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

Due: Wednesday, 2/9 at 10:00pm

Section	**Points**	**Recommended Completion Date**
2. Ethics	9	Friday, 1/28
3. Feature Extraction	12	Friday, 1/28
4. Hyperparameter and Model Selection	35	Wednesday, 2/2
5. Asymmetric Cost Functions and Class Imbalance	20	Friday, 2/4
6. Challenge	14	Tuesday, 2/8
Code Appendix	10	Tuesday, 2/8

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. 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

Serafina has been tasked with monitoring Piazza for EECS 445. Part of her responsibilities include boosting student morale, as well as forwarding any grateful Piazza posts to the rest of the staff to boost staff morale. To help her find these posts, she's decided to enlist a group of EECS 445 students (that's you!) who have become well-versed in solving supervised machine learning problems. She would like a classifier that will decide whether a post displays sadness or gratitude.

Since the historical log of EECS 445 Piazza posts does not result in a large enough dataset, Serafina will test the feasibility of this classification task on Reddit data. The Reddit comments contains thousands of different phrases across many users, and the main two emotions you will identify are gratitude and sadness. 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 3.9 virtual environment.

2. `scikit-learn` (1.0.2): https://scikit-learn.org/stable/index.html

3. `numpy` (1.22): http://www.numpy.org/
4. `pandas` (1.3.5): https://pandas.pydata.org/
5. `matplotlib` (3.5.1): https://matplotlib.org/

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 file `dataset.csv` has Reddit comments. These csv files have 4 columns: *text*, *created_utc*, *label*, and *emotion*. Each row in the csv 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. 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.

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. There may be additional functions that you will have to implement on your own:

- `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.

2 Ethics [9 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. 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?

Solution: Answers may vary.

Example:

i) Yes, it is possible to identify an individual in this dataset. One can do an exact search for the full text of the comment to discover the user who posted the comment. With information from the user's post history, it may be possible to determine the identity of that user.

ii) One application of this dataset would be to classify sad comments and offer support to those users. Another application might be to build a language model for gratitude text. A language model is a probability distribution over a sequence of words. With a language model trained on the gratitude text, we can both identify the likelihood that a given text is gratitude as well as generate new pieces of text that are likely to be gratitude.

iii) This dataset was collected from the social media site Reddit; it might be a bad idea to use this dataset for application in unrelated domains. For example, we might not want to build a sadness classifier and use that when processing government paperwork (where the language being used differs significantly from the casual language on Reddit.) It might also be a bad idea to apply this dataset to forms of communication that follow a different structure than Reddit. For example, Piazza posts have a "Question - Answer" structure that's different from the coversational structure of Reddit comments. This means the language being used might also be different, and we should take greater care in evaluating the model on in-domain data (like Piazza posts and comments) before deploying into production. Finally, it would be unethical to use this dataset to construct a profile of a user without their consent. It is a violation of privacy to analyze profile information or deanonymize user profiles.

3 Feature Extraction [12 pts]

Given a dictionary containing d unique words, we can transform the n variable-length 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. Given that the four words {`'book':0, 'was':1, 'the':2, 'best':3`} are the only four 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. Also, note that since the word "ever" was not in the original dictionary, it is ignored as a feature. In real-world scenarios, 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:

```
'BEST book ever! It\'s great'
```

Hint: You might find `string.punctuation` along with the method `string.replace()` useful.

Solution: `['best', 'book', 'ever', 'it', 's', 'great']`

(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 provided by running get_split_binary_data on `dataset.csv`. 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.

Solution: $d = 4855$

(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`.

  Solution: Average number of nonzero features: 12.241

- The word appearing in the most number of comments. You will need to make use of the dictionary returned by `get_split_binary_data`.

  Solution: Most common word: i

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 non-positive (i.e., "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)` – 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.

Solution: Stratified splits are important because the fundamental assumption of most ML algorithms is that the training set is a representative sample of the test set i.e., the training and test data are drawn from the same underlying distributions. If the ratio of positive to negative examples (the class balance) differs significantly between the training and test sets (across folds) this assumption will not hold.

(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.

  Solution: Implementations vary.

- 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:

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

  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.

  Solution: Implementations vary.

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**	**Performance**
Accuracy		
F1-Score		
AUROC		
Precision		
Sensitivity		
Specificity		

Solution:

Performance Measures	**C**	**Performance**	**Test Performance**
Accuracy	1	0.9210	0.9199
F1-Score	1	0.9202	0.9205
AUROC	0.1	0.9723	0.9786
Precision	0.01	0.9980	0.9922
Sensitivity	1	0.9106	0.9262
Specificity	0.01	0.9985	0.9938

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.

Solution:

- For sensitivity and F1-score, the performance starts quite low and rapidly increases to the optimal value of C, and then begins to steadily decline.
- For accuracy, AUROC, precision and specificity, the performance increases as C increases before C reaches the optimal value. However, after C reaches the optimal value, the performance drops and then stays relatively constant as C increases.
- Choosing the value of C to optimize for is an open-ended question. Each metric has its own strengths and weaknesses. We will accept any answer as long as your explanation is sensible.

(c) (3 pt) Now, using the value of C that maximizes your chosen performance measure, 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	

Solution:

For $C = 0.01$:

Performance Measures	Performance
Accuracy	0.8884
F1-Score	0.8753
AUROC	0.9680
Precision	0.9922
Sensitvity	0.7831
Specificity	0.9938

For $C = 0.1$:

Performance Measures	Performance
Accuracy	0.9384
F1-Score	0.9355
AUROC	0.9786
Precision	0.9831
Sensitvity	0.8923
Specificity	0.9846

For $C = 1$:

Performance Measures	Performance
Accuracy	0.9199
F1-Score	0.9205
AUROC	0.9734
Precision	0.9149
Sensitvity	0.9262
Specificity	0.9137

(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 $\bar{\theta} \in \mathbb{R}^d$:

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

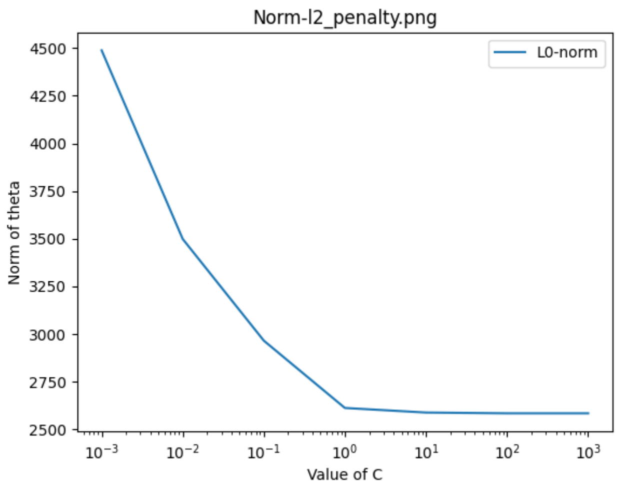
where $\mathbb{I}\{\theta_i \neq 0\}$ is 0 if θ_i is 0 and 1 otherwise.

Solution: Implementations vary.

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.

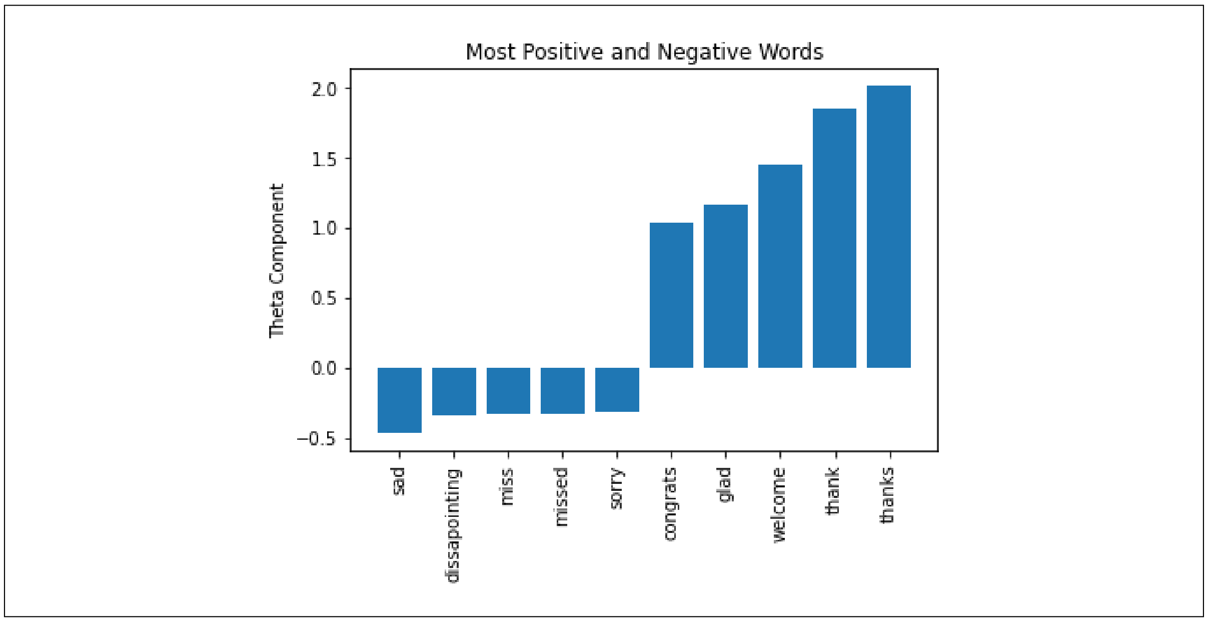
Solution:



(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. The words on the bar chart should be sorted by coefficient value in ascending order.

Solution:



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

Solution: Open-ended.

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:

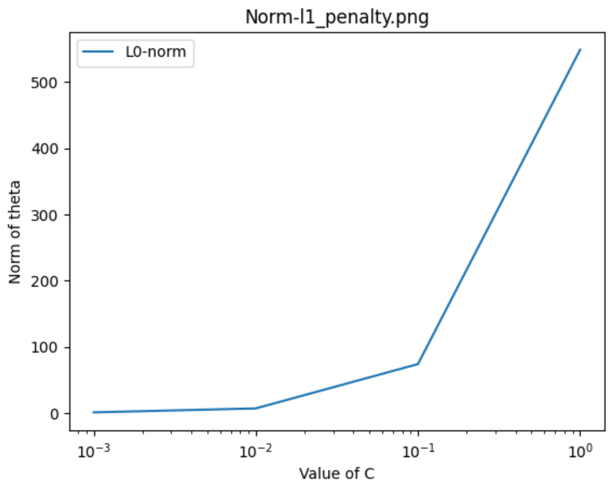
```
LinearSVC(penalty='l1', loss = 'squared_hinge', C=c, dual=False).
```

(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.

Solution: The value of C that maximizes CV performance is $C = 1.0$ with AUROC performance using cross-validation of 0.9682 and AUROC performance on the entire test set of 0.9768.

(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.

Solution:



(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)

Solution:

- First, Looking at scale, we can see that the L1 penalty produces much sparser theta vectors for all settings of C. Since the gradient of the L1 norm is constant with respect to each element, there is just as much value in reducing a small value as there is a large value. This ends up pushing the coefficients of the unimportant features all the way to zero thus resulting in a sparser graph. Likewise, for the L2 norm, the gradient is linear with respect to each element meaning that it more value is gained from reducing large elements and less is gained from reducing small elements. This incentivizes balancing the weights of all features in order to not have expensive large values.

(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?

Solution:

- Squared hinge loss is lenient on the cases where a data point is classified correctly but it is within the margin compared to hinge loss. On the other hand, squared hinge loss punishes misclassifications more harshly. So, we see a comparatively wider margin and more support vectors when using squared hinge loss.

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')`. Make sure to test your implementation with the `test_cases.test_select_param_quadratic()` method.

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. 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 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 (our project solution runs in about 60-90 minutes on our test computer).

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 (our solution runs in about 60 minutes on our test computer).

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			

Solution:

Reported CV Performance

Tuning Scheme	**C**	**r**	**AUROC**
Grid Search	1	100	0.9770
Random Search	419.06	0.16	0.9768

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

Solution:

- How does the 5-fold CV performance vary with C and r?
  - **Random Search -** the performances fluctuates around 0.92 to 0.97 for different pairs of C and r. Even though the result for random search varies and is not deterministic, we do expect your performances to be around 0.9 to 0.97.
  - **Grid Search -** if we fix C, the performance first increases then decreases as r increases in general. If we fix r, the performance generally increases also as C increases.
- Is the use of random search better than grid search?
  - By looking at the performance, random search performed as well as grid search. As random search is not deterministic, we may not get the same set of hyperparameters if we run random search multiple times. Consequently, random search may in some cases perform better or worse than grid search. The best performance of random search is very close to that of grid search most of the time.
    Given that we search through fewer pairs of parameters when using random search compared to grid search, random search tends to be more computationally efficient than grid search. However, as mentioned, the deterministic nature of grid search can be useful for comparing algorithms and models.

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).

Solution:

$$\begin{aligned}
\bar{x} &= [x_1 x_2 ... x_n]^T \\
\bar{x}' &= [x'_1 x'_2 ... x'_n]^T \\
(\bar{x}\cdot\bar{x}' + r)^2 &= (x_1x'_1 + x_2x'_2 + ... + x_nx'_n + r)^2 \\
&= (x_1x'_1)^2 + (x_2x'_2)^2 + ... + (x_nx'_n)^2 + r^2 + 2r(x_1x'_1 + x_2x'_2 + ... + x_nx'_n) \\
&+ 2(x_1x'_1x_2x'_2 + ... + x_{n-1}x'_{n-1}x_nx'_n) \\
&= x_1^2x_1'^2 + ... + x_n^2x_n'^2 + r\cdot r + \sqrt{2r}x_1\sqrt{2r}x'_1 + ... + \sqrt{2r}x_n\sqrt{2r}x'_n \\
&+ \sqrt{2}x_1x_2\sqrt{2}x'_1x'_2 + ... + \sqrt{2}x_{n-1}x_n\sqrt{2}x'_{n-1}x'_n \\
\phi(x) &= \langle r, \underbrace{\sqrt{2r}x_1, \ldots, \sqrt{2r}x_n}_{\text{linear terms}}, \underbrace{\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_1x_n, \sqrt{2}x_2x_3 \ldots, \sqrt{2}x_{n-1}x_n}_{\text{cross terms } x_ix_j, i\neq j}, \underbrace{x_1^2, \ldots, x_n^2}_{\text{square terms}}\rangle
\end{aligned}$$

(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.

Solution: Pros: We retain interpretability and can access feature weights for each feature if we use explicit feature mapping.
Cons: For d features in original data, we have $(d^2 + d + 1)$ features in the transformed data, which is very computationally expensive

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:

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

We can modify it in the following way:

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

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.

> **Solution:** This modification changes the objective function by effectively changing the importance of different points in the minimization process. That is, points in a class with a higher weight (in this case, points with negative labels) are more likely to correspond to a lower slack term ξ_i as the slack term is more highly weighted in the minimization. Therefore, negative-labeled points are more likely to be correctly classified.

(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 change the objective function? Refer to the weighted SVM formulation to justify your answer.

Solution: If the ratios between the class weights are the same, we are essentially changing the C value being applied in the objective function. For example, using the first set of weights is the same as using the second set of weights with $C' = C/4$. Increasing C decreases the relative importance of having a small theta and increases the importance of correctly classifying all of the points.

(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	

Solution:

Performance Measures	**Performance**
Accuracy	0.6174
F1-Score	0.7219
AUROC	0.9628
Precision	0.5673
Sensitivity	0.9923
Specificity	0.2419

(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(e). In a real machine learning setting, you'd have to be more careful about how you compare models.

Solution: Answers will depend based on what metric was optimized for in Question 4.1(e). In general, we would expect sensitivity to increase and specificity to decrease due to the classifier penalizing misclassified positive points more heavily than negative ones.

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 800 positive points and 200 negative 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	

Solution:

Class Weights	**Performance Measures**	**Performance**
$W_n = 1, W_p = 1$	Accuracy	0.8057
$W_n = 1, W_p = 1$	F1-Score	0.8915
$W_n = 1, W_p = 1$	Auroc	0.9512
$W_n = 1, W_p = 1$	Precision	0.8052
$W_n = 1, W_p = 1$	Sensitivity	0.9985
$W_n = 1, W_p = 1$	Specificity	0.0368

(b) (1 pt) Which performance metrics were affected most by training on imbalanced data? Which were affected least? Include justification.

Solution: Training on an imbalanced data set has affected sensitivity, which increased, along with specificity, which decreased significantly. Changing the data set to consist of far more positive samples than negative samples results in the classifier trained on this data tending to classify many more positive samples correctly, as shown by the sensitivity and specificity metrics. The sensitivity (true positive rate) is close to 1, which indicates all positive samples were classified correctly, while the specificity (true negative rate) is close to 0, which indicates none of the negative samples were correctly classified as negative.

(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?

Solution: The F1-Score is not heavily impacted when training on this imbalanced data. This is because the positive points far outnumber the negative points, so the number of true positives is high, while the number of false positives is necessarily low (since there are few negative points to begin with).

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).

Solution: **AUROC** is a good metric to use when we want to evaluate how much the model has learned. AUROC will be high when the model learns the classifications with more confidence, and thus will still have a reflection of overall model performance even with imbalanced data. Weights found using Cross Validation with respect to AUROC across a reasonably large range of weights (Wn=2 to 9, Wp=2 to 9) are: **Wn=9, Wp=7** (CV Score: 0.9687749908615293).

(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	

Solution:

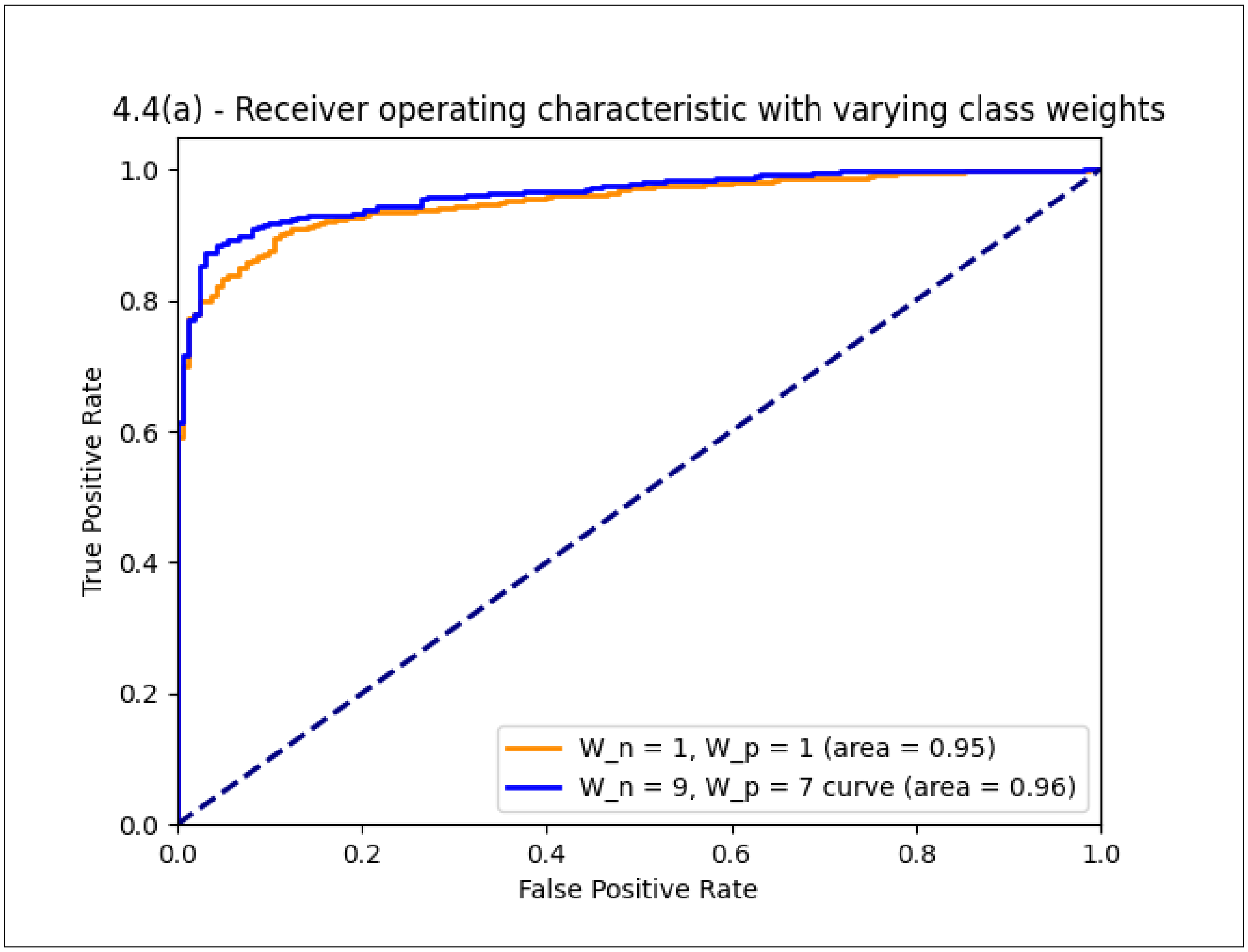
Class Weights	Performance Measures	Performance
$W_n = 9, W_p = 7$	Accuracy	0.9127
$W_n = 9, W_p = 7$	F1-Score	0.9440
$W_n = 9, W_p = 7$	Auroc	0.9626
$W_n = 9, W_p = 7$	Precision	0.9692
$W_n = 9, W_p = 7$	Sensitivity	0.9200
$W_n = 9, W_p = 7$	Specificity	0.8834

Answers may vary based on Wn and Wp weights determined by student in 5.3(a).

5.4 The ROC curve [3 pts]

Given the above results, we are interested in investigating the AUROC metric more. First, 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.

Solution:



6 Challenge [14 pts]

Now, a challenge: in the previous problems, we had transformed the data into a binary dataset by combining multiple labels to generate two labels.

For this challenge, you will consider the original multiclass labels of the comments. We have already prepared a held-out test set `heldout_features` for this challenge. We also provide multiclass training data `multiclass_features, multiclass_labels`. **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.) Your goal is to train a multiclass classifier using the `SVC` or `LinearSVC` classes to predict the true ratings of the held-out test set, i.e., you will train 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 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 ratings
   (b) Using only a subset of the raw features
   (c) Using the number of times a word occurs in a ratings 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 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 the file. 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 three 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

Solution: Answers will vary.

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.

Solution: Answers will vary.

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 b i**: around 60-90 minutes
- **Problem 4.3 b ii**: around 30-40 minutes
- **Problem 4.5**: around 10 seconds
- **Problem 5.3**: around 10 minutes

N.B. 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 might be wrong. To help save time, we recommend testing your implementations first with the provided test cases when possible.

Appendix B: Topics and Concepts

The relevant topics for each section are as follows:

- **Problem 4.1** a, b, e, f, **Problem 4.4** c, d
  - Support Vector Machines; Primal Formulation; Geometric Margin; Loss Functions and Regularization
- **Problem 4.2** a, **Problem 4.3** a, **Problem 5.1** a
  - Dual Formulation; Kernels
- **Problem 4.1** c, d, **Problem 3.2** b, **Problem 4.3** b, **Problem 3.4** a, b, **Problem 5.1** b, c, **Problem 5.2** a, b, **Problem 5.3** a, b, **Problem 5.4** a, b
  - 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]

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

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

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