalty. Why does this difference occur? (**Hint:** Think about the gradients)

(d) (1 pt) Note that using the Squared Hinge Loss (as opposed to the Hinge Loss) changes the objective function as shown above. What effect do you expect this will have on the optimal solution?

4.3 Hyperparameter Selection for a Quadratic-Kernel SVM [9 pts]

Similar to the hyperparameter selection for a linear-kernel SVM, you will perform hyperparameter selection for a quadratic-kernel SVM. Here we are assuming a kernel of the form $(\bar{x} \cdot \bar{x}' + r)^2$, where r is a hyperparameter. In this part, we will make use of the `sklearn.svm.SVC` class.

(a) (3 pt) Implement the `select_param_quadratic` function to choose a setting for C and r as in the previous part. Your function should call your CV function (implemented earlier) passing in instances of `SVC(kernel='poly', degree=2, C=c, coef0=r, gamma='auto')`.

The function argument `param_range` should be a matrix with two columns with first column as values of C and second column as values of r, where each row represents a pair of parameters. You will need to try out the range of parameters via two methods:

i) Grid Search: In this case, we look at all the specified values of C and r in a given set and choose the best setting after tuning. For this question, the values should be considered in powers of 10 for both C (between 10^{-2} and 10^{3}) and r (between 10^{-2} and 10^{3}) [A total of 36 pairs]. This code will take a substantial time to run (expect to wait for up to 90 minutes for full results).

ii) Random Search: Instead of assigning fixed values for the parameter C and r, we can also sample random values from a particular distribution. For this case, we will be sampling from a log-uniform distribution, i.e., the log of random variables follows a uniform distribution:

$$P[a \leq \log C \leq b] = k(b-a)$$

for some constant k so the distribution is valid. In other words, we sample a uniform distribution with the same range as above to yield exponents x_i, and corresponding values of $C = 10^{x_i}$.

In your case, the values should range from the powers of 10 which you used in part (i). Choose 25 pairs of such sampled pairs of (C, r). Again, this code may take time to run (expect to wait up to an hour for full results).

Find the best C and r value for AUROC using both tuning schemes mentioned above and the training data from question 3. Report your findings in tabular format. The table should have four columns: Tuning Scheme, C, r and AUROC. Again, in the case of ties, report the lower C and the lower r values that perform the best (prioritizing a lower C) . Your table should look be similar to the following:

Tuning Scheme	**C**	**r**	**AUROC**
Grid Search			
Random Search			

(b) (6 pt) How does the 5-fold CV performance vary with C and r for each tuning scheme, grid search and random search? Also, explain the pros and cons of grid search and random search.

4.4 Learning Non-linear Classifiers with a Linear-Kernel SVM [4 pts]

Here, we will explore the use of an explicit feature mapping in place of a kernel. (Note: you do not need to write any code for this question)

(a) (2 pt) Describe a feature mapping, $\phi(\bar{x})$, that maps your data to the same feature space as the one implied by the quadratic kernel from the question above (please use $x_1, x_2, ..., x_n$ and constant r to write your answer).

(b) (2 pt) Instead of using a quadratic-kernel SVM, we could simply map the data to this higher dimensional space via this mapping and then learn a linear classifier in this higher-dimensional space. What are the tradeoffs (pros and cons) of using an explicit feature mapping over a kernel? Explain.

5 Asymmetric Cost Functions and Class Imbalance [20 pts]

The training data we have provided you with so far is *balanced*: the data contain an equal number of positive and negative ratings. But this is not always the case. In many situations, you are given imbalanced data, where one class may be represented more than the others.

In this section, you will investigate the objective function of the SVM, and how we can modify it to fit situations with class imbalance. Recall that the objective function for an SVM in scikit-learn is as follows:

$$\begin{aligned} \underset{\bar{\theta},b,\xi_i}{\text{minimize}} \ & \frac{||\bar{\theta}||^2}{2} + C\sum_{i=1}^{n}\xi_i \\ \text{subject to } & y^{(i)}\left(\bar{\theta}\cdot\phi(\bar{x}^{(i)}) + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, \forall i = 1, 2, 3, ..., n \end{aligned}$$

We can modify it in the following way:

$$\begin{aligned} \underset{\bar{\theta},b,\xi_i}{\text{minimize}} \ & \frac{||\bar{\theta}||^2}{2} + W_p * C\sum_{i|y^{(i)}=1}\xi_i + W_n * C\sum_{i|y^{(i)}=-1}\xi_i \\ \text{subject to } & y^{(i)}\left(\bar{\theta}\cdot\phi(\bar{x}^{(i)}) + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, \forall i = 1, 2, 3, ..., n \end{aligned}$$

where $\sum_{i|y^{(i)}=1}$ is a sum over all indices i where the corresponding point is positive $y^{(i)} = 1$. Similarly, $\sum_{i|y^{(i)}=-1}$ is a sum over all indices i where the corresponding point is negative $y^{(i)} = -1$.

5.1 Arbitrary class weights [6 pts]

W_p and W_n are called "class weights" and are built-in parameters in scikit-learn.

(a) (1 pt) Describe how this modification will change the solution. If W_n is much greater than W_p, what does this mean in terms of classifying positive and negative points? Refer to the weighted SVM formulation for a brief justification of your reasoning.

(b) (1 pt) What is the difference between setting $W_n = 0.25, W_p = 1$ and $W_n = 1, W_p = 4$? How does this difference affect the objective function? Refer to the weighted SVM formulation to justify your answer.

(c) (3 pts) Create a linear-kernel SVM with hinge loss and L2-penalty with $C = 0.01$. This time, when calling `LinearSVC`, set `class_weight = {-1: 1, 1: 10}`. This corresponds to setting $W_n = 1$ and $W_p = 10$. Train this SVM on the training data `X_train, y_train`. Report the performance of the modified SVM on the test data `X_test, y_test` for each metric below.

Performance Measures	Performance
Accuracy	
F1-Score	
AUROC	
Precision	
Sensitivity	
Specificity	

(d) (1 pt) Compared to your work in question 4.1(c), which performance measures were affected the most by the new class weights? Why do you suspect this is the case?

Note: We set $C = 0.01$ to ensure that interesting trends can be found regardless of your work in question 4. This may mean that your value for C differs in 5 and 4.1(c). **This does not mean your answer for 4.1(c) is wrong if you did not use** $C = 0.01$**.** In a real machine learning setting, you'd have to be more careful about how you compare models.

5.2 Imbalanced data [5 pts]

You just saw the effect of arbitrarily setting the class weights when our training set is already balanced. Let's return to the class weights you are used to: $W_n = W_p$. We turn our attention to class imbalance. Using the functions you wrote in part 3, we have provided you with a second feature matrix and vector of binary labels `IMB_features, IMB_labels`. This class-imbalanced data set has roughly 4 times as many positive data points as negative data points. It also comes with a corresponding test feature matrix and label vector pair `IMB_test_features, IMB_test_labels`, which have the same class imbalances.

(a) (3 pts) Create a linear-kernel SVM with hinge loss, L2-penalty and as before, $C = 0.01$. Return the SVM to the formulation you have seen in class by setting `class_weight={-1: 1, 1: 1}`. Now train this SVM on the class-imbalanced data `IMB_features, IMB_labels` provided. Use this classifier to predict the provided test data `IMB_test_features, IMB_test_labels` and report the accuracy, specificity, sensitivity, precision, AUROC, and F1-Score of your predictions:

Class Weights	Performance Measures	Performance
$W_n = 1, W_p = 1$	Accuracy	
$W_n = 1, W_p = 1$	F1-Score	
$W_n = 1, W_p = 1$	Auroc	
$W_n = 1, W_p = 1$	Precision	
$W_n = 1, W_p = 1$	Sensitivity	
$W_n = 1, W_p = 1$	Specificity	

(b) (1 pt) Compared to your work in 4.1(c), Which performance metrics were affected most by training on imbalanced data? Which were affected least? Include justification.

Note: We set $C = 0.01$ to ensure that interesting trends can be found regardless of your work in question 4. This may mean that your value for C differs in 5 and 4.1(c). **This does not mean your answer for 4.1(c) is wrong if you did not use** $C = 0.01$**.** In a real machine learning setting, you'd have to be more careful about how you compare models.

(c) (1 pt) Does the F1-Score match your intuitions for training on imbalanced data? Why or why not? Justify your response using the formulation for the F1-Score. Hint: does the precision match your intuitions?

5.3 Choosing appropriate class weights [6 pts]

(a) (4 pt) Now we will return to setting the class weights given the situation we explored in Part 5.2. Using what you have done in the preceding parts, find an appropriate setting for the class weights that mitigates the situation in part 5.2 and improves the classifier trained on the imbalanced data set. That is, find class weights that give a good mean cross-validation performance (Think: which performance metric(s) are informative in this situation, and which metric(s) are less meaningful? Make sure the metric you use for cross-validation is a good choice given the imbalanced class weights). Report here your strategy for choosing an appropriate performance metric and weight parameters. This question requires you to choose hyperparameters based on cross-validation; **you should not be using the test data to choose hyperparameters**. You should perform this cross-validation over a reasonably large range of weight values (i.e 1-2 orders of magnitude).

(b) (2 pt) Use your classifier to predict the provided `IMB_test_labels` using `IMB_test_features` again, and report the accuracy, specificity, sensitivity, precision, AUROC, and F1-Score of your predictions:

Class Weights	Performance Measures	Performance
$W_n = ?, W_p = ?$	Accuracy	
$W_n = ?, W_p = ?$	F1-Score	
$W_n = ?, W_p = ?$	Auroc	
$W_n = ?, W_p = ?$	Precision	
$W_n = ?, W_p = ?$	Sensitivity	
$W_n = ?, W_p = ?$	Specificity	

5.4 The ROC curve [3 pts]

Another way to understand the impact of class weights when training on imbalanced data is to study the difference in ROC curves for a weighted and unweighted classifier. Using data from the imbalanced dataset, provide a plot of the ROC curve with labeled axes for both $W_n = 1, W_p = 1$ and your custom setting of W_n, W_p from above. Put both curves on the same set of axes. Make sure to label the plot in a way that indicates which curve is which.

6 Challenge [14 pts]

Now, a challenge: In the previous sections, we solved a binary classification problem by only considering comments with a label of 1 or -1, and ignoring comments with a label of 0.

For this challenge, you will consider the original multiclass labels of the comments. For this challenge, we have already prepared a held-out test set `heldout_features` for this challenge, as well as a multiclass training dataset stored `multiclass_features, multiclass_labels`. You can access these via the `helper.py` functions `get_heldout_features` and `get_multiclass_training_data`, respectively. **You must work only with the provided data; acquiring new data to train your model is not permitted.** (Notice, if you look into the dataset, there may be additional information that could

be leveraged. See appendix C for more.) Your goal is to learn a multiclass classifier using the `SVC` or `LinearSVC` class to predict the true ratings of the held-out test set. You will do this by training your model on `multiclass_features` and test on `heldout_features`. If you wish to take advantage of the other features in the dataset, you will have to modify either the `get_multiclass_training_data` function in `helper.py` or the `generate_feature_matrix` function in `project1.py`.

Note that the class balance of this training set matches the class balance of the heldout set. Also note that, given the size of the data and the feature matrix, training may take several minutes.

In order to attempt this challenge, we encourage you to apply what you have learned about hyperparameter selection, and also to consider the following extensions:

1. **Try different feature engineering methods**. The bag-of-words models we have used so far are simplistic. There are other methods to extract different features from the raw data, such as:
   (a) Using a different method for extracting words from the comments
   (b) Using only a subset of the raw features
   (c) Using the number of times a word occurs in a comment as a feature (rather than binary $0, 1$ features indicating presence)
   (d) Include phrases from ratings in addition to words.
   (e) Scaling or normalizing the data
   (f) Alternative feature representations
2. **Read about one-vs-one and one-vs-all**. These are the two standard methods to implement a multiclass classifier using binary classifiers. You should understand the differences between them and implement at least one.

You may use other packages as long as you are training an SVM model. In the nltk package, you are only allowed to use stemming, lemmatization, stop words, and position tagging. In addition, you are not allowed to use pretrained models of any kind. You will have to save the output of your classifier into a `csv` file using the helper function `generate_challenge_labels(y, uniqname)` we have provided. The base name of the output file must be your uniqname followed by the extension `csv`. For example, the output filename for a user with uniqname `foo` would be `foo.csv`. This file will be submitted according to the instructions at the end of this section and/or on the first page of this spec. You may use the file `test_output.py` to ensure that your output has the correct format. To run this file, simply run `python test_output.py -i uniqname.csv`, replacing the file `uniqname.csv` with your generated output file.

We will evaluate your performance in this challenge based on the following components:

1. Write-Up and Code [8 pts]: We will evaluate how much effort you have applied to attempt this challenge based on your write-up and code. **Ensure that both are present.** Within your write-up, you must provide discussions of the choices you made when designing the following components of your classifier:
   - Feature engineering
   - Hyperparameter selection

- Algorithm selection (e.g., quadratic vs. linear kernel)
- Multiclass method (e.g., one-vs-rest vs. one-vs-all)
- Any techniques you used that go above and beyond current course material

2. Test Scores [6 pts]: We will evaluate your classifier based on accuracy. Consider the following confusion matrix:

	-1	**0**	**1**
-1	x_1		
0		x_2	
1	y_1		x_3

where each column corresponds to the actual class and each row corresponds to the predicted class. For instance, y_1 in the matrix above is the number of comments with true rating -1 (sadness), but are classified as a comment with rating 1 (gratitude) by your model. The accuracy for a multiclass classification problem is defined as follows:

$$\text{accuracy} = \frac{x_1 + x_2 + x_3}{n}$$

where n is the number of samples. **Note**: We do not expect your classifier to have perfect accuracy. All challenge test scores will be considered in the context of the performance of the class at-large.

Include your entire project code (copy/paste or screenshot) as an appendix in your report submission. Please try to format lines of code so they are visible within the pages.

Upload your file `uniqname.csv` containing the label predictions for the held-out data to the canvas assignment named Project 1 Challenge Submissions.

Appendix A: Approximate Run-times for Programming Problems

- **Problem 4.3 a i**: around 60-90 minutes
- **Problem 4.3 a ii**: around 30-40 minutes
- **Problem 5.3**: around 10 minutes
- **All other coding problems**: less than 1 minute

These are approximate times, not exact. Different computers will result in different run-times, so do not panic if yours is a little different. Algorithmic optimization can also improve run-time noticeably in certain cases. However, if it is taking more than twice as long, something may be wrong with either your implementation or your hardware. To help save time, we recommend testing your implementations with the debug dataser before running them on the larger dataset when possible. Students facing significant hardware challenges can use Google Colab or CAEN computers to run their solutions, or reach out to EECS 445 Staff for more assistance.

Appendix B: Topics and Concepts

The relevant topics for each section are as follows:

- **Problem 4.1** a, b, e, f, **Problem 4.2**,
  - Support Vector Machines; Primal Formulation; Geometric Margin; Loss Functions and Regularization
- **Problem 4.1** a, **Problem 4.4**, **Problem 5.1** a
  - Dual Formulation; Kernels
- **Problem 4.1** c, d, **Problem 4.3** b, **Problem 5.1** b, c, **Problem 5.2** a, b, **Problem 5.3** a, b, **Problem 5.4**
  - Performance Measures

Appendix C: Further Reading

Below are some topics (in no particular order) you may find useful to research for the challenge portion of this project. This is not an exhaustive list, nor do we know for certain that they will improve your classifier performance, but they are avenues for you to explore.

1. Term Frequency - Inverse Document Frequency (TF-IDF)
2. Topic Modeling (Latent Dirichlet Allocation)

3. Data Augmentation[1]
4. Stemming and Lemmatization
5. Part-of-speech tagging and position[2]
6. N-grams

[1] https://www.aclweb.org/anthology/D19-1670.pdf
[2] https://www.aclweb.org/anthology/W02-1011.pdf