UNIVERSITY OF MICHIGAN
Department of Electrical Engineering and Computer Science
EECS 445 Introduction to Machine Learning
Winter 2019

Project 1 - Steven's Variety of Methods for Reducing Bad Flights
Due: Tuesday, 02/12 at 11:59pm

1 Introduction

Ever since Steven's horrible flight from LAX to DTW where the stewardess accidentally gave him non-gluten-free water, he has been passionate about finding which airlines are the best. Being a true millennial, he constantly consults every and all opinions on Twitter in order to figure out which airlines have more positive reviews. However, with his other life commitments in the way, he no longer has time to pursue his passion of browsing through all reviews on Twitter. Thankfully, we can harness the power of Machine Learning to solve this imperative task by training a model to deduce the sentiment of the reviews (i.e. determine if the tweet author thinks the airline is dope, wack, or meh).

In this project, we have given you data from Twitter that reviews six different airlines: Sigmoid Express, Jet SVM, Perceptron Airlines, Huimin International, AeroHristo, and Air Kutty. The Twitter dataset contains tens of 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 so that Steven can easily learn which airlines are the best. In this process, we will also explore some very useful `scikit-learn` packages and data science techniques.

1.1 Requirements:

1. Updated version of Anaconda (`https://docs.continuum.io/anaconda/install/`), with a Python 3.6 environment.
2. Updated version of `scikit-learn` (0.20.2) in your Anaconda install: `https://scikit-learn.org/stable/index.html`
3. Updated version of `numpy` (1.15) in your Anaconda install: `http://www.numpy.org/`
4. Updated version of `pandas` (0.24.0) in your Anaconda install: `https://pandas.pydata.org/`
5. Updated version of `matplotlib` (3.0.2) in your Anaconda install: `https://matplotlib.org/`
    - You can verify your version of Anaconda-managed packages by running `conda list`
    - If needed, you can update a package by running `conda update [package name]`
    - You may create an Anaconda environment for the project with only the packages you will need, or you may simply use the Anaconda instance of Python which will make packages managed by Anaconda available for your use. Please reference the Anaconda documentation.

1.2 Getting Started

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

- `data/dataset.csv`
- `data/heldout.csv`
- `data/imbalanced.csv`
- `project1.py`
- `helper.py`
- `test_output.py`

The files `dataset.csv` and `imbalanced.csv` have reviews from Twitter. These csv files have 6 columns: *airline*, *retweet_count*, *text*, *tweet_created*, *usertime_zone*, *label*. Each row in the csv file corresponds to one review. The *text* column contains the text of the actual review. The *label* column is a multiclass label: 1 if positive, 0 if neutral, and -1 if negative.

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

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, along with the helper function `select_classifier` which you may implement to return SVM classifiers depending on the given input parameters. The file `test_output.py` allows you to test your output csv file before submission to make sure the format is correct.

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:

- `extract_dictionary(df)`
- `generate_feature_matrix(df, word_dict)`
- `cv_performance(clf, X, y, k=5, metric=‘accuracy’)`
- `select_param_linear(X, y, k=5, metric=‘accuracy’, C_range=[], penalty=‘l2’)`
- `plot_weight(X, y, penalty, metric, C_range)`
- `select_param_quadratic(X, y, k=5, metric=‘accuracy’, param_range = [])`
- Optional: `select_classifier(penalty=‘l2’, c=1.0, degree=1, r=0.0, class_weight=‘balanced’)`
- Optional: `performance(y_true, y_pred, metric=‘accuracy’)`

2 Feature Extraction [20 pts]

Given a dictionary containing d unique words, we can transform the n variable-length reviews 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} review, and 0 otherwise. Given that the four words {`'the':0, 'airline':1, 'was':2, 'best':3`} are the only four words we ever encounter, the review *"BEST airline ever!!"* would map to the feature vector $[0, 1, 0, 1]$.

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 5.

(a) Start by implementing the `extract_dictionary(df)` function. You will use this function to read all unique words contained in `dataset.csv` into a dictionary (as in the example above). You can start implementing this function by removing all the punctuation in the dataset. While removing punctuation, please make sure that you do not accidentally combine two different words that are separated by a punctuation mark. For instance, after you remove punctuation from *"Airline was awesome!Yay"*, you should produce *"Airline was awesome Yay"*, not *"Airline was awesomeYay"*. After removing all the punctuation, you should convert all the words to lowercase and start building your dictionary. Your function should return a dictionary of d unique words.

Note: You will need to report the number of unique words in 2(c).

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

(b) Next, implement the `generate_feature_matrix(df, word_dict)` function. For each review j, construct a feature vector of length d, where the i^{th} entry in the feature vector is 1 if the i^{th} word in the dictionary is present in review j, or 0 otherwise. 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.

(c) The function `get_split_binary_data` in `helper.py` uses the functions you implemented in (a) and (b). Examine how it is implemented. Then, use `get_split_binary_data` to get the training feature matrix `X_train`.

In your write-up, include the following:

- The value of d which you recorded after extracting the training data (the number of unique words). You should be able to extract d from the size of the training feature matrix.
- The average number of non-zero *features* per rating in the training data. You will need to calculate this on `X_train`.

Solution:

- The value of d which you recorded after extracting the dictionary: 2850
- The average number of non-zero *features* per rating: 15.624

3 Hyperparameter and Model Selection [40 pts]

In the skeleton code, the reviews have been transformed into a feature matrix `X_train` and a label vector `y_train` using the functions you implemented in question 2. Test data `X_test, y_test` has also been read in for you. **You will use these data for all of question 3.** You may notice that `X_train, y_train` only have 1000 reviews, while the `dataset.csv` file has 3000. Here, we only give you a subset of the data to train on. You will work with all 3000 reviews in question 5.

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 "poor or average" and 1 means "good." This is a binary classification problem, which you know how to solve!

For the classifier, we will use SVMs with two different kernels: linear and quadratic. In parts 3.1-3.3 we will make use of the `sklearn.svm.SVC` class. At first, we will explicitly set only two of the initialization parameters of `SVC()`: the `kernel`, and `C`. In addition, we will use the following methods in the `SVC` class: `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 (see the documentation for more details).

As discussed in lecture, SVMs have hyperparameters that 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 the F1-score, it is possible that a divide-by-zero may occur which throws a warning. Consider how this metric is 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 (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.

3.1 Hyperparameter Selection for a Linear-Kernel SVM [20 pts]

(a) To begin, implement the function `cv_performance(clf, X, y, k=5,metric='accuracy')` as defined in the skeleton code. Here you will make use of the `fit(X,y)`, `predict(X)`, and `decision_function(X)` methods in the SVC class. The function returns the mean k-fold CV performance for the performance metric passed into the function. The default metric is `'accuracy'`, however 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 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 this case, the class proportions should be roughly equal across folds, since the original training data has equal class proportions.

You must implement this function without using the `scikit_learn` implementation of CV. You will need to employ the following class for splitting the data: `sklearn.model_selection.StratifiedKFold()`. Do not shuffle points when using this function (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.

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) Now implement the `select_param_linear(X, y, k=5, metric='accuracy', C_range = [], penalty='l2')` function to choose a value of C for a linear SVM based on the training data and the specified metric. Note that scikit-learn uses a slightly different formulation of SVM from the one we introduced in lecture, namely:

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

Essentially, the C is inversely proportional to the λ we used in lecture. Your function should call your CV function (implemented earlier) passing in instances of `SVC(kernel='linear', C=c, class_weight='balanced')` with a range of values for C chosen in powers of 10 between 10^{-3} and 10^{3} (i.e. $10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}$). You may choose to implement and use the helper function `select_classifier` to instantiate the needed classifier.

Solution: Implementations vary.

(c) Finally, using the training data from question 2 and the functions implemented here, 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:

Acceptable answer 1: CV performance

Performance Measures	C	Performance
Accuracy	0.1	0.8390
F1-Score	0.1	0.8377
AUROC	0.1	0.9204
Precision	10	0.8413
Sensitivity	0.1	0.8380
Specificity	10	0.8440

Acceptable answer 2: Test performance

Performance Measures	C	Performance
Accuracy	0.1	0.8325
F1-Score	0.1	0.8295
AUROC	0.1	0.9205
Precision	10	0.8368
Sensitivity	0.1	0.8150
Specificity	10	0.8450

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 several minutes (our project solution time is about 12 minutes for this question on our test computer).

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 performance measure will be used in part d.

Solution:

- For sensitivity, the performance starts high and steadily decreases as C increases. This suggests that if we wanted to optimize this metric, we should be searching smaller values of C.
- For specificity and precision, the performance steadily increases as C increases until plateauing to a constant value as C continues to increase.

- For F1-score, accuracy, and AUROC 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 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.

(d) Now, using the value of C that maximizes your chosen performance measure, create an SVM as in the previous question. Again, you may choose to use the helper function `select_classifier`. 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:

Using $C = 0.1$ which maximizes Accuracy, F1-Score, AUROC and sensitivity performance.

Performance Measures	**Performance**
Accuracy	0.8325
F1-Score	0.8295
AUROC	0.9205
Precision	0.8446
Sensitvity	0.8150
Specificity	0.8500

Using $C = 10$ which maximizes Precision and Specificity performance.

Performance Measures	**Performance**
Accuracy	0.8200
F1-Score	0.8154
AUROC	0.9005
Precision	0.8368
Sensitvity	0.7950
Specificity	0.8450

(e) Finish the implementation of the `plot_weight(X, y, penalty, metric, C_range)` 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 `SVC` object may require you to dig into the documentation. The L0-norm is given as follows, for $\bar{\theta} \in \mathbb{R}^d$:

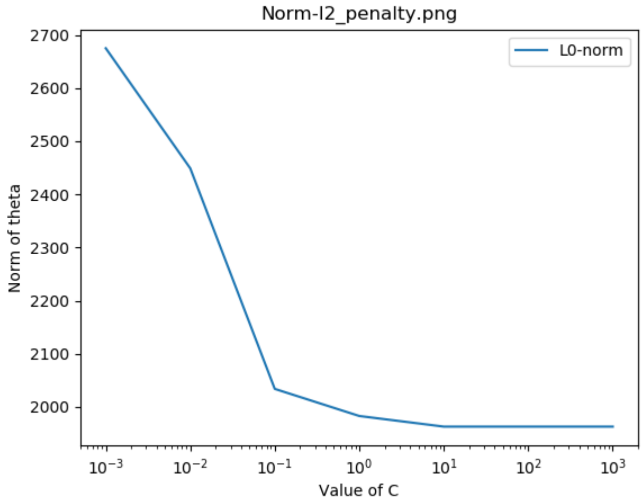
$$\|\bar{\theta}\|_0 = \sum_{i=1}^{d} \mathbb{I}\{\theta_i \neq 0\}$$

where $\mathbb{I}\{a \neq 0\}$ is 0 if a 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 and describe any interesting trends you observe.

Solution:



As the C value increases in the formulation with L2 penalty, the L0 norm decreases until C is approximately $C = 0.1$ ($C = 1$ is also acceptable).
This is potentially due to the increased weight of the slack variables ξ_i in the objective function as C increases, which allows for a larger L2 norm of $\bar{\theta}$. This allows for larger components θ_j in the parameter vector $\bar{\theta}$ which were previously discouraged due to the effect of the L2 norm (squaring elements before summing), and consequently encourages fewer non-zero values θ_j each with larger magnitudes.

(f) Recall that each coefficient of $\bar{\theta}$ is associated with a word. The more positive a coefficient is, 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 $C = 0.1$ (for consistency with our results), train an SVM on `X_train, Y_train`. On this trained SVM, find the top 4 most positive coefficients and the top 4 most negative coefficients of $\bar{\theta}$. Using the dictionary created on the training data `dictionary_binary`, find the words that these coefficients correspond to, and report both the coefficients and the corresponding words. As before, you may choose to use the helper function `select_classifier`.

Positive Coefficient	**Word**

Negative Coefficient	**Word**

Solution:

Positive Coefficient	**Word**
0.595908	good
0.765423	great
0.969453	thanks
0.901084	thank

Negative Coefficient	**Word**
-0.507433	worst
-0.520821	due
-0.615788	hours
-0.549505	delayed

3.2 Hyperparameter Selection for a Quadratic-Kernel SVM [10 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.

(a) Implement the `select_param_quadratic(X, y, k=5, metric='accuracy', param_range = [])` 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, class_weight='balanced')` with the same range of C that we use in 3.1(b). You should also use the same range for r.

Again, you may choose to use the helper function `select_classifier`. 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^{-3} and 10^3) and r (between 10^{-3} and 10^3) [A total of 49 pairs]. This code will take a substantial time to run (our project solution runs in about 17 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 12 minutes on our test computer).

Solution: Implementations vary.

(b) Finally, using the training data from question 2 and the function implemented here, find the best values for C and r for AUROC and both tuning schemes mentioned above. 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. Your table should look be similar to the following:

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

How does the 5-fold CV performance vary with C and r? Also, is the use of random search better than grid search? Give reasons for your conclusion.

Solution:

(c) Acceptable Table 1: Reported CV Performance:

Tuning Scheme	**C**	**r**	**AUROC**
Grid Search	1000	0.1	0.9178
Random Search	31.743918	6.56197	0.9232

(d) Acceptable Table 2: Reported Test Performance:

Tuning Scheme	**C**	**r**	**AUROC**
Grid Search	1000	0.1	0.9191
Random Search	31.743918	6.56197	0.9200

- How does the 5-fold CV performance vary with C and r?
  - **Random Search -** the performances fluctuates around 0.75 to 0.95 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 0.75 to 0.95.
  - **Grid Search -** if we fix C, the performance increases 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 better than 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 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.

3.3 Learning Non-linear Classifiers with a Linear-Kernel SVM [5 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 question 3.3)

(a) Describe a feature mapping, $\phi(\bar{x})$, that maps your data to a feature space similar to the one implied by the quadratic kernel from the question above.

Solution:

$$\phi(x) = \langle r, \underbrace{\sqrt{2r}x_1, \ldots, \sqrt{2r}x_n}_{\text{linear terms}}, \underbrace{\sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \ldots, \sqrt{2}x_1x_n, \ldots, \sqrt{2}x_nx_1 \ldots, \sqrt{2}x_nx_{n-1}}_{\text{cross terms } x_ix_j, i\neq j}, \underbrace{x_1^2, \ldots, x_n^2}_{\text{square terms}}\rangle$$

(b) 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

3.4 Linear-Kernel SVM with L1 Penalty and Squared Hinge Loss [5 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.

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

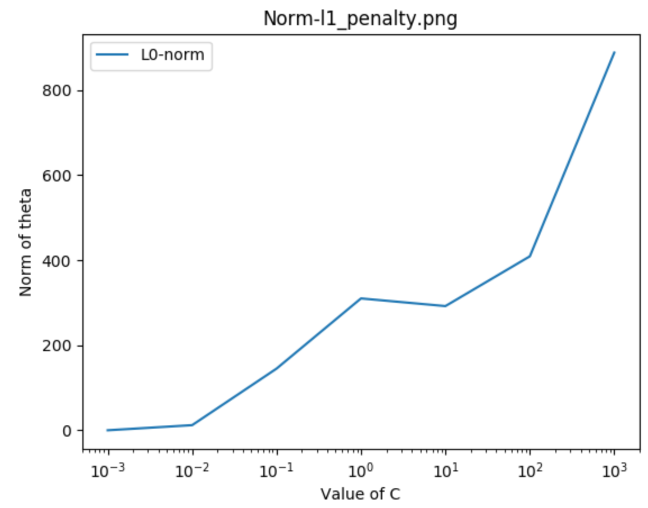
where $\text{loss}(z) = \max\{0, (1 - z)\}^2$. We will make use of the `LinearSVC()` class, which uses the squared hinge loss and allows us to specify the penalty. We will consider only a linear-kernel SVM and the original (untransformed) features. When calling `LinearSVC` please use the following settings: `LinearSVC(penalty=‘l1’, dual=False, C=c, class_weight=’balanced’)`. As always, you may implement and use the helper function `select_classifier` to instantiate your SVM classifier.

(a) Using the training data from question 2 and 5-fold CV, find the best setting for C that maximizes the AUROC using grid search CV with range specified in 3.1(b). When we say ”grid search” here, we mean searching a one-dimensional grid as the only hyperparameter that is changing is C. In the case of ties, report the lower C value. Report your findings.

Solution: The value of C that maximizes CV performance is $C = 1000$ with AUROC performance using cross-validation of 0.9163 and AUROC performance on the entire test set of 0.8896.

(b) Similar to 3.1(e), 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) Beyond any change in performance you may notice, what effect does the $L1$ penalty have on the optimal solution? (**Hint:** Have a careful look at the plots you generated!)

Solution:

- Beyond any change in performance, the L1 penalty produces a sparser solution. When C is small, many weights are forced to zero.

(d) 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 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 (500 ratings per class). But this is not always the case. In many situations, you are given imbalanced data, where one class may be represented more than the others.

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

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

We can modify it in the following way:

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

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) 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 ξ_i is more highly weighted in the minimization. Therefore, negative-labeled points are more likely to be correctly classified.

(b) Create a linear-kernel SVM with hinge loss and L2-penalty with $C = 0.01$. This time, when calling `SVC`, set `class_weight={-1: 10, 1: 1}`, or implement and use your `select_classifier` helper function. This corresponds to setting $W_n = 10$ and $W_p = 1$. 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	
Sensitvity	
Specificity	

Solution:

Performance Measures	**Performance**
Accuracy	0.5625
F1-Score	0.2222
AUROC	0.9050
Precision	1.000
Sensitvity	0.1250
Specificity	1.000

(c) Also, answer the following: Compared to your work in question 3.1(d), 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 3. This may mean that your value for C differs in 4 and 3.1(d). In a real machine learning setting, you'd have to be more careful about how you compare models.

Solution: The metrics that changed the most include accuracy (decreased), F1-score (decreased), precision (increased), sensitivity (decreased), and specificity (increased). AUROC did not change as significantly.

Changing class weights to unequal weights skews the decision boundary towards a specific class, which in this case increased specificity (true negative rate) and decreased sensitivity (true positive rate). This is a result of setting the class weight of the negative class higher than that of the positive class. Doing so also increased the precision, since our classifier more conservatively predicts positive points. Meanwhile, the AUROC measure is calculated using the receiver operating characteristic (ROC) curve which compensates for class skews by including true/false positive rates at different classification thresholds. In addition, accuracy and F1-score decreased due to the inability of the model to classify the data in an unbiased manner, as the class weights in this case favor the negative class over the positive class.

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 class imbalance. 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 300 positive points and 700 negative points. It also comes with a corresponding test feature matrix and label vector pair `IMB_test_features, IMB_test_labels`, which has the same class imbalance.

(a) Create a linear-kernel SVM with hinge loss, L2-penalty and as before, $C = 0.01$. Set `class_weight={-1: 1, 1: 1}`, which returns the SVM to the formulation you have seen in class. 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	
$W_n = 1, W_p = 1$	Sensitivity	
$W_n = 1, W_p = 1$	Specificity	
$W_n = 1, W_p = 1$	Precision	
$W_n = 1, W_p = 1$	AUROC	
$W_n = 1, W_p = 1$	F1-Score	

Solution:

Class Weights	Performance Measures	Performance
$W_n = 1, W_p = 1$	Accuracy	0.3840
$W_n = 1, W_p = 1$	Sensitivity	0.2300
$W_n = 1, W_p = 1$	Specificity	1.0000
$W_n = 1, W_p = 1$	Precision	1.0000
$W_n = 1, W_p = 1$	AUROC	0.9114
$W_n = 1, W_p = 1$	F1-Score	0.3740

(b) How has training on an imbalanced data set affected performance?

Solution: Training on an imbalanced data set has affected sensitivity, which increased significantly, 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 nearly all samples as positive, as shown by the sensitivity and specificity metrics. The sensitivity (true positive rate) is 1.00 which indicates all positive samples were classified correctly, while the specificity (true negative rate) is 0.05 which indicates few negative samples were classified as negative.

4.3 Choosing appropriate class weights [6 pts]

(a) 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 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, (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 these parameters. This question requires you to choose hyperparameters based on cross-validation; you should not be using the test data to choose hyperparameters.

Solution: Weights found using Cross Validation with respect to **f1-score** across a reasonably large range of weights (Wn=1 to 4, Wp=2 to 9) are: **Wn=5, Wp=8** (CV Score: 0.7827894887089262). In general, Wp should be reasonably greater than Wn.

(b) Use your custom classifier to predict the provided test data `IMB_test_features, IMB_text_labels` 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	
$W_n = ?, W_p = ?$	Sensitivity	
$W_n = ?, W_p = ?$	Specificity	
$W_n = ?, W_p = ?$	Precision	
$W_n = ?, W_p = ?$	AUROC	
$W_n = ?, W_p = ?$	F1-Score	

Solution:

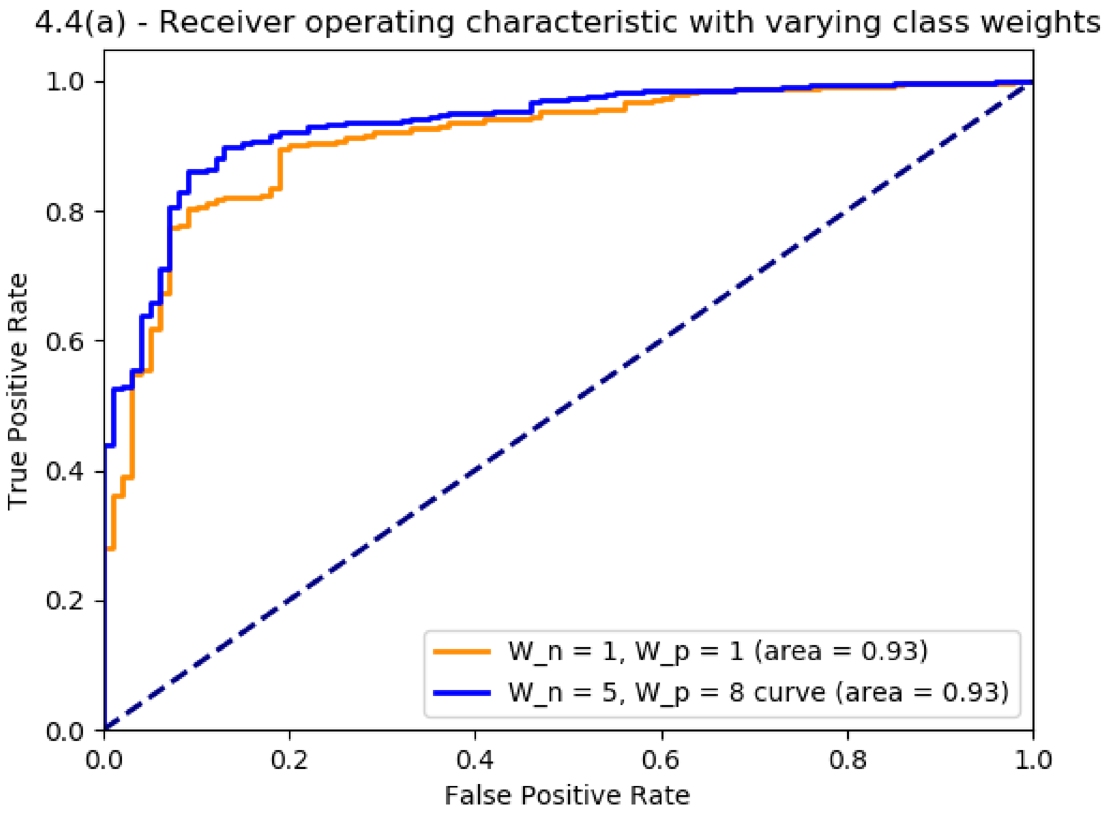
Class Weights	Performance Measures	Performance
$W_n = 5, W_p = 8$	Accuracy	0.8300
$W_n = 5, W_p = 8$	Sensitivity	0.8075
$W_n = 5, W_p = 8$	Specificity	0.9200
$W_n = 5, W_p = 8$	Precision	0.9758
$W_n = 5, W_p = 8$	AUROC	0.9338
$W_n = 5, W_p = 8$	F1-Score	0.8837

Answers may vary based on Wn and Wp weights determined by student in 4.3(a).

4.4 The ROC curve [3 pts]

(a) 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:



5 Challenge [20 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 reviews. We have already prepared a held-out test set `heldout_features` for this challenge, and multiclass training data `multiclass_features`, `multiclass_labels`. This training data has 3000 reviews, 1000 of each class. **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 are additional information that could be leveraged, such as the timezone of the tweet). Your goal is to train a multiclass classifier using `SVC` and `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 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) Scaling or normalizing the data
   (e) 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 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 two components:

1. Write-Up and Code [10 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 [10 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 with true rating -1 (poor), but are classified as a review with rating 1 (good) 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.

REMEMBER to submit your project report by Tuesday, 11:59pm ET on February 12th, 2019 to Gradescope.

Include your code as an appendix (copy and pasted) in your report. 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 at this link `https://tinyurl.com/eecs445w19p1` providing your `umich.edu` email.

Appendix A: Approximate Run-times for Programming Problems

- **Problem 3.1 c**: around 10 minutes
- **Problem 3.2 a i**: around 15 minutes
- **Problem 3.2 a ii**: around 8-10 minutes
- **Problem 4.3 b**: around 4 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.

Appendix B: Lecture Topics and Concepts

The relevant topics for each section are as follows:

- **Problem 3.1** a, b, e, f, **Problem 3.4** c, d
  - Support Vector Machines; Primal Formulation; Geometric Margin
- **Problem 3.2** a, **Problem 3.3** a, **Problem 4.1** a
  - Dual Formulation; Kernels
- **Problem 3.1** c, d, **Problem 3.2** b, **Problem 3.3** b, **Problem 3.4** a, b, **Problem 4.1** b, c, **Problem 4.2** a, b, **Problem 4.3** a, b, **Problem 4.4** a, b
  - Performance Measures

The date provided for a problem indicates the last lecture in which relevant material is discussed. Some problems can be completed before their listed date.