Understanding Public Perceptions of the ACA by Name Used: A Sentiment Analysis Approach Utilizing Open-Source Data

Checkpoint 2

GitHub Repository: Link

Team: Aïcha Camara, Matt Jackson, Rohit Kandala, Summer Long

Overview

The Affordable Care Act was signed into law on March 23, 2010. Since then, it has been colloquially referred to as both the ‘Affordable Care Act’ and ‘Obamacare.’ Despite being the same, a poll from the Morning Consult reports that 35% of respondents did not confidently know if they were the same policy. Naturally, a misunderstanding of these terms having different meanings could naturally result in different sentiments around these terms.

In recent years, Twitter has been more involved with politics. Allegations that Twitter has interfered with election results have been prominent, as Senators, news outlets, and academic publications have written about it. Twitter has also served as a stage for policies to be discussed, including the Affordable Care Act. A natural hypothesis to form is that tweets referring to the policy as the ‘Affordable Care Act’—a name that does not have a person and therefore an inherent political affiliation to it—and ‘Obamacare’—a name that is affiliated with a Democratic Party—will yield two different skews of sentiment. Furthermore, a tweet that refers to it as both in the same tweet could lie in the middle of the sentiments, or have its own skew altogether.

It would be tedious to comb through hundreds of thousands of tweets to manually assign sentiment. Instead, our project plans to leverage Natural Language Processing and Machine Learning to learn a sentiment model from open source Yelp data and apply this model to our sample of Twitter data. In this Checkpoint, we will discuss the models built thus far—reporting on metrics such as F1-Score, Accuracy, and AUC—and our next steps.

Sampling Strategies

The Yelp dataset is large—nearly 500 MB uncleaned. This is too large to load into memory, and would be difficult to train models on. We decided to sample the data and have access to two datasets: a true random sample of 100,000 reviews, and an artificially sampled dataset that has 20,000 reviews per star rating.

Sentiment From Stars Assignment Decision: 1 & 2 Negative; 3, 4, & 5 Positive

The Yelp data is a great dataset to use for this task because it already has human-assigned ‘sentiment’ associated with the text of reviews in the form of star ratings. This is on a 1 to 5 scale system where 1 star is intended to be the most negative, and 5 the most positive. However, in order to approach this task as a binary classification task with positive and negative sentiment, exploratory data analysis was conducted to determine a threshold for positive sentiment from the star ratings.

When conducting this exploratory data analysis, the following factors were taken into consideration:

1. Subjective, human analysis from top words for each star rating
2. VADER sentiment analysis

Full results can be read here, but a short excerpt summarizes this well: if we were looking at things from a merely statistical standpoint using the VADER compound score, it might make sense to set the cutoff between 1-star and 1-star. However, the assessment of common words for the 2-star reviews brings up a lot of intensely negative terms, such as (to look at truly unique words in the 1000 most common) 'flavorless', 'rushed', 'raw', 'stale', 'tasteless', 'uncomfortable', and 'annoyed.' I don't want to discount that impression.

Therefore: our official decision is that, for the remainder of this project, we are classifying Yelp reviews with 3 or more stars as "positive", and all others (1 or 2 stars) as "negative" , with no "neutral" category. This will capture the sharp jump in positivity scores at the 3-star level and the subjective assessment that 2-star reviews have significantly more negative terms even if the naive settings for VADER didn't recognize them as such. It will also enable us to use binary classification methods, which build most directly on the course material covered thus far.

As we proceed, though, we will keep in mind that the Yelp star reviews are a rather messy, lossy measure (which makes sense, as different users don't have to follow the same standards when deciding how to review an establishment), and simplifying them to a binary classification introduces even more mess and loss. Some of our labels may strike us "incorrect" or "misleading" in ways that we can't really adjust going forward. As such, we shouldn't be surprised if it is difficult to obtain high accuracy in classification models.

Pre-processing Steps

Popular models for Natural Language Processing Machine Learning tasks do not use raw words, but instead leverage methodologies to convert them to vectors and numeric values. In order to ensure the ability to consistently compare models, we created a standardized process for pre-processing our data.

Leveraging scikit-learn and NLTK, we performed the following steps:

1. Tokenizing the data: The input text is tokenized into a list of words using the NLTK package's 'word_tokenize' function. The text is also converted to lowercase.
2. Stopword removal: ‘Stop words’ refer to common words in texts that do not necessarily contribute to the overall meaning of a sentence, such as ‘in’, ‘of’, the’, ‘a’, etc. This is done with NLTK’s ‘stopwords’ function of English words.
3. Re-joining the text: This step essentially ‘reassembled’ the reviews together again.
4. Sentiment column creation: A value of 0 was mapped onto a column named ‘sentiment’ if the star rating was 1 or 2, and 1 for star ratings 3, 4, or 5.
5. TF-IDF Vectorizer: Stands for "Term Frequency-Inverse Document Frequency". It reflects the importance of a word in a document, relative to its importance relative to all

reviews. The output is a matrix in each row. One key thing to note is that this vectorizer was fit on the training data and applied to the testing data afterwards to avoid data leakage (discussed here).

A Preview of Models

Preliminarily, we decided to investigate the following models:

1. XGBoost
2. Random Forest
3. Support Vector Machines
4. Logistic Regression

These models are trained on the randomly sampled data.

A Note on Grid Search for Hyperparameter Selection

Working with a sample of data gives us flexibility to perform more computationally expensive tasks in order to discover the best model. The strategy we chose to deploy was grid search. Grid search is a method to search through a defined set of hyperparameters by making combinations of models until the best one is discovered. Each model will have its own unique hyperparameters to search over. The default metric used to determine the best model is accuracy. While we did search over the default setting, we are considering multiple metrics that are reported to determine our best model between the ones trained. Our grid search also included performing 5-fold cross validation for each model.

A Note on Evaluation Metrics

The dataset is imbalanced. In the dataset used, the ‘positive’ sentiment class is approximately 77%, while the ‘negative’ sentiment class is only 23%. Accuracy alone may be deceiving in determining the ‘best’ model. For example, a model that ends up predicting the majority class 100% of the time will have an accuracy of 77% due to the imbalance. Therefore, alternative metrics will be considered in addition to the accuracy when evaluating the ‘goodness’ of a model. Such metrics include F1-score, Precision, Recall, and AUC-ROC score.

XGBoost

XGBoost, short for Extreme Gradient Boosting, is a machine learning algorithm that builds an ensemble of decision trees with gradient boosting. While there are ten total hyperparameters to fit, three were searched for—decided with a balance of importance and computational power: the number of decision trees, the learning rate, and the maximum depth of each tree. The number of decision trees in the ensemble decides the capacity of the model to capture complexity, needing to be balanced with the risk of overfitting. The learning rate controls how much the weights of the models are updated at each iteration. The maximum depth controls how many nodes a particular decision tree can have.

XGBoost is tolerant of class imbalance–that is to say, class imbalance is likely to not negatively impact the training of an XGBoost model. So, no over or undersampling was conducted prior to running the model. Below details the values within the grid search.

Values Searched Over for Grid Search of XGBoost

Hyperparameter	Number of trees	Learning rate	Maximum depth
Values	[50, 100, 200]	[0.01, 0.1, 0.5]	[3, 5, 7]

The grid search took approximately 185 minutes to run, as 27 unique models were trained. The best hyperparameters selected were 200 trees, a learning rate of 0.5, and a maximum depth of 7.

Best XGBoost Model from Grid Search Reports

	Precision	**Recall**	**F1-Score**
0	.85	.77	.81
1	.93	.96	.94

Accuracy	**AUC-ROC**
0.91395	0.96087

While this model does perform better on the majority class than the minority class (negative sentiment), it still has strong accuracy metrics (91% > 77%) and AUC-ROC metrics (96%). With an F1-Score of .81, it has satisfactory performance on the minority class (negative sentiment).

Random Forest

Random forest is an ensemble method that creates a model by combining multiple decision trees. The hyperparameters to fit are the number of decision trees, the maximum depth of each decision tree, the maximum number of features considered for splitting at each node, the minimum samples required to split a node in a decision tree, and the minimum number of samples in a node in a decision tree. The number of decision trees can improve the accuracy of the model but must be weighed with the risk of overfitting. The maximum depth of each decision tree limits how many splits can be made in each tree. The maximum number of features considered for splitting at each node can reduce overfitting and improve generalizability as it limits the number of features. The minimum number of samples required to split a node dictates the point in which a decision tree will stop growing. The minimum number of samples in a leaf dictates how many nodes will be made—as nodes with fewer samples than the minimum will be grouped together in a different node**.**

In theory, random forest is tolerant to class imbalance. While no over or under-sampling was conducted prior to training the model the first time, poor results from the first run prompted over-sampling of the minority class to improve model performance. Below details the values within the grid search.

Values Searched Over for Grid Search of Random Forest

Hyperparameter	Number of trees	Maximum depth of tree	Maximum number of features	Minimum samples required to split a node	Minimum samples in a node
Values	[10, 50, 100]	[3, 5, 7]	[‘sqrt’, ‘log2’]	[2, 5, 10]	[1, 2, 4]

Version 1: No Oversampling

For this, the grid search took approximately 40 minutes to run, as 108 unique models were trained. The best hyperparameters selected were 10 trees, a maximum depth of 7, maximum features as the square root of total features, a minimum of 5 samples required to split a node, and a minimum of 2 samples in a node.

Best Random Forest Model (True Sampled Data) from Grid Search Reports

	Precision	**Recall**	**F1-Score**
0	.96	.01	.02
1	.77	1.00	.87

Accuracy	AUC-ROC
0.7689	0.80943

Without oversampling the minority class, this model has poor performance overall. The Recall of the negative sentiment class is .01 and the positive class is 1.0, meaning that the model is predicting almost all samples as positive. This is further backed up by the fact that the Precision of the positive class is the same as its proportion in the data (77%), and the accuracy is almost the same as the proportion of the positive class. This model is practically the same as predicting the majority class 100% of the time.

Version 2: Oversampling via SMOTE

Seeing poor performance from the data, oversampling of the minority class was done with SMOTE (Synthetic Minority Oversampling Technique). It creates synthetic examples of the minority class by interpolating between existing minority class samples. The same grid was searched over, and the same best hyperparameters were selected as Version 1.

Best Random Forest Model (Oversampled Data) from Grid Search Reports

	Precision	Recall	F1-Score
0	.55	.50	.53
1	.85	.88	.86

Accuracy	AUC-ROC
0.78855	0.78533

While oversampling did improve the model's predictive ability on the minority class substantially—as the recall is now .50—the accuracy and AUC-ROC saw marginal changes. It appears Random Forest may not be the best model for this classification problem.

Logistic Regression

A Logistic Regression is a type of statistical analysis that uses a probabilistic model to classify into the proper outputs. It's different from a perceptron because it is not a linear classifier; rather, it uses probability to similarly "classify" inputs into their respective outputs. Four hyperparameters were explored: magnitude of penalty, solving algorithm, regularization parameter, and class weighting. The magnitude of penalty, *C,* determines how much the model is penalized for incorrect predictions. The solving algorithm determines how the optimization is done. The regularization parameter determines how the coefficients of features are shrunk by adding a penalty to the cost function. The class weight hyperparameter assigns different weights to different classes during training in the cost function, and can help achieve better performance on the minority class by penalizing mistakes in misclassification for that class stronger.

By tuning these hyperparameters, Logistic Regression can be a model tolerant to class-imbalance. As a result, oversampling was not conducted during training.

Values Searched Over for Grid Search of Logistic Regression

Hyperparameter	Magnitude of penalty (C)	Solving algorithm	Regularization parameter	Class weights
Values	[0.1, 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 5.0, 10.0]	['lbfgs', 'liblinear']	['L1', 'L2', 'None']	['None', 'Balanced']

This grid search trained 108 models and took approximately 2 minutes. It should be noted that some combinations failed, because the ‘lbfgs’ solver is incompatible with L1 regularization. The best hyperparameters selected were a magnitude of penalty of 3.0, the ‘lbfgs’ solving algorithm, ‘L2’ regularization parameter, and no class weighting.

Best Logistic Regression Model from Grid Search Reports

	Precision	**Recall**	**F1-Score**
0	.87	.81	.84
1	.94	.96	.95

Accuracy	**AUC-ROC**
0.92755	0.88618

The Logistic Regression has strong performance on both classes. While this model does perform better on the majority class than the minority class (negative sentiment), it still has strong accuracy metrics (93% > 77%) and AUC-ROC metrics (89%). With an F1-Score of .84, it has satisfactory performance on the minority class (negative sentiment).

Support Vector Machines (SVM)

Like a perceptron, the method of support vector machines (SVM) attempts to create a separating hyperplane. More specifically, SVM attempts to create a separating hyperplane that maximizes the margin (distance from hyperplane to closest point(s) on either side). A ‘support vector’ is a vector from a point in the data to the hyperplane, whose length is exactly equal to the desired margin; ideally, an SVM model will result in a hyperplane that has a large number of support vectors. However, if the data is not linearly separable in the original n-dimensional space, SVM ensures maximal separation by making use of the "kernel trick." This uses a kernel function to transform the data into a representation in higher-dimensional feature space, so as to find a boundary that lineary separates the data *in that higher-dimensional space* (even if that boundary is a curve in the original n-dimensional space).

Four hyperparameters were explored: penalty for misclassification, the kernel function, the shape of the kernel function, and the maximum iterations permitted before stopping. The penalty for misclassification, *C*, can have an effect on the generalizability of the model and control the trade-off between bias and variance. The type of kernel function dictates the mapping of points onto a plane so separating a hyperplane can be found. The gamma parameter dictates the shape of the kernel function as a coefficient. The maximum iterations can optimize efficiency of the algorithm running, as it prevents models that fail to converge from running too long.

SVM can handle class imbalance, as in many cases it will yield similar results without weight correction. Therefore, no oversampling is conducted prior to training the model.

Values Searched Over for Grid Search of SVM

Hyperparameter	Magnitude of penalty (C)	Kernel function	Shape of kernel function (gamma)	Maximum iterations
Values	[0.01, 0.1, 1.0]	['linear', 'poly', 'rbf', 'sigmoid'*]	['scale', 'auto']	[100, 1000, 10000]

*sigmoid is named 'sigmoid', but it is actually tanh as shown in class

The grid search trained 72 models and took roughly 5 hours and 17 minutes. The best hyperparameters chosen were a magnitude of penalty of 1.0, kernel function of 'rbf', shape of kernel function as ‘scale’, and maximum iterations as ‘10000’. This was by far the most computationally expensive model to train.

We noticed that runtime got really long as max_iter went up, and the models were still not converging with maximum iterations as high as 10000. It turns out that SVC runtime scales at something like O(n^2) or even O(n^3) of length of training data, and the standard recommendation is to avoid a nonlinear kernel for data of size n > 10000. And our training data (80000 Yelp reviews, compressed into observations with 65910 features each) is bigger than that. With the number of iterations to convergence (and the amount of time it’d take to converge) unknown, and the supralinear big-O suggesting it’d possibly be too long to complete before the due date, we decided to settle on 10000 as the highest maximum number of iterations.

Best SVM Model from Grid Search Reports

	Precision	**Recall**	**F1-Score**
0	.86	.81	.83
1	.94	.96	.95

Accuracy	**AUC-ROC**
0.9208	0.96806

Similar to XGBoost, this model has extremely strong performance. While this model does perform better on the majority class than the minority class (negative sentiment), it still has strong accuracy metrics (92% > 77%) and AUC-ROC metrics (97%). With an F1-Score of .83, it has satisfactory performance on the minority class (negative sentiment).

Preliminary Comparisons Between Models

Out of the models trained, Random Forest performed the worst by far, even when oversampling the minority class. So when considering the best model to use, Random Forest has been ruled out of consideration.

Comparison of Metrics Between Models

Model	F1 Score of majority class	F1 Score of minority class	Accuracy	AUC-ROC	Time to train best model
SVM	**.95**	.83	.9208	**.96806**	~51 minutes
Logistic Regression	**.95**	**.84**	**.92755**	.88618	**< 2 minutes**
XGBoost	.94	.81	0.91395	.96087	~7 minutes

Bolded numbers mean that out of the 3 models, the model had the best result out of the 3 models for that particular metric. Some key things stand out:

- While the logistic regression had the best accuracy and F1 scores, it had a significantly weaker AUC-ROC metric. This is not an issue we can ignore, as the logistic regression had lower AUC-ROC score than both XGBoost and SVM by .06, and AUC-ROC is the preferred metric on imbalanced data.
- XGBoost is marginally worse than SVM in every metric, but took about a seventh of the time to train.
- SVM had the strongest performance for the majority class, the second strongest performance on the minority class, and the strongest performance via AUC-ROC score,

but took by far the longest to train. It took at least 51 times as long to train as logistic regression, and at least 7 times as long to train as XGBoost.

Next Steps and Feedback Request

Before making a call on how to proceed with our model results, we will first randomly assign each group member 50 tweets to manually classify as 'positive' or 'negative' (200 in total). This is so that we have a testing set when we are applying the models we made, which were saved to .pkl files to avoid spending time training again. When we are applying the models to the Twitter data, we are going to fit the same tf-idf vectorizer that we made for the Yelp data onto the Twitter data, and clean the data to remove Twitter specific terms (@'s, RTs, etc.)

Once that is done, we are looking for feedback about how to proceed. Should we…

1. Try all 3 models on the Twitter 'test set' we made?
2. Select the 'best model' from the 3 we made, and then only try that one on the Twitter data?

We have not formally made a decision as to which is the 'best model' for the Yelp data, but we are leaning towards SVM due to its similar performance on the minority class as Logistic Regression and strongest AUC-ROC score. We did not weigh time heavily in making a decision as we already saved the model to a .pkl file.

One other thing to note is that the grid search selection parameter was not specified, so the default was accuracy. However, we used AUC-ROC heavily to make a decision. We did not realize this until we had finished training all models. Keeping time and correctness both in mind, should we re-run the grid searches for each model and specify the scoring metric as AUC-ROC?

There are some other things we have in mind to potentially push the performance of models even higher. For example, in the XGBoost model, the best hyperparameters selected were the highest out of all the values defined for each. So, a new grid search could be conducted with a redefined hyperparameter grid to start at the best hyperparameters selected and continue higher. For SVM, the number of iterations could be pushed higher—but this is not necessarily as feasible due to the amount of time training took for one model alone. We may also consider running the logistic regression, XGBoost, and SVM with balanced classes to see if the metrics are further increased.

Overall, deploying one of the two strategies above, we are hoping to apply the model(s) onto the Twitter data and synthesize a report of sentiment based on factors such as seasonality, terms used, and engagement (retweets). This report would primarily focus on the difference in sentiments (if observed) between 'Obamacare' and the 'Affordable Care Act' terminology, but would also highlight interesting patterns of seasonality, terms used, and engagement as well if broadly observed.