UNIVERSITY OF MICHIGAN
Department of Electrical Engineering and Computer Science
EECS 445 — Introduction to Machine Learning
Winter 2024 Project 1 - Sachchit's Screenplay SVMs

Due: Tuesday, 2/13 at 10:00pm

Section	Points	Recommended Completion Date
2. Feature Extraction	12	Monday, 2/5
3. Hyperparameter and Model Selection	35	Monday, 2/5
4. Asymmetric Cost Functions and Class Imbalance	20	Wednesday, 2/7
5. Identifying Bias	11	Thursday, 2/8
6. Challenge	14	Monday, 2/12
7. Code Appendix	8	Monday, 2/12

This spec contains 20 pages, including Appendices with approximate run-times for programming problems, as well as a list of topics, concepts, and further reading.

You will submit the project components to three separate Gradescope assignments:

- Submit your write-up for sections 2-6 to the Gradescope assignment titled "Project 1 Writeup"
- Submit your file `uniqname.csv` containing the label predictions for the heldout data to the Gradescope assignment titled "Project 1 Challenge Submission".
- Submit all of your project code to the Gradescope assignment titled "Project 1 Code Appendix". You should include any code from `project1.py` and `helper.py`, as well as any additional functions or code you wrote to generate the output you reported in your write-up. You can submit your code as is, including any comments or print statements. Accepted file formats for the Gradescope submission include py files, zip files, ipynb files, and PDFs of your code. Your code appendix will be manually graded for effort and completeness.

1 Introduction

Sachchit loves watching movies. In fact, he watches one every evening before going to bed. Unfortunately, being a very critical movie connoisseur, he spends hours reading reviews before making his choice, so he tends to fall asleep before the end of the movie. To rub more salt into the wound, he forgot to cancel his Amazon Prime student subscription last month, so now he has no choice but to watch all the interesting movies on Amazon Prime Video. Thankfully, Sachchit now has a group of EECS 445 students at his disposal who have become well-versed in solving complex supervised Machine Learning problems. To find the most critically-acclaimed movies, he plans to train a model to deduce the sentiments of their Amazon reviews (i.e. determine if the review author thinks the movie is worth watching).

In this project, we have given you review data from Prime Video's large catalogue of movies. The Amazon dataset contains thousands of reviews and ratings from different users. You will work with this dataset to

train various Support Vector Machines (SVMs) to classify the sentiment of a review. That way Sachchit can automate the process of choosing and will be able to finish watching his movie before falling asleep. In this process, we will also explore some very useful `scikit-learn` packages and data science techniques.

1.1 Requirements:

1. Python (https://www.python.org/downloads/), with a **Python 3.11** virtual environment.
2. `scikit-learn` (1.3.0): documentation available at https://scikit-learn.org/1.3/
3. `numpy` (1.25.2): documentation available at https://numpy.org/doc/1.25/index.html
4. `pandas` (2.1.0): documentation available at https://pandas.pydata.org/pandas-docs/version/2.1/index.html
5. `matplotlib` (3.7.2): documentation available at https://matplotlib.org/3.7.2/index.html
6. `gensim` (4.3.2): documentation available at https://radimrehurek.com/gensim/auto_examples/

See the Python setup guide in Canvas under Files/Quickstart/Python Quickstart for help on installing Python and using a virtual environment. To install the correct versions of the required packages, run the command `pip install -r requirements.txt` while your virtual environment is activated.

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 Amazon movie reviews. If you open `dataset.csv`, you will find that it has 5 columns: *reviewText*, *unixReviewTime*, *reviewTime*, *rating*, and *label*. Each row in the file corresponds to one review.

- The *reviewText* column contains the text of the actual review.

- The *unixReviewTime* and *reviewTime* columns respectively contain the unix time and human-friendly time when the review was posted.
- The *rating* column contains the rating the reviewer gave the movie, ranging from 1 to 5 stars.
- The *label* column is a multiclass label representing the sentiment of the review: 1 if **positive** (4 or 5 star rating), 0 if **neutral** (3 stars), and -1 if **negative** (1 or 2 stars).

You will use the *reviewText* and *label* columns for most of the project. We will ignore the reviews labeled 0 in order to make the label binary. The final challenge portion, however, will utilize all labeled reviews including those with label 0.

The file `helper.py` provides helper functions for behavior such as reading in the data from `csv` files or graphing certain plots. 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.

Do not change any of the implementations in `helper.py` (for sections 2-5) or `test_output.py`

Run your implementation using the debug dataset `data/debug.csv` and compare your results against `debug_output.txt` to check the correctness of your code before running it on `data/dataset.csv`. **Please note that this is not an exhaustive test case suite.** Other points to consider when using the debug dataset:

- 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 the debug dataset, you should feel confident running your code on the actual dataset.
- **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`
  2. You use the `random_state=445` keyword argument when instantiating your SVMs

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`
- `train_word2vec`
- `compute_association`
- Optional: `performance`

1.3 Submitting Your Work

This project contains both coding and written response questions. **Coding questions will be highlighted green, and questions that require written answers will be highlighted blue.**

Please submit the project components to three separate Gradescope assignments:

- Submit your write-up for sections 2-6 to the Gradescope assignment titled "Project 1 Writeup"
- Submit your file `uniqname.csv` containing the label predictions for the heldout data to the Gradescope assignment titled "Project 1 Challenge Submission".
- Submit all of your project code to the Gradescope assignment titled "Project 1 Code Appendix". You should include any code from `project1.py` and `helper.py`, as well as any additional functions or code you wrote to generate the output you reported in your write-up. You can submit your code as is, including any comments or print statements. Accepted file formats for the Gradescope submission include py files, zip files, ipynb files, and PDFs of your code. Your code appendix will be manually graded for effort and completeness.

2 Feature Extraction [12 pts]

Given a dictionary containing d unique words, we can transform a review into a feature vector of length d, by setting the i^{th} element of the feature vector to 1 if the i^{th} word is in the review and 0 otherwise.

For example, if {`'movie':0, 'was':1, 'the':2, 'best':3`} are the only words encountered in the training data, the review *"BEST movie ever!!"* would map to the feature vector $[1, 0, 0, 1]$.

Note that

- We do not consider case ("Best" and "beSt" are the same)
- Since the word "ever" was not in the original dictionary, it is ignored as a feature

Since we use the training data to construct our dictionary for the test data set, there may be words in the test data that you do not encounter in the dictionary. There are many potential 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 the `replace()` method for strings. No external library is needed. You should only consider punctuation that is in `string.punctuation`.

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

- The average number of non-zero *features* per review in the training data. You will need to calculate this on `X_train`.
- The word appearing in the greatest number of reviews. You will need to make use of the dictionary returned by `get_split_binary_data` (this has already been called for you in the `main()` function of `project1.py`).

3 Hyperparameter and Model Selection [35 pts]

In section 2, you implemented functions that can transform the reviews 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 3.**

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

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 3.1 and 3.2, we will make use of the `sklearn.svm.LinearSVC` class. In 3.3 we will make use of the `sklearn.svm.SVC` class.

In addition, we will use the following methods in both classes: `fit(X,y)`, `predict(X)` and `decision_function(X)` – 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)` produces 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 functions 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

Read the documentation carefully, as when calling this function you should set `labels=[-1,1]` for a deterministic ordering of your confusion matrix output.

3.1 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 negative) 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 in `cv_performance`, passing in instances of `LinearSVC()` with a range of values for C and the appropriate parameters from `select_param_linear()`. Please use the following parameters for any `LinearSVC()` instances you use in Section 3.1: `loss = "hinge"`, `penalty = "l2"`, `dual = True`, `random_state = 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 chosen in powers of 10 between 10^{-3} and 10^{3} (i.e. $10^{-3}, 10^{-2}, \ldots, 10^{3}$) 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.

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 pts) 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 pts) Finish the implementation of `plot_weight(X, y, penalty, C_range, loss, dual)`. 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.

Once you implement the function, use it to plot the L0-norm $\|\bar{\theta}\|_0$ against $C \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ and save it to a file. Use the complete training data `X_train, Y_train` (i.e, don't use cross-validation for this part). Use `loss = "hinge", penalty = "l2"`, and `dual=True`.

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

(e) (2 pts) 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 review 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`. Set the other parameters as you did in earlier parts. 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. If two words have the same coefficient, they can be listed in any order.

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

3.2 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. We will use the L1 penalty and squared hinge loss corresponding to the following optimization problem:

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

where $Loss(z) = \max\{0, (1-z)\}^2$. 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, random_state=445).
```

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

(b) (1 pt) Similar to 3.1(d), plot the L0-norm of the learned parameter $\bar{\theta}$ against $C \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ 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 L0-norm graphs that you generated for the L1 penalty in 3.2(b) and the L2 penalty in 3.1(d). What is the range of L0-norm values for each graph? How do the gradients of the L1 and L2 penalty influence this?

(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?

3.3 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 pts) 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', random_state=445).`

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 can take a substantial time to run (expect to wait for up to 75 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 range $[a, b]$ to yield exponents x_c and x_r, and produce corresponding hyperparameters of $C = 10^{x_c}$ and $r = 10^{x_r}$.

Sample 25 pairs for (x_c, x_r) to generate a total of 25 hyperparameter combinations $(C = 10^{x_c}, r = 10^{x_r})$ to test during your random search. Sample C and r values from the range $[10^{-2}, 10^{3}]$. Expect to wait up to 40 minutes for this code to complete.

Note: It is possible that a correct implementation on random search returns different results for the debug output due to variations in computing platforms.

Find the best C and r value for AUROC using both tuning schemes mentioned above and the training data from question 2. Compute and report the AUROC using the test set.

Note: the next question asks you to compare performance between all C and r pairs for both grid and random search. To save time, we recommend printing any necessary information now so you don't have to rerun this code for the next question.

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 similar to the following:

Tuning Scheme	C	r	AUROC
Grid Search			
Random Search			

(b) (6 pts) Look at the AUROC score for every (C, r) pair tested during grid search. What trends do you notice in how the classifier's AUROC changes as you change C and/or r? Also, give one advantage of grid search over random search and give one advantage of random search over grid search

3.4 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 pts) 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: $(\bar{x} \cdot \bar{x}' + r)^2$. Use $x_1, x_2, ..., x_n$ and constant r to express your answer.

(b) (2 pts) 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.

4 Asymmetric Cost Functions and Class Imbalance [20 pts]

The training data we have provided you with so far is *balanced*: the data contains an equal number of positive and negative reviews. 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. For some applications, an imbalanced dataset can introduce unwanted biases into our model.

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 ($y^{(i)} = -1$).

4.1 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 3.1(c), which performance metrics increased due to the new class weights? Which decreased? 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 3. This may mean that your value for C differs in 4 and 3.1(c). **This does not mean your answer for 3.1(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.

4.2 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 a dataset with class imbalanced data. Using the functions you wrote in part 2, 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 provided class-imbalanced data `IMB_features, IMB_labels`. Use this classifier to predict the labels of the provided test data `IMB_test_features`, and report the accuracy, F1-Score, AUROC, precision, sensitivity, and specificity of your predictions when compared to the true labels `IMB_test_labels`:

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 3.1(c), which performance metrics increased the most by training on imbalanced data? Which decreased the most? Include justification.

Note: We set $C = 0.01$ to ensure that interesting trends can be found regardless of your work in question 3. This may mean that your value for C differs in 4 and 3.1(c). **This does not mean your answer for**

3.1(c) is wrong if you did not use $C = 0.01$. In a real machine learning setting, you should 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 F1-Score's formulation, which is based on both precision and sensitivity. Hint: does the precision match your intuitions?

4.3 Choosing appropriate class weights [6 pts]

(a) (4 pts) Now we will return to setting the class weights given the situation we explored in Part 4.2. Using what you have done in the preceding parts, find and report an appropriate setting for the class weights that mitigates the situation in part 4.2 and improves the classifier trained on the imbalanced data set. That is, **find class weights that give a good mean cross-validation performance.**

Report your strategy for choosing an appropriate performance metric and weight parameters. Consider 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.

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. Please note that there are many correct answers to this question, and the solution in `debug_output.txt` is only one sample answer for the debug dataset.

Note: As in part 4.2, use a linear-kernel SVM with hinge loss, L2-penalty, and $C = 0.01$.

(b) (2 pts) Use your classifier to predict the provided `IMB_test_labels` using `IMB_test_features` again, and report the accuracy, F1-Score, AUROC, precision, sensitivity, and specificity of your predictions. Make sure to use the same final weight parameters from (a). **Note: If your debug values for this section specifically don't work but all previous parts do, try running all previous parts before running 4.3b.**

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	

4.4 The ROC curve [3 pts]

(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 with labeled axes of the ROC curve 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 curves in a way that indicates which curve is which.

5 Identifying Bias [11 pts]

Machine learning models are increasingly relied upon for critical decision-making, such as medical diagnoses and hiring decisions. In such situations, it is essential that we do not deploy models that learn biased criteria and discriminate against certain groups or individuals. In this section, we will investigate potential sources of bias in the Amazon dataset and see how that bias might influence our models.

5.1 Dataset Exploration [5 pts]

Here we will evaluate the Amazon dataset for potential gender bias[1] Specifically, we will assess how the dataset represents actors versus actresses. Make sure to use `filename="data/dataset.csv"` as your dataset when reporting your final results. For parts (a) through (c), we recommend looking through the implementation of the provided functions in `helper.py` to ensure you understand what each function is doing as well as the format of the output.

(a) (1 pt) Run the function `count_actors_and_actresses()` provided in `helper.py`. Report the number of reviews containing the words 'actor' and/or 'actors' and the number of reviews containing the words 'actress' and/or 'actresses'. How do these counts differ? Suggest one possible reason for this difference.

(b) (1 pt) Run the function `plot_actors_and_actresses()` provided in `helper.py`. Use the parameter `x_label='label'`. This function will plot the distribution of labels for reviews containing 'actor(s)' and reviews containing 'actress(es)'. This plot will be saved to your working directory. As a reminder, a label of -1 corresponds to a negative review, a label of +1 corresponds to a positive review, and a label of 0 is neutral. Include the plot in your write-up. How do the two distributions differ, and how might this indicate potential implicit bias among reviewers?

(c) (1 pt) Re-run the function `plot_actors_and_actresses()` with the parameter `x_label='rating'`. This shows the distribution of original ratings in reviews mentioning 'actor(s)' and 'actress(es)' rather than the distribution of labels. Include the plot in your write-up.

Ratings range from 1 to 5 stars, and for this project we converted them to labels as follows:

- 1 or 2 star rating $\rightarrow$ label of -1
- 3 star rating $\rightarrow$ label of 0
- 4 or 5 star rating $\rightarrow$ label of +1

Suppose you find out that the dataset was noisy and some of the datapoints were mislabeled! In particular, in the dataset, suppose all ratings of 5 and 1 were accidentally swapped *only for reviews mentioning 'actor(s)'*. Thus, in the plot denoted 'Actor', the proportion reported for the rating 5 corresponds to the proportion for rating 1 and vice versa. In the true data distribution (where there is no mislabeling), does the bias observed in question 5.1(b) still persist? Explain your reasoning.

Note: This is a thought experiment! This is the only question in this project where you should assume the ratings were mislabeled.

[1] Note that we work with binary genders in this analysis. An extension to non-binary analysis would be similar but would require additional data. A recent study of gender bias in embeddings is available here.

(d) (2 pts) Use the `LinearSVC` class to train a Linear SVM on the same `X_train` and `Y_train` as the other parts of this project. Set C = 0.1, L2 penalty, hinge loss, and random state 445. In the learned classifier, report the theta vector's coefficients for the words 'actor' and 'actress'. How do these coefficients indicate that the model may have learned a bias from the dataset? How does this demonstrate the importance of interpretability in machine learning?

5.2 Word Embeddings [3 pts]

A word embedding is a way to represent a word as a multi-dimensional numerical vector. Word embeddings are designed to capture the semantic meaning of words, and the spatial proximity of two word embeddings reflects their semantic similarity. For example, the word embeddings for words with similar meanings (e.g. "dog" and "cat") will be located close to each other. In this section, we will use the Amazon dataset to train a Word2Vec model that learns embeddings for the words in this dataset. In the next section, we will evaluate these word embeddings for potential bias.

Note: While the project settings make this unlikely, in a small number of cases, the word embeddings learned by Word2Vec may differ on different computers. Due to these potential differences, sections 5.2 and 5.3 do not have associated debug output in `debug_output.txt`.

In general, `debug_output.txt` is not meant to be an exhaustive test set. So while true for the entire project, be especially sure to test your implementation logic separately for sections 5.2 and 5.3, as in case of such an output discrepancy, we will check your attached code. We strongly recommend writing test cases to validate the logic of your implementation. For example, you may consider testing `cosine_similiarity()` in 5.3 by creating test cases comparing the output from your code to manually computed values.

(a) (2 pts) First, we will train a Word2Vec model on the Amazon dataset. Complete the function `train_word2vec()` provided in the starter code, using the parameter `workers=1` in the Word2Vec constructor. It may help to review the documentation here. Using the returned model, print out the word embedding for the word 'actor' and report the dimensionality of this word embedding.

(b) (1 pt) Using the Word2Vec model trained in question 5.2(a), report the five most similar words to the word 'plot'. Recalling that the Amazon dataset contains movie reviews, how has the context of the dataset influenced the word embeddings? How might the most similar words have been different if the dataset had been about geometry instead of movie reviews?

5.3 Word Embedding Association Test [3 pts]

The **Word Embedding Association Test (WEAT)** is a standard measure of bias in word embeddings. For a target word which we suspect of having a biased embedding representation, WEAT defines a metric to compare this target word against the attributes along which it might be biased.

This section will walk you through using the metrics defined by WEAT to evaluate how word embedding vectors from Word2Vec may encode gender bias.

(a) (1 pt) Given a word embedding $\bar{w}$ and two sets A and B containing embeddings for sets of attribute words, WEAT defines the **association** between $\bar{w}$ and sets A and B as:

$$s(\bar{w}, A, B) = \operatorname*{mean}_{\bar{a} \in A} \cos(\bar{w}, \bar{a}) - \operatorname*{mean}_{\bar{b} \in B} \cos(\bar{w}, \bar{b})$$

where $\cos(\bar{x}, \bar{y})$ is the cosine similiarity between the embedding vectors $\bar{x}$ and $\bar{y}$. The association is used to determine whether $\bar{w}$ is more similar to the words represented in A or the words represented in B.

Implement the above function definition by completing the function `compute_association()` provided in the starter code. Using this function, report the association between the word embedding of "smart" ($\bar{w}$) and the sets:

$$A = \{\text{her, woman, women}\}$$
$$B = \{\text{him, man, men}\}$$

(b) (2 pts) Based on the association you found in 5.3(a), do the word embeddings associate the word 'smart' more with the female-gendered attributes or male-gendered attributes? How can you tell from the sign of the association? How do these results relate to the bias identified in 5.1(b)?

6 Challenge [14 pts]

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

For this challenge, you will consider the original multiclass labels of the reviews. 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.

In `main()` in `project1.py`, we have already provided code that will read in data from `dataset.csv` and `heldout.csv` to produce the multiclass training set (`multiclass_features`, `multiclass_labels`) and heldout test set `heldout_features` respectively.

We created the training set and test set by calling the `helper.py` functions `get_multiclass_training_data()` and `get_heldout_reviews()`, which call some of the functions you implemented earlier in the project. **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 details.)

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` (making modifications to `helper.py` is allowed in this section) or the `generate_feature_matrix()` function in `project1.py`. Ideally, you would put the modified versions in new functions, so that all of the necessary code for other parts is still present.

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 used in sections 2 through 4 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 reviews

(b) Using only a subset of the raw features

(c) Using the number of times a word occurs in a review as a feature (rather than binary $0, 1$ features indicating presence)

(d) Include phrases from the reviews 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. Specifically,

- 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 may not use word embeddings of any kind (including those trained in Section 5 of the project).

You will save the predictions made by your classifier on the heldout set to a `csv` file using the helper function `generate_challenge_labels(y, uniqname)` that 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-one 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 reviews that have a true label of -1 (negative), but are classified as a review with label 1 (positive) 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.

7 Code Appendix [8 pts]

Submit all of your project code to the Gradescope assignment titled "Project 1 Code Appendix". You should include any code from `project1.py` and `helper.py`, as well as any additional functions or code you wrote to generate the output you reported in your write-up. You can submit your code as is, including any comments or print statements. Accepted file formats for the Gradescope submission include py files, zip files, ipynb files, and PDFs of your code. Your code appendix will be manually graded for effort and completeness.

Appendix A: Approximate Run-times for Programming Problems

- **Problem 3.3 a i**: around 80 minutes
- **Problem 3.3 a ii**: around 60 minutes
- **All other coding problems**: less than 1 minute

These are approximate times. 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 dataset before running them on the larger dataset when possible. As an alternative option, students 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 3.1** a, b, e, f, **Problem 3.2**
  - Support Vector Machines; Primal Formulation; Geometric Margin; Loss Functions and Regularization
- **Problem 3.4**, **Problem 4.1** a
  - Dual Formulation; Kernels
- **Problem 3.1** c, d, **Problem 3.3** b, **Problem 4.1** b, c, **Problem 4.2** a, b, **Problem 4.3** a, b, **Problem 4.4**
  - 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[2]

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

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

Appendix D: Operating System Reference for IAs/GSIs

For setup help specific to your operating system, please refer to the table below to see which IAs/GSIs use which operating systems.

IA/GSI Name	Operating System
Akseli	MacOS
Alex	Windows
Emre	Windows
Evan	Linux (Ubuntu)
Jason	MacOS
Jiangnan	MacOS
Julie	MacOS
Madhavan	MacOS
Matthew	Windows
Oskar	MacOS
Sachchit	MacOS
Tiffany	Windows

Table 1: Operating Systems used by each IA/GSI

[3] https://www.aclweb.org/anthology/W02-1011.pdf