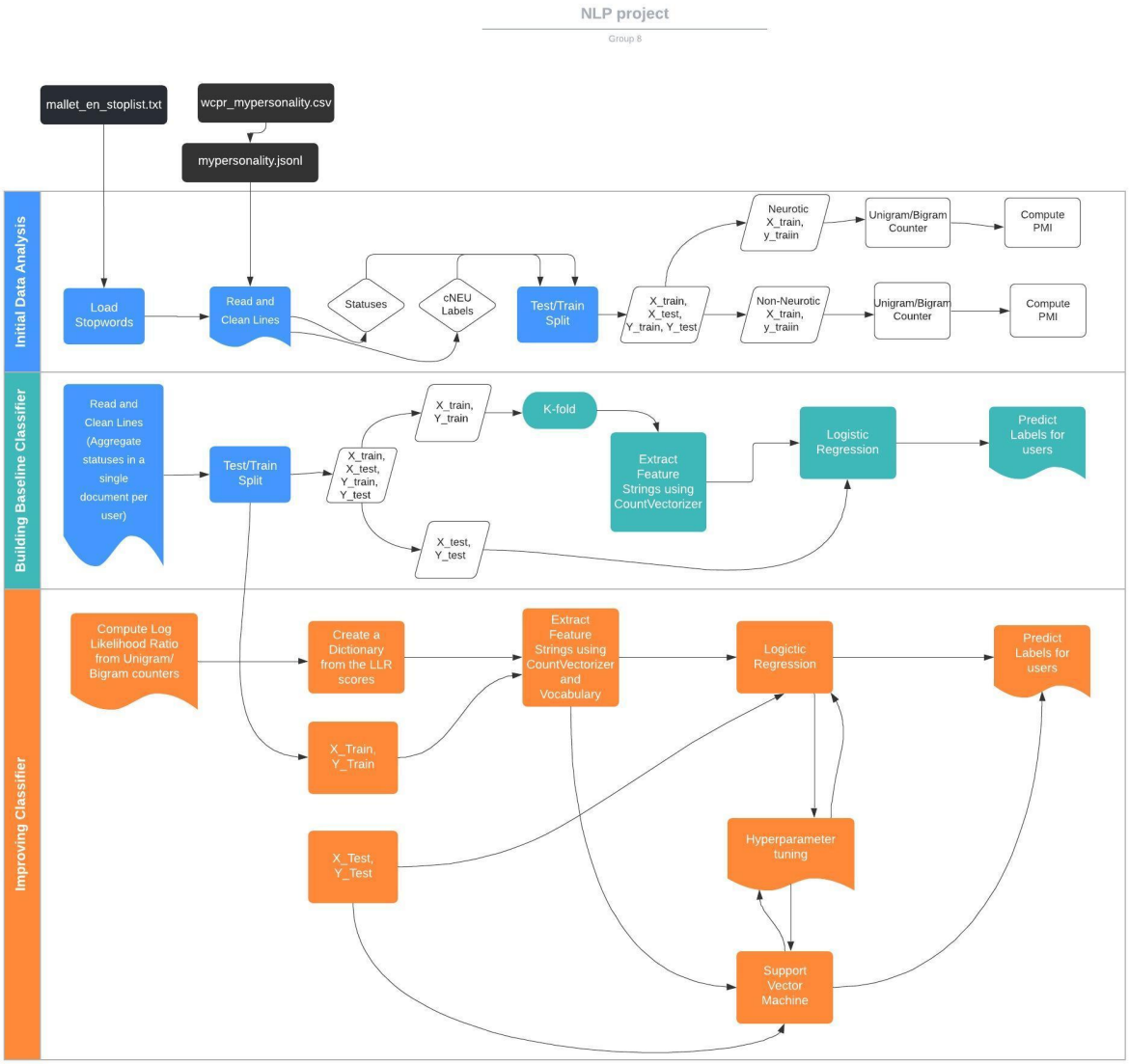
Project Write Up - DATA641

Fred Watts, Shrey Nair, Shrey Patel

Pipeline Diagram



§ Refer to code instructions on pages 14-15 for more information

Introduction

This research project was performed by Shrey Nair, Shrey Patel, and Fred Watts. Our team attempted the first project option: developing a model that uses personal writing samples to predict neuroticism as it is classified in the Big-5 Personality Inventory.

Our team began by studying an earlier iteration of the project from the paper *Workshop on Computational Personality Recognition: Shared Task* (Celli, Pianesi, Stillwell, and Kosinski, 2013) published by the Association for the Advancement of Artificial Intelligence. Researchers compared the results of automatic classification to "gold-standard" personality annotations found with the Big-5 Personality Inventory. They tested their work on two datasets: myPersonality, a set of stream-of-consciousness texts made by students who self-reported their Big-5 scores, and Essays, a set of status updates from Facebook users who took a Big-5 site questionnaire. Eight research teams attempted automatic classification on these datasets. They found personality recognition very challenging despite different techniques and evaluation metrics because predictive features proved sparse.

For our model, we used the same myPersonality structured dataset as Celli, et al. Its corpus was produced by 250 users and contains 9,917 text samples. While this is one of the largest datasets of its type, it seems modest compared to the size of datasets frequently available in other disciplines of data science. This may help to explain the difficulty claimed by the paper's research teams.

The most successful team in Celli, et al's study reached over 90% performance on the myPersonality dataset and attributed their success to "a very large feature space". They collected these features by taking advantage of special lexical tools as well as social and demographic information. Our team used this as a lesson to temper our performance expectations as we don't have access to the same variety of features. Most teams in the paper reached 50% to 70% performance over several evaluation metrics, which seems a more feasible goal.

We also thought it was interesting that these text samples were timestamped and collected over several years. It would be an interesting future project to consider how the date impacts neuroticism. Perhaps neuroticism has weekly or seasonal cycles, or perhaps it spikes around certain annual events. It's also possible that the trait remains

steady but something about these times compel neurotic individuals to post on social media.

Initial Data Analysis

We targeted writing samples where the Big-5 Personality recognition test found neuroticism. This was represented by the symbol cNEU. When cNEU's label was "y", the sample was neurotic. Non-neurotic samples had an cNEU of "n". Roughly 38% of samples were neurotic.

Once we had studied the previous research, we began to code our model by running a k-folds cross-validation to see how our model would perform on unseen data. K-fold function consists of two parts. One, where we perform k-folds using the Logistic Regression classifier, and the other, using the SVM classifier. The parameters like 'test_size', 'num_folds', 'stratify', and 'random_seed' are passed by the user at run time[§]. Looking at the training data, we realized that there was a significant difference between the number of neurotics and non-neurotics, and hence, we decided to perform stratified kfold unless there is a --no_stratify flag specified by the user at run time.

Below are the results with stratify=true, num_fold=5 and the test_size=0.3.

```
Doing cross-validation splitting with stratify=True. Showing 10 indexes for items in train/test splits in 5 folds.
Running 5-fold cross-validation on 70.0% of the data, still holding out the rest for final testing.
----------------------------Logistic Regression----------------------------
accuracy scores = [0.45714286 0.65714286 0.57142857 0.62857143 0.62857143], mean = 0.5885714285714285, stdev = 0.0713714056959817
----------------------------Support Vector Classifier----------------------------
accuracy scores = [0.62857143 0.62857143 0.62857143 0.62857143 0.62857143], mean = 0.6285714285714286, stdev = 0.0
----------------------------Logistic Regression With Vocab----------------------------
accuracy scores = [0.51428571 0.62857143 0.57142857 0.68571429 0.6       ], mean = 0.6, stdev = 0.05714285714285716
----------------------------Support Vector Classifier With Vocab----------------------------
accuracy scores = [0.62857143 0.62857143 0.62857143 0.62857143 0.62857143], mean = 0.6285714285714286, stdev = 0.0
```

As we can see there is minimal standard deviation from the mean scores, and hence, we could go ahead with the predefined number of folds and the test-size.

Once our k-folds were done, we loaded a list of stopwords. This was a text file of the Mallet word2vec English stoplist. Then we converted the myPersonality dataset's original csv file to a jsonl format. This had several advantages. The greatest advantage is that csv files with text may feature accidental combinations of commas and quotation marks that Python csv readers will interpret as a delimiter. The myPersonality dataset includes over a hundred of these messy rows where, for example, cNEU becomes a number or sentence fragment. Meanwhile, a converted json lines file produces no such

[§] Refer to code instructions on pages 14-15 for more information

errors. The json file also has a dictionary-style format that is well-suited for data analysis.

We divided the data into training and testing sets using a 70-30 split: 6,941 training samples and 2,976 testing samples. The 70-30 split is a trusted standard, and at this stage we see no reason to explore other split varieties. This was performed with the sci-kitlearn library which we used for many of this project's machine learning tasks.

```
Read 9917 documents
Read 9917 labels
Training set label counts: Counter({'n': 4333, 'y': 2608})
Test set     label counts: Counter({'n': 1867, 'y': 1109})
```

Training and Test Sets

The next task was to count bigrams and compute PMI (Pointwise Mutual Information, a measure of correlation between two events, such as words in a bigram). We used the spaCy library to load and tokenize text. We set nlp tokenizer rules to not split contraction words. It iterated through each line of raw text, morphing them into tokens which combined to form bigrams. Then we removed bigrams where both tokens were stopwords or either token was punctuation.

The punctuation filter was difficult to implement. We suspect that our dataset had more and a greater variety of punctuation tokens than most NLP projects. This is because social media is home to an informal, avant garde style of writing that makes frequent use of punctuation for emphasis, such as multiple exclamation or question marks as well as ellipses with extra periods. Punctuation is also used to create a kind of modern pictograph called emoji. It would be interesting to compare personality traits to modern emphasis or emoji punctuation, but that is beyond the scope of our research so we decided to sidestep the question and remove punctuation as best we could.

After obtaining filtered bigram counts, we split the bigrams to count unigrams in both bigram positions. With these we computed PMI. Then we were able to sort bigrams by their PMI rank to find the top fifty neurotic and top fifty non-neurotic results by both frequency and PMI score. A short sample is shown in the lists toward the end of this section.

To analyze these lists, we first had to clean the results further. We removed bigrams that were formed from common low-context words that happened to be split with a space. For example, "gon_na" topped all lists. We also removed bigrams with an

§ Refer to code instructions on pages 14-15 for more information

ellipsis as one of the tokens, such as “today_…”, “tonight_…”, and “back_…” along with “propname”. These occupied at least a quarter of both frequency lists.

The cleaned lists weren’t very illuminating. Even the most common bigrams were fairly infrequent, and leading topics were often shared by both neurotic and non-neurotic authors, such as “can’t_wait”, “can’t_sleep”, Christmas, birthdays, and Harry Potter. As for topics unique to neurotics: law school, social analysis, carbon monoxide, Nietzsche, and “due tomorrow” were common. Non-neurotics cared about comedy clubs, good days, great days, good nights, great weekends, and “woo hoo”. While there seems to be a tendency for neurotic authors to mention more unpleasant or nerve-wracking topics, this tendency was subjective and tenuous at best.

We’ve also entertained the possibility that, with frequencies so low, a single post might repeat an idiosyncratic bigram often enough to single-handedly make the top frequency list. This might explain how “crap_crap” or “moist_moist” appeared six or eight times respectively. The word “crap” repeated seven times would produce six bigrams, and it’s easy to imagine a despondent author making the top frequency list in a single sentence.

Top Neu bigrams by frequency	Top PMI Scores for Neu bigrams by frequency
merry_christmas 11	harry_potter 14.662333572604055
ha_ha 11	social_analysis 13.92536797843785
cant_wait 8	late_goodbye 13.92536797843785
law_school 8	criminal_minds 13.340405477716692
can't_sleep 7	union_square 13.340405477716692
long_time 7	ko_chang 13.340405477716692
happy_birthday 6	jets_jets 13.340405477716692
crap_crap 6	gunpowder_treason 13.340405477716692
la_la 6	friedrich_nietzsche 13.340405477716692
good_times 5	san_jose 13.340405477716692
ice_cream 5	chosen_whites 13.340405477716692
harry_potter 5	mousse_torte 13.340405477716692
happy_thanksgiving 4	buenos_aires 13.340405477716692
years_ago 4	desk_chair 13.340405477716692
due_tomorrow 4	carbon_monoxide 13.340405477716692

Top 15 neurotic bigrams by frequency and PMI score

§ Refer to code instructions on pages 14-15 for more information

Top Non-Neu bigrams by frequency	Top Non-Neu bigrams by PMI scores
can't_wait 28	woo_hoo 15.445854455852565
birthday_wishes 18	cape_breton 14.860891955131411
merry_christmas 16	isla_vista 14.445854455852565
comedy_club 14	bombay_elvis 14.445854455852565
downtown_comedy 12	harry_potter 14.445854455852565
happy_birthday 12	carving_pumpkins 14.445854455852565
great_day 11	blah_blah 14.18282005001877
good_day 9	gosh_darn 13.860891955131411
til_7:30 8	favourite_scent 13.860891955131411
can't_sleep 8	envelope_mailed 13.860891955131411
great_weekend 8	size_wired 13.860891955131411
good_night 8	wired_hangers 13.860891955131411
3_days 8	cabin_fever 13.860891955131411
work_3:30 8	california_institute 13.860891955131411
moist_moist 8	swine_flu 13.860891955131411

Top 15 non-neurotic bigrams by frequency and PMI score

Building Classifier

With this initial analysis complete, we had a fair sense of our data and could proceed to designing a classifier. For a baseline classifier, we collected a list of unique users with their aggregated statuses as a single document, split the training and testing sets into ‘X’ and ‘y’ divisions, and appended each to its cNEU status. Then we again iterated through the samples and converted them to feature strings of unigrams and bigrams. We excluded unigrams and bigrams that are stopwords or punctuation, facing the same difficulty as we encountered in the initial data analysis. Finally, we conjoined the feature lists of unigram and bigram tokens to turn them into a space-separated string of features for each user.

For example, if our unigrams are ['coffee', 'cup'], and our bigrams is ['coffee_cup', 'white_house'], then the resulting feature string will be:

'coffee cup coffee_cup white_house'.

These train and test feature strings are then fed into sci-kitlearn’s CountVectorizer which produces a sparse matrix of token counts. With this baseline classifier (evaluation metrics shown in below table “Baseline LR(liblinear)”), we created a logistic regression classifier trained on the featurized training data. We used sci-kitlearn’s LogisticRegression tool.

§ Refer to code instructions on pages 14-15 for more information

Improving Classifier

In order to improve our accuracy results and perform more experiments on the dataset, we computed the LLR scores between the Unigram/ Bigram Counter for Neurotic users and the same for Non-neurotic users. We maintained the separate counters while computing PMI as part of the Initial Data Analysis. After tuning the LLR comparison from a range of 100 - 5000 words, we extracted the top 150 unigrams and bigrams for both neurotic and non-neurotic users and added them to a dictionary. The idea behind this was to feed it as vocabulary for the CountVectorizer to implement feature selection based on a limited vocabulary and re-train the Logistic regression model. The results can be observed in the table below [Look for “Baseline LR (liblinear) and w/ improved features using LLR(Y)”] which bumped the overall accuracy and other evaluation metrics. This feature selection implementation was then performed on SVM along with hyperparameter tuning explained later.

We also decided to add Support Vector Machines as an additional classifier. With the help of hyperparameter tuning, we tried to find the best combination of parameters that would improve our models’ efficiency. The process of hyperparameter tuning is the same for both classifiers. First, we define a range of parameters for each classifier.

Following are the parameters we chose for LR and SVM respectively:

```
param_grid_lr = {'solvers':['newton-cg','lbfgs','liblinear'],'penalty': ['l2'],'c_values': [100, 10, 1.0, 0.1, 0.01]}
```

```
param_grid_svm = {'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],'kernel': ['rbf']}
```

Once we chose our possible parameters, we defined a grid that would hold the accuracy scores from each combination of parameters. With the help of GridSearchCV, we fit the grid, along with the stratified k-fold object, into our predefined LR and SVM models which are called estimators in grid search. Using the grid search attributes like best_score_ and best_params_, we retrieve the combination of parameters with the best accuracy score, which we then pass them back to the main function to rerun the baseline models with a better set of parameters. Following were chosen as the best parameters for their corresponding models:

LR | Best: 0.652397 using {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}

SVM | Best: 0.635512 using {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}

§ Refer to code instructions on pages 14-15 for more information

From hyperparameter tuning we realized that the LR model seems to perform better when the optimization algorithm in the backend is 'newton-cg'. We saw about 10% increase in accuracy when we changed the solver from liblinear to newton-cg, as it went from 62% to 68%. Along with the LR model, the improvised SVM model's accuracy bumped up by 14% as it went from 54% to 62%. The following table represents the metrics obtained from the experiments performed:

Models	w/ improved features using LLR	Accuracy Score	Precision for neuroticism	Recall for neuroticism	Precision for non-neuroticism	Recall for non-neuroticism
Baseline LR (liblinear)	N	0.49	0.24	0.4	0.71	0.53
	Y	0.64	0.38	0.68	0.85	0.63
Baseline LR (newton-cg)	N	0.5	0.26	0.43	0.70	0.53
	Y	0.68	0.44	0.75	0.88	0.65
LR 2.0	N	0.56	0.03	1	1	0.55
	Y	0.57	0.06	1	1	0.56
SVM (linear)	N	0.52	0.65	0.48	0.41	0.59
	Y	0.65	0.5	0.65	0.78	0.65
SVM 2.0 (rbf)	N	0.54	N/A	N/A	N/A	N/A
	Y	0.54	N/A	N/A	N/A	N/A

Blue cells = top/standout scores in their column

Green cells = perfect scores

§ Refer to code instructions on pages 14-15 for more information

Conclusions

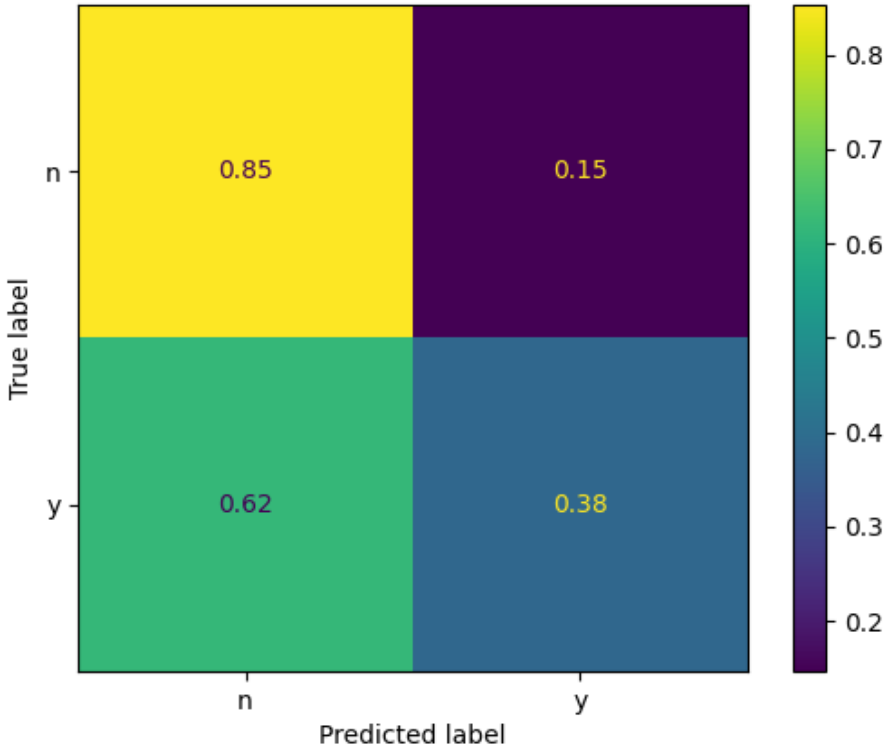
We made the following inferences from the Models table above:

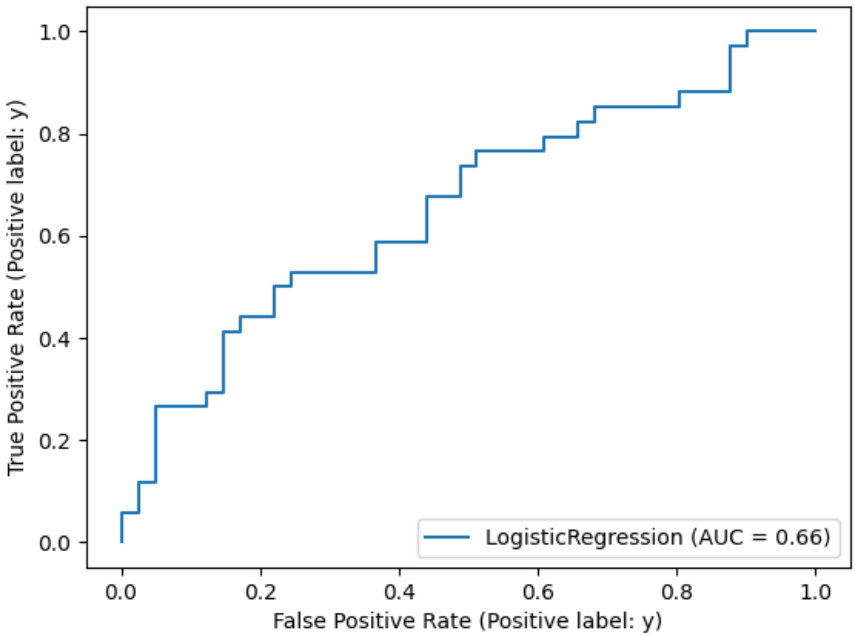
- LR has been the best classifier for our dataset so far.
- It is clear from the above table that computing the LLR could boost the accuracy scores quite significantly.
- In almost all the experiments, hyperparameter tuning helped increase the precision and recall for the non-neurotic users, whereby an improvised LR had a perfect recall for neurotic users and a perfect precision for the non-neurotic users, regardless of the LLR implementation.
- Hence if we were to use LR 2.0 to classify users, we could say that from those that were predicted as neurotic, all of them were neurotic in actuality. On the other hand, from all the actual non-neurotic users, LR 2.0 was able to classify them all as non-neurotic.
- For an experiment where we want to make sure we don't miss out on non-neurotic users, while making sure that all users that are classified as neurotic are indeed neurotic, LR 2.0 is a great model, although for better accuracy, we'd suggest using our Baseline LR model either will liblinear or newton-cg as the optimization algorithm.
- Baseline LR model (newton-cg) has the highest precision for non-neurotic users, while maintaining a relatively higher accuracy score.The same holds true when considering the recall for neurotic users as well.

Below are the confusion matrices and the ROC curves from the experiments that returned top scores:

§ Refer to code instructions on pages 14-15 for more information

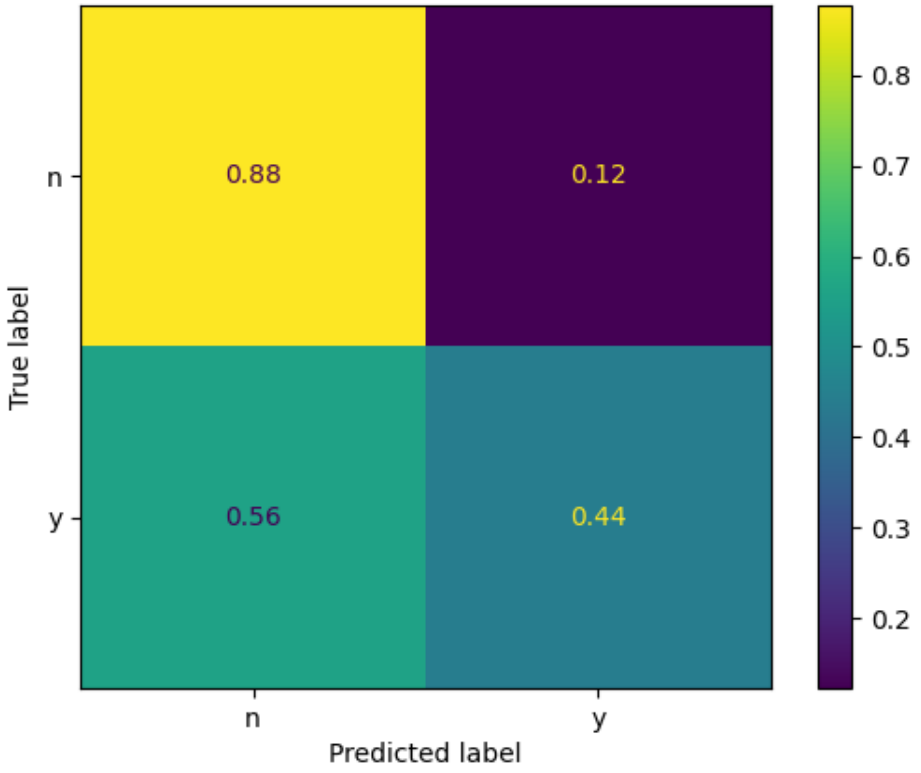
Baseline LR (liblinear) [w/ LLR improved features]

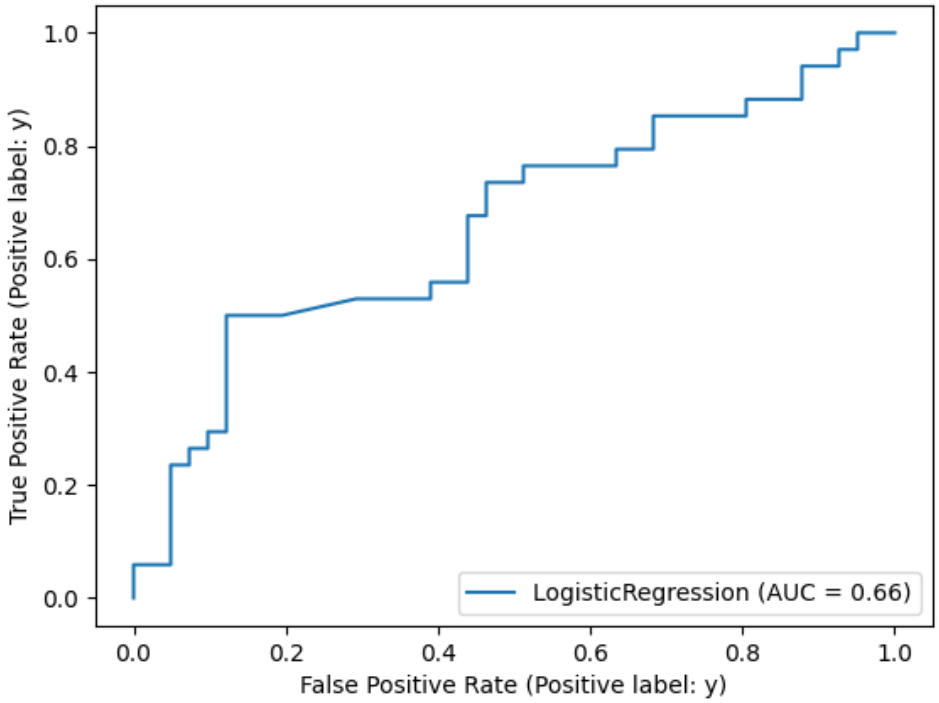




§ Refer to code instructions on pages 14-15 for more information

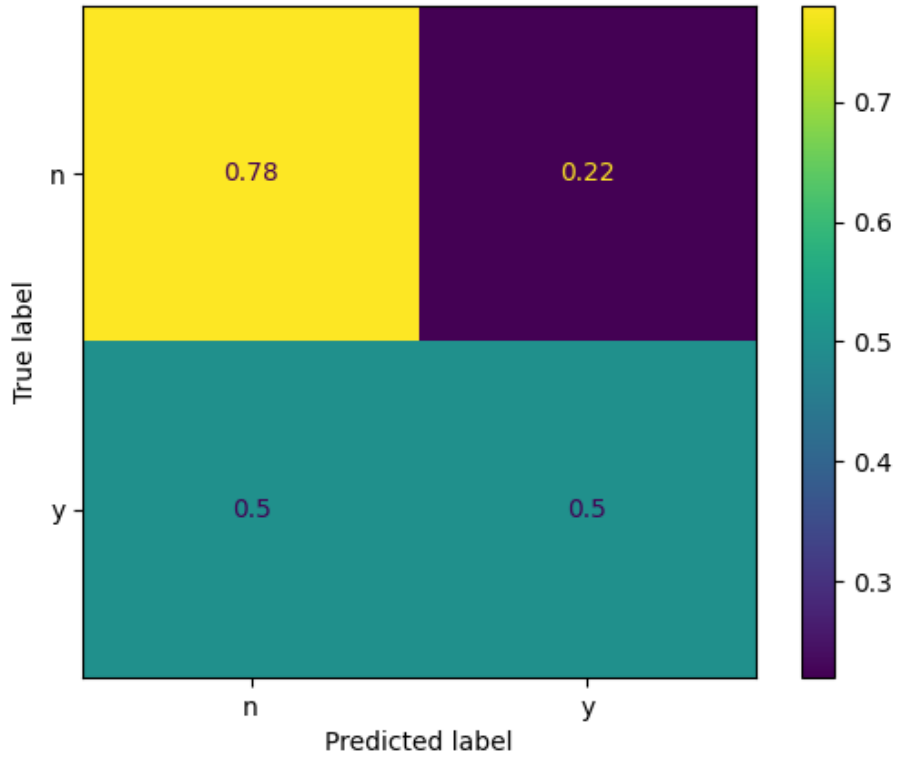
Baseline LR (newton-cg) [w/ LLR improved features]

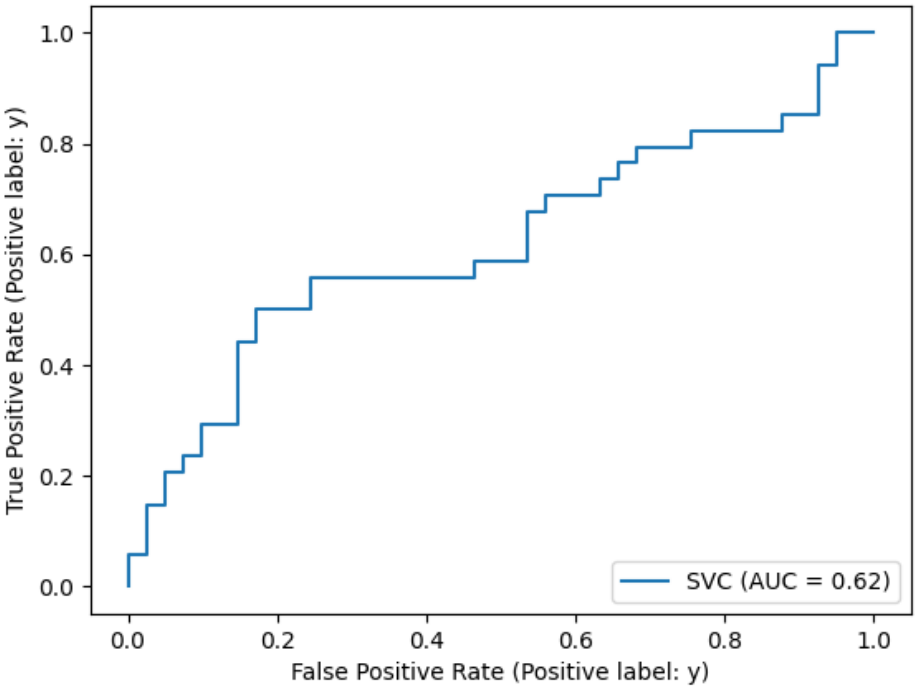




§ Refer to code instructions on pages 14-15 for more information

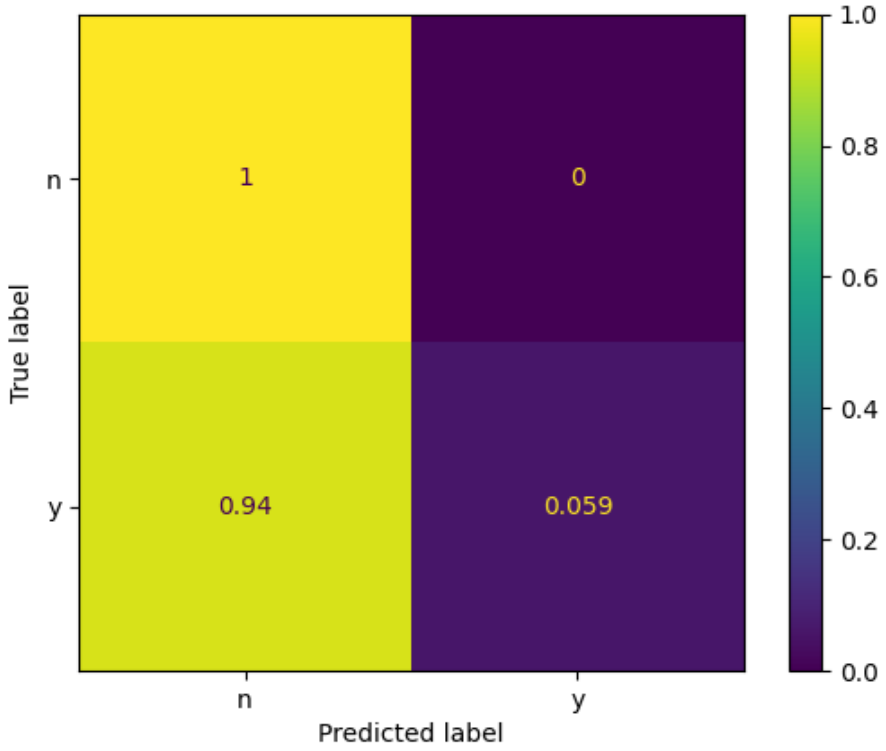
SVM (linear) [w/ LLR improved features]

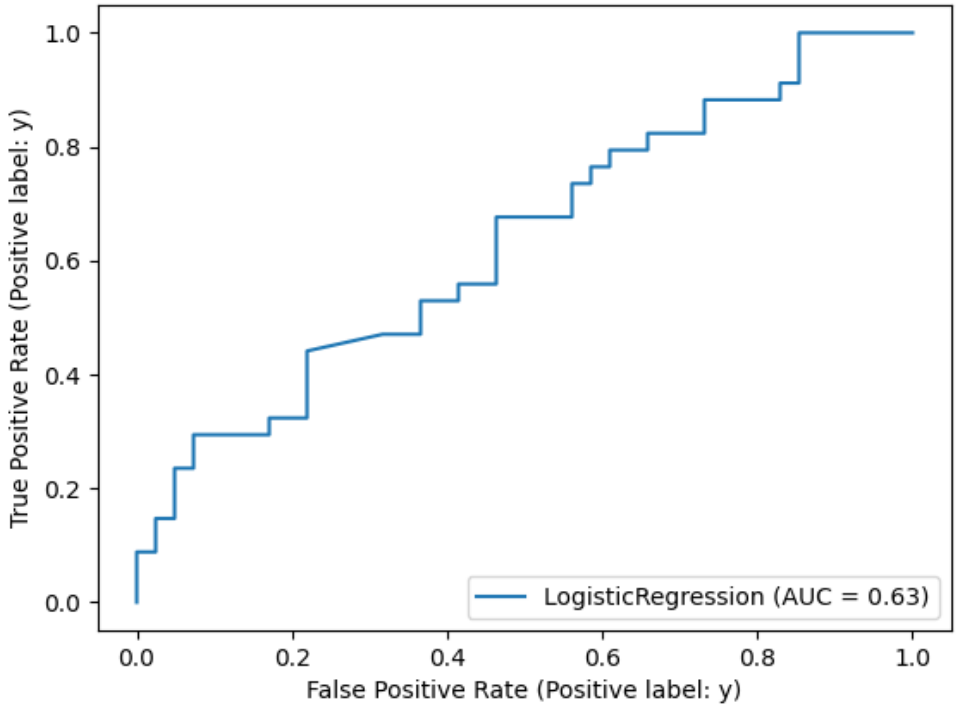




§ Refer to code instructions on pages 14-15 for more information

LR 2.0 [post-hyperparameter tuning] [w/ LLR improved features]



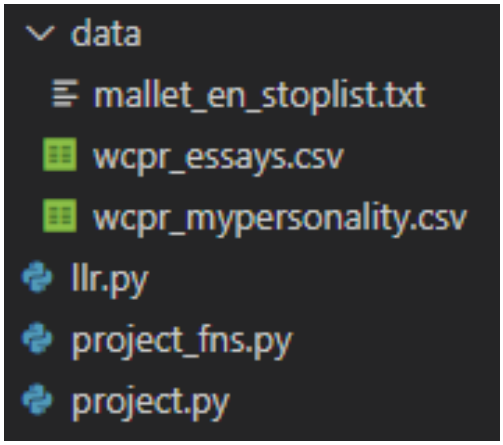


§ Refer to code instructions on pages 14-15 for more information

Instruction On How To Run The Code:

Figure shows the directory structure.

- ❖ llr.py - the script to measure Log Likelihood Ratio
- ❖ project_fns.py - holds functions used for the main project
- ❖ project.py - main file



Project Directory Structure

usage: project.py [-h]

[--use_sklearn_features]

[--test_size TEST_SIZE]

[--num_folds NUM_FOLDS] [--no_stratify] [--seed SEED]

[--plot_metrics]

[--num_most_informative NUM_MOST_INFORMATIVE]

[--infile INFILE]

§ Refer to code instructions on pages 14-15 for more information

Optional Arguments:

-h, --help	show this help message and exit
--use_sklearn_features	Use sklearn's feature extraction
--test_size TEST_SIZE	Proportion (from 0 to 1) of items held out for final testing
--num_folds NUM_FOLDS	Number of folds for cross-validation (use 2 for just a train/test split)
--no_stratify	Use stratified rather than plain cross-validation
--seed SEED	Random seed
--plot_metrics	Generate figures for evaluation
--num_most_informative NUM_MOST_INFORMATIVE	Number of most-informative features to show
--infile INFILE	Input .csv file

§ Refer to code instructions on pages 14-15 for more information