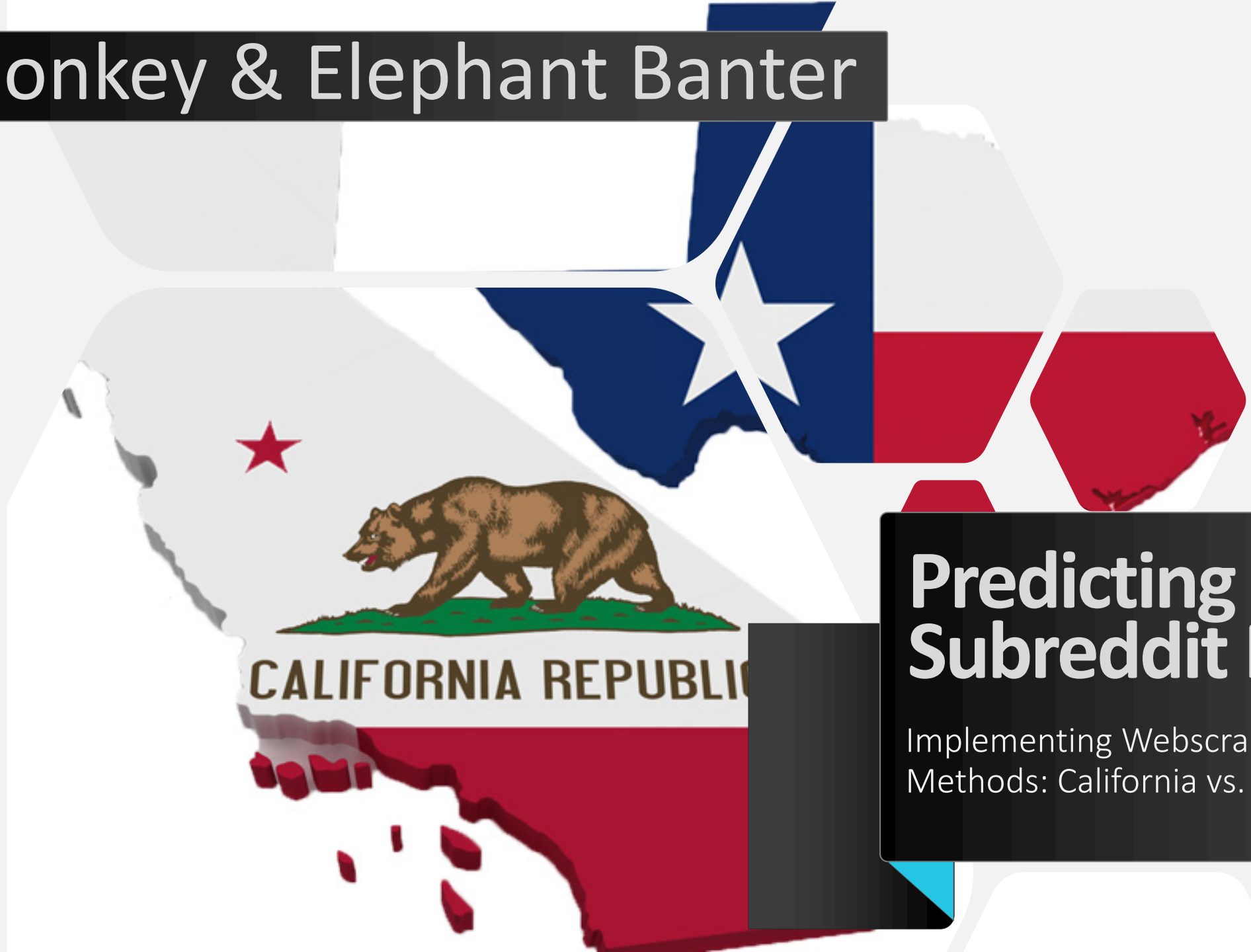
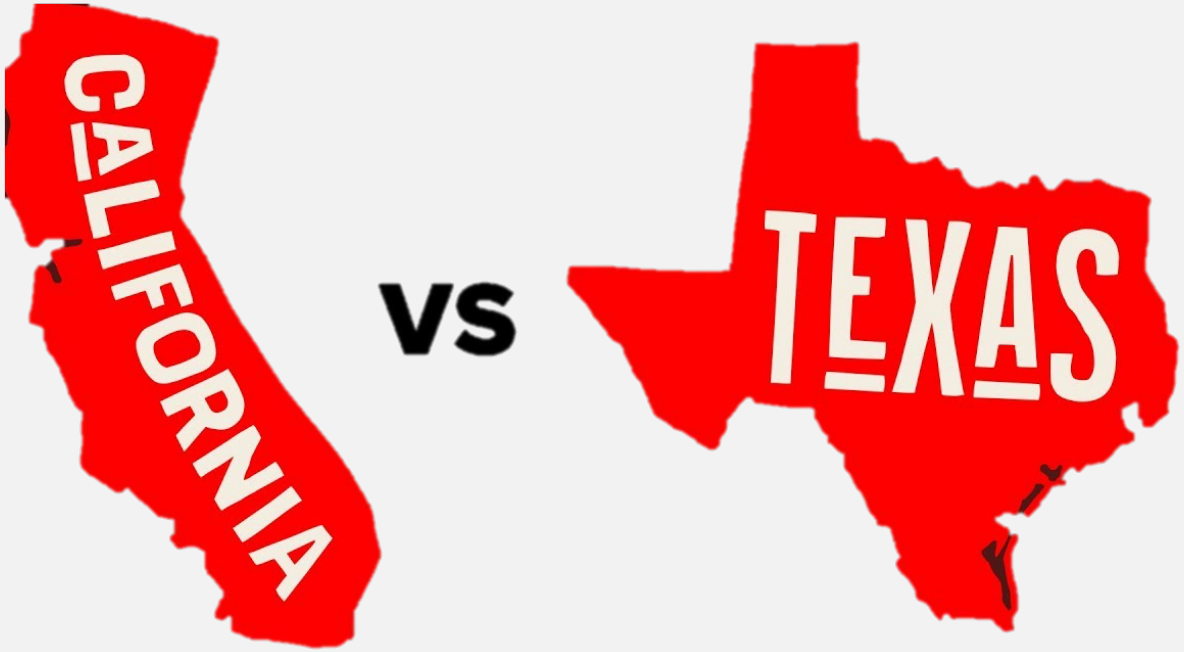
Donkey & Elephant Banter



Predicting Subreddit Posts

Implementing Webscraping & NLP Methods: California vs. Texas Politics

The Process:



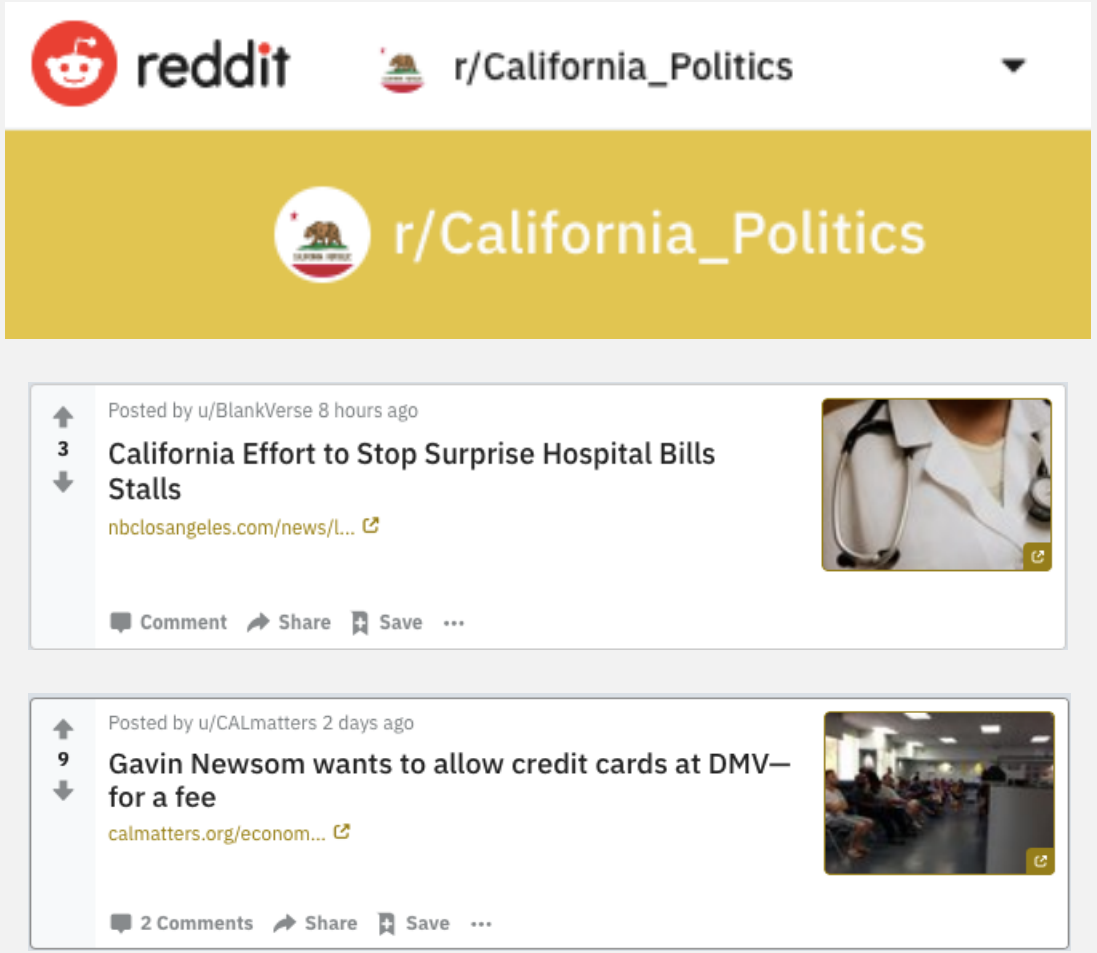
- Data gathering
- Data cleaning
- Visualizing
- Modeling
- Analysis
- Conclusions

Data Gathering

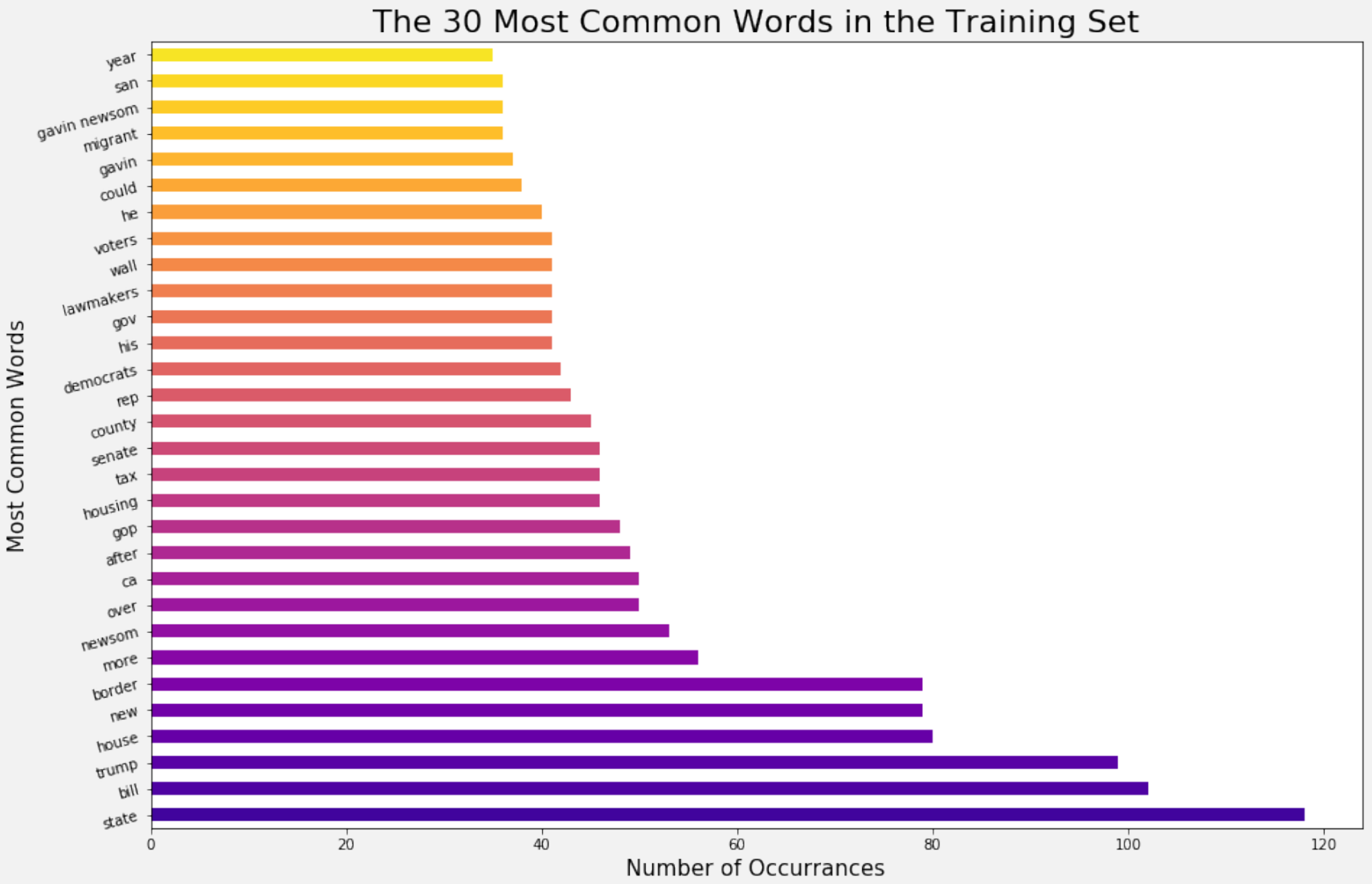
- Accessed the Reddit API
- Scrapped Data From 2 Subreddits
  - California Politics
  - Texas Politics
- Scrapped the 1000 most recent posts
  - Rendered 938 unique CA Posts
  - Rendered 982 unique TX Posts

Data Cleaning

- Compared the body of each post to the title
- Determined that the title was going to be more consistent
- Put all information into a DataFrame
- Kept Post Title and Subreddit Indicator
- Used California as my dependent variable
- Used a CountVectorizer to vectorize my data
- Split Data into Training and Testing set

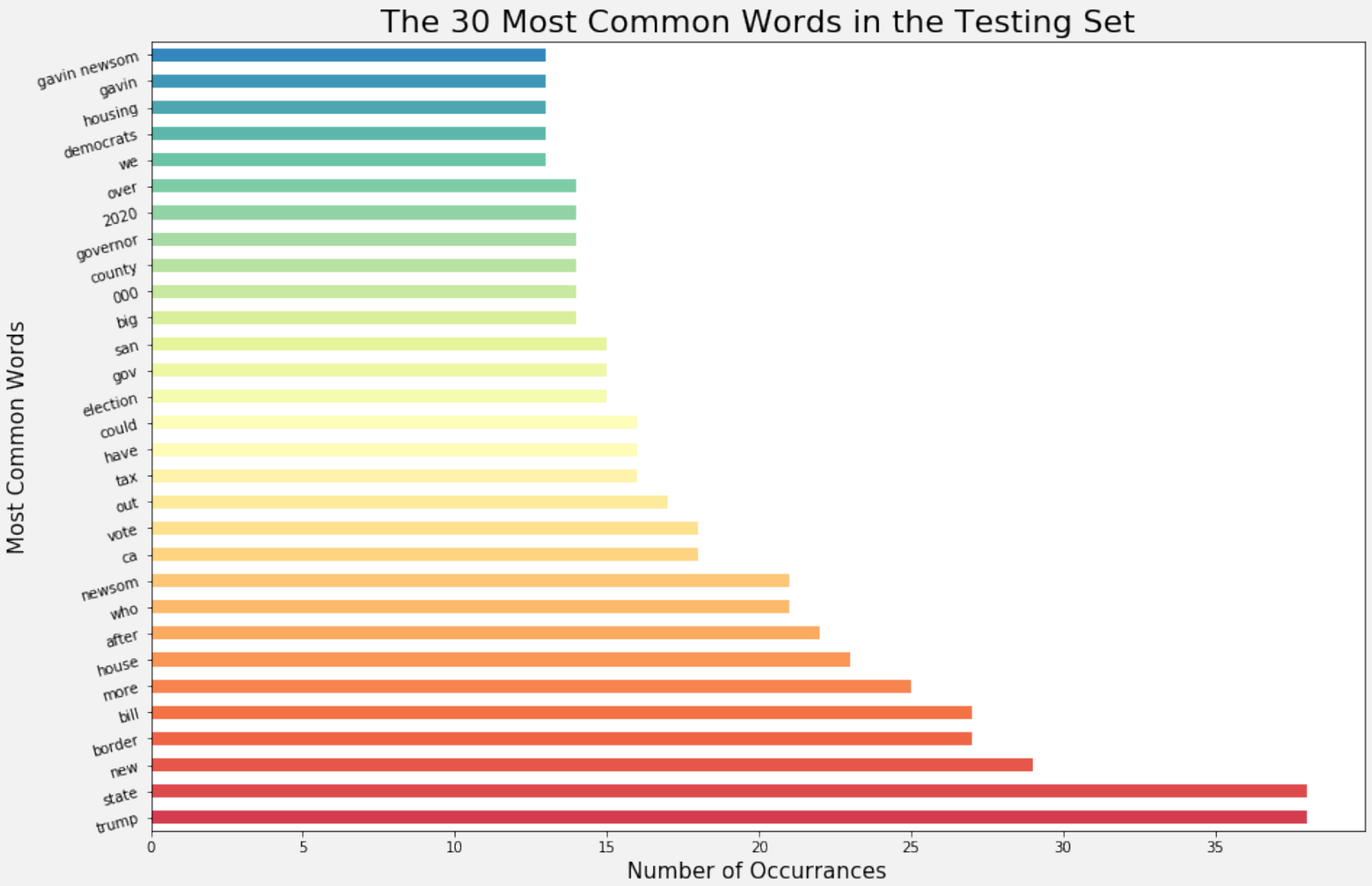


Visualizing the Data: Training Set



- Examined what the most common words were from my training corpus
- Removed any words that did not appear to be predictive
- "California" and "Texas" were perhaps too predictive
- Run model with and without and compare

Visualizing the Data: Testing Set

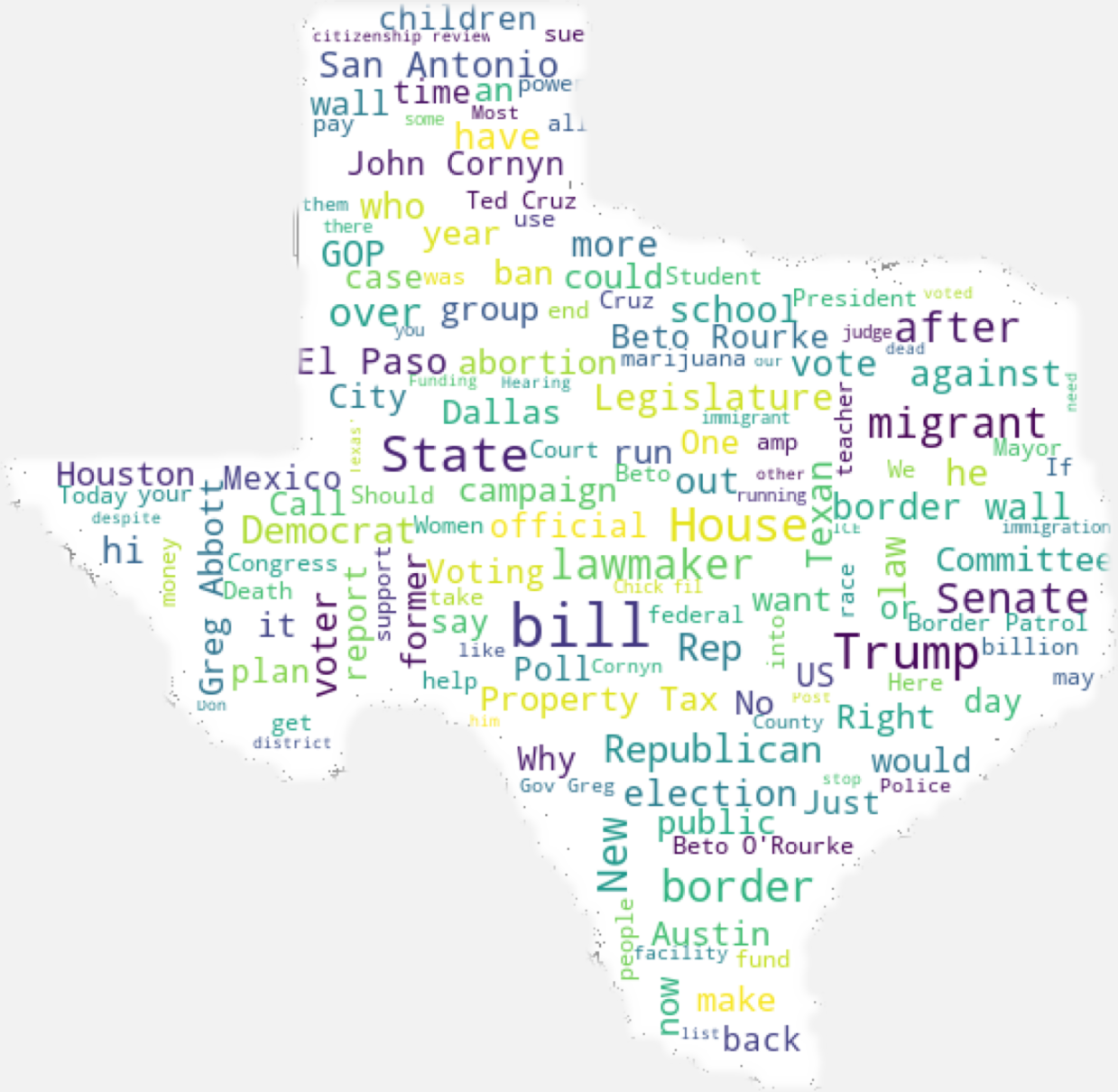


- Similarly on the testing corpus
- "Texas" and "California" play a huge factor again
- So we scored the corpus with and without them as stopwords.

Visualizing the Data: Comparing Subreddits: California Subreddit

- While not scientific, its clear some key words are present:
  - California
  - Trump
  - State
  - Gavin Newsom
  - Proposition
  - Housing
- These words are driving the California Subreddit

Visualizing the Data: Comparing Subreddits: Texas Subreddit



- Contrasting with the California Subreddit:
  - Bill
  - Migrant
  - Republican
  - Border
  - John Cornyn
  - Beto

- Words that they share with California
  - State
  - Trump
  - House

Modeling the Data

What are we testing: **If a post is from the California Politics Subreddit**
Baseline Model Accuracy: 48.85%

- Set up a Pipline:
  - Vectorizers :: CountVectorizer & TfidfVectorizer
  - Classification Models :: Logistic Regresstion, Multinomial Naive-Bayes Classifier, Random Forest Classifier, DecisionTree Classifier
- Gridsearch through all of the different hyperparameters
- Get a cross fold validation score
- Score the training data
- Score the testing data
- Repeat with tuned parameters

Modeling the Data: Set Up

```
# Deciding which words to remove via stop words
stop_words = ['to', 'the', 'in', 'of', 'for', 'and', 'on', 'is', 'it',
              'with', 'what', 'about', 'are', 'as', 'from', 'at', 'will',
              'that', 'says', 'by', 'be', 'this', 'can', 'has', 'how',
#               'california', 'texas'
             ]
# Setting up our hyperparameters to pass through our pipeline
pipe_params = {
    'vec' : [CountVectorizer(), TfidfVectorizer()],
    'vec__max_features': [1500, 1600, 1700],
    'vec__min_df': [2, 3],
    'vec__max_df': [0.4, 0.5],
    'vec__ngram_range': [(1,2), (1,1)],
    'model' : [LogisticRegression(),
               LogisticRegression(penalty='l1', solver='liblinear'),
               LogisticRegression(penalty='l2', solver='liblinear'),
               MultinomialNB(alpha=1.1),
               RandomForestClassifier(n_estimators=500),
               DecisionTreeClassifier()],
    'vec__stop_words': [frozenset(stop_words)]
}

# Defining a function to do our model analysis. This function takes in X, y, and any pipe parameters
def model_analysis(X, y, **pipe_params):
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
    pipe = Pipeline([
            ('vec', CountVectorizer()),
            ('model', LogisticRegression())])

    gs = GridSearchCV(pipe, param_grid=pipe_params, cv=3, verbose=1, n_jobs=2)
    gs.fit(X_train, y_train)

    print(f' Best Parameters: {gs.best_params_}')
    print('')
    print(f' Cross Validation Accuracy Score: {gs.best_score_}')
    print(f' Training Data Accuracy Score: {gs.score(X_train, y_train)}')
    print(f' Testing Data Accuracy Score: {gs.score(X_test, y_test)}')
```

Modeling the Data: Results

```
Best Parameters: {'model': LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False), 'vec': TfidfVectorizer(analyzer='word', binary=False, decode_erro
r='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=0.4, max_features=1500, min_df=2,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=frozenset({'of', 'be', 'this', 'are', 'has', 'is', 'at', 'the', 'will', 'that', 'says', 'what', 'a
nd', 'it', 'to', 'can', 'by', 'about', 'in', 'from', 'as', 'with', 'on', 'how', 'for'}),
        strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None), 'vec__max_df': 0.4, 'vec__max_features': 1500, 'vec__min_df': 2, 'vec__ngram_range': (1,
1), 'vec__stop_words': frozenset({'of', 'be', 'this', 'are', 'has', 'is', 'at', 'the', 'will', 'that', 'says', 'wha
t', 'and', 'it', 'to', 'can', 'by', 'about', 'in', 'from', 'as', 'with', 'on', 'how', 'for'})}

Cross Validation Accuracy Score: 0.9243055555555556
Training Data Accuracy Score: 0.9861111111111112
Testing Data Accuracy Score: 0.925
```

- Best Model:
  - Logistic Regression: C=1, L2 Penalty
  - TfidfVectorizer, n_gram range = (1,1), max_features=1500

  - Testing Data Accuracy Score = 0.925
  - Training Data Accuracy Score = 0.986

Modeling the Data: Results

```
Best Parameters: {'model': LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False), 'vec': TfidfVectorizer(analyzer='word', binary=False, decode_erro
r='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=0.4, max_features=1500, min_df=2,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=frozenset({'of', 'be', 'this', 'are', 'has', 'is', 'at', 'the', 'will', 'that', 'says', 'what', 'a
nd', 'it', 'to', 'can', 'by', 'about', 'in', 'from', 'as', 'with', 'on', 'how', 'for'}),
        strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None), 'vec__max_df': 0.4, 'vec__max_features': 1500, 'vec__min_df': 2, 'vec__ngram_range': (1,
1), 'vec__stop_words': frozenset({'of', 'be', 'this', 'are', 'has', 'is', 'at', 'the', 'will', 'that', 'says', 'wha
t', 'and', 'it', 'to', 'can', 'by', 'about', 'in', 'from', 'as', 'with', 'on', 'how', 'for'})}

Cross Validation Accuracy Score: 0.9243055555555556
Training Data Accuracy Score: 0.9861111111111112
Testing Data Accuracy Score: 0.925
```

- Best Model:
  - Logistic Regression: C=1, L2 Penalty
  - TfidfVectorizer, n_gram range = (1,1), max_features=1500

  - Testing Data Accuracy Score = 0.925
  - Training Data Accuracy Score = 0.986

Modeling the Data: Look at Beta Values

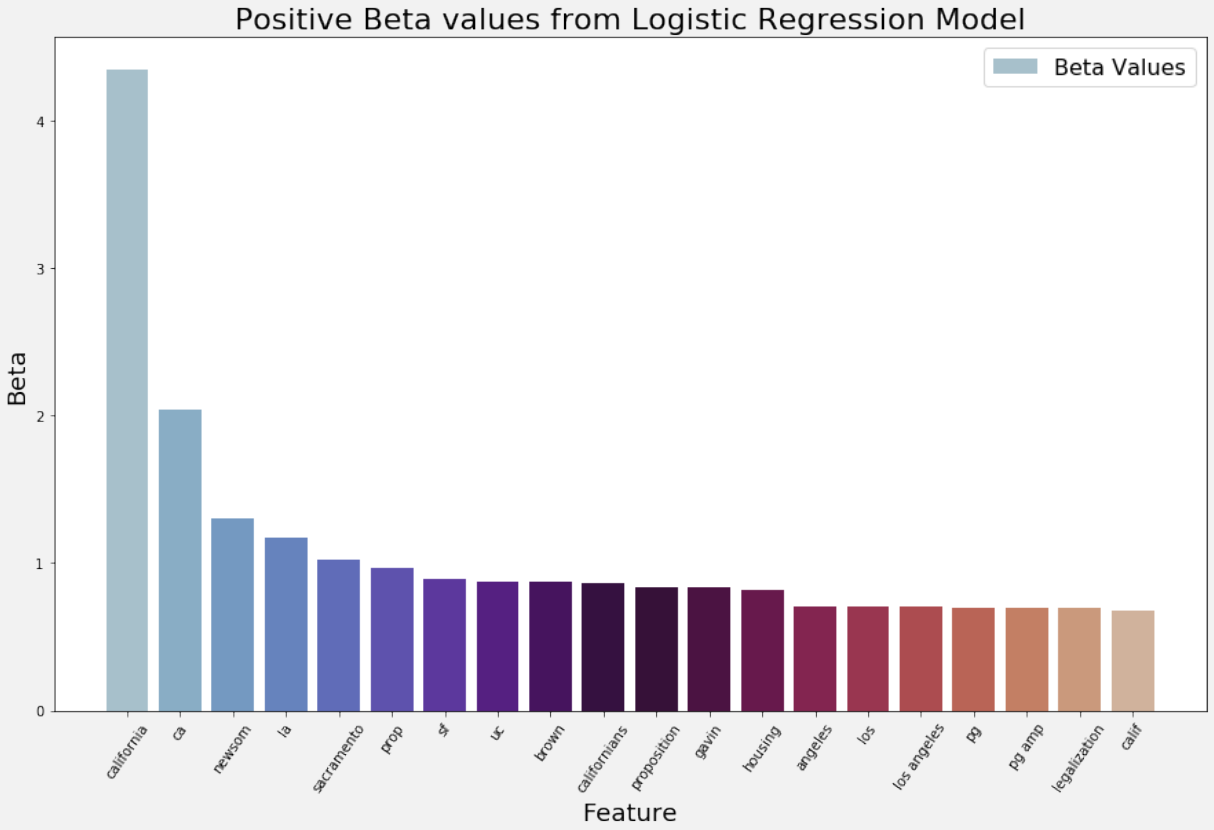
Positive Beta Values

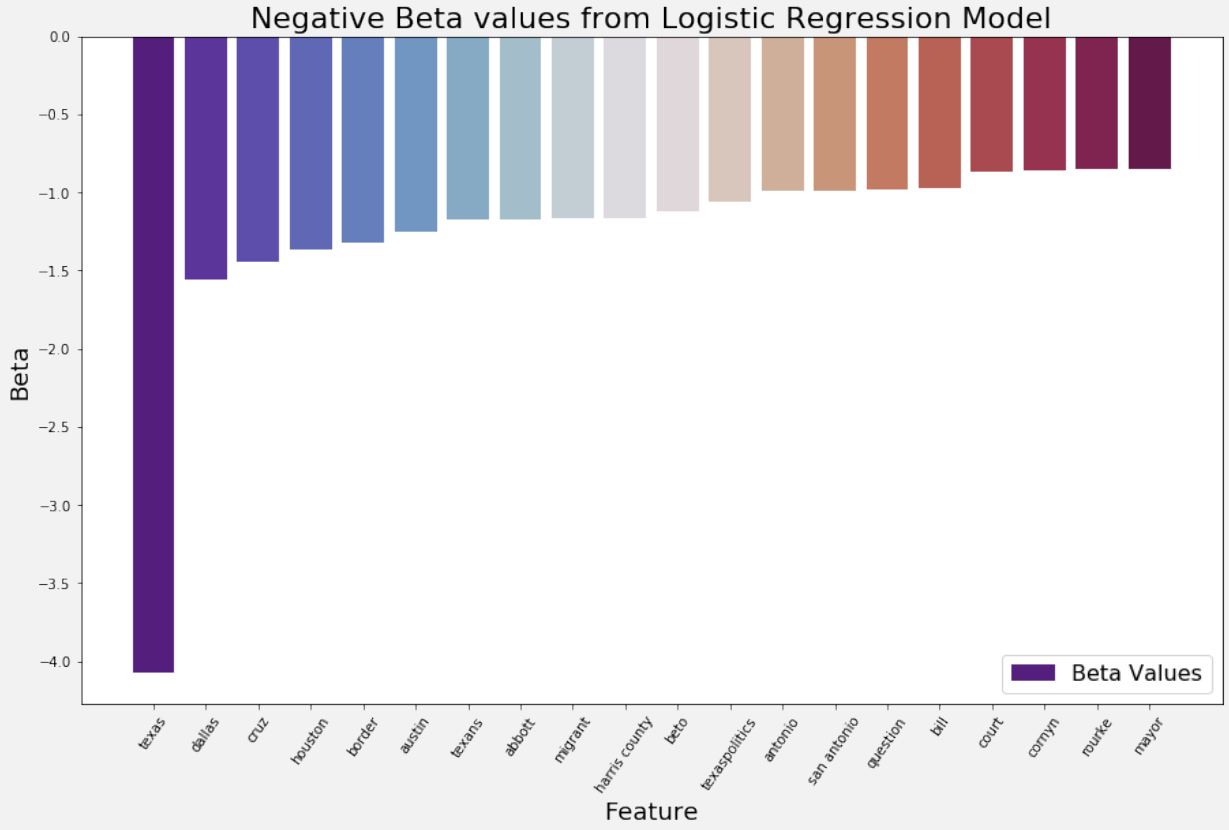
variables	ß - Beta
california	4.348470
ca	2.041404
newsom	1.299012
la	1.172019
sacramento	1.025437
prop	0.967720
sf	0.890655
uc	0.873198
brown	0.871232
californians	0.866007
proposition	0.840535
gavin	0.837271
housing	0.819056
angeles	0.709679
los	0.709679

Negative Beta Values

variables	ß - Beta
texas	-4.071233
dallas	-1.560892
cruz	-1.440955
houston	-1.362039
border	-1.317470
austin	-1.252564
texans	-1.176951
abbott	-1.172178
migrant	-1.161452
harris county	-1.160788
beto	-1.121492
texaspolitics	-1.055495
antonio	-0.987358
san antonio	-0.987358
question	-0.981557

Modeling Data: Beta Values Continued





- Here we can see the intensity of the effect of each feature on predicting a California Politics Subreddit Post
- Note** This is including "California" and "Texas" as features in the classification model.
- Also ran without but didn't include in presentation due to time.

Analysis: What have we learned

- The California & Texas Politics Subreddits are distinctive
- When including all predictive features the model performs with almost 93% accuracy on the testing data
- When removing the two most powerful features the model's accuracy drops to about 80-81%
- Reminder that we looked at titles and not bodies of the posts. This could have made the process a lot different
- Next is to examine what classification errors we had
- From 480 observations:
  - 20 False Positive
  - 17 False Negatives
  - Specificity = 0.918
  - Sensitivity = 0.927

Analysis: What did we misclassify?

Post 1: "Santa Cruz leaders to review homeless solutions, camp alternatives " – False Negative

Post 2: "Judge's Ruling Could Have Big Implications For Proposed High-Speed Rail Line " – False Positive

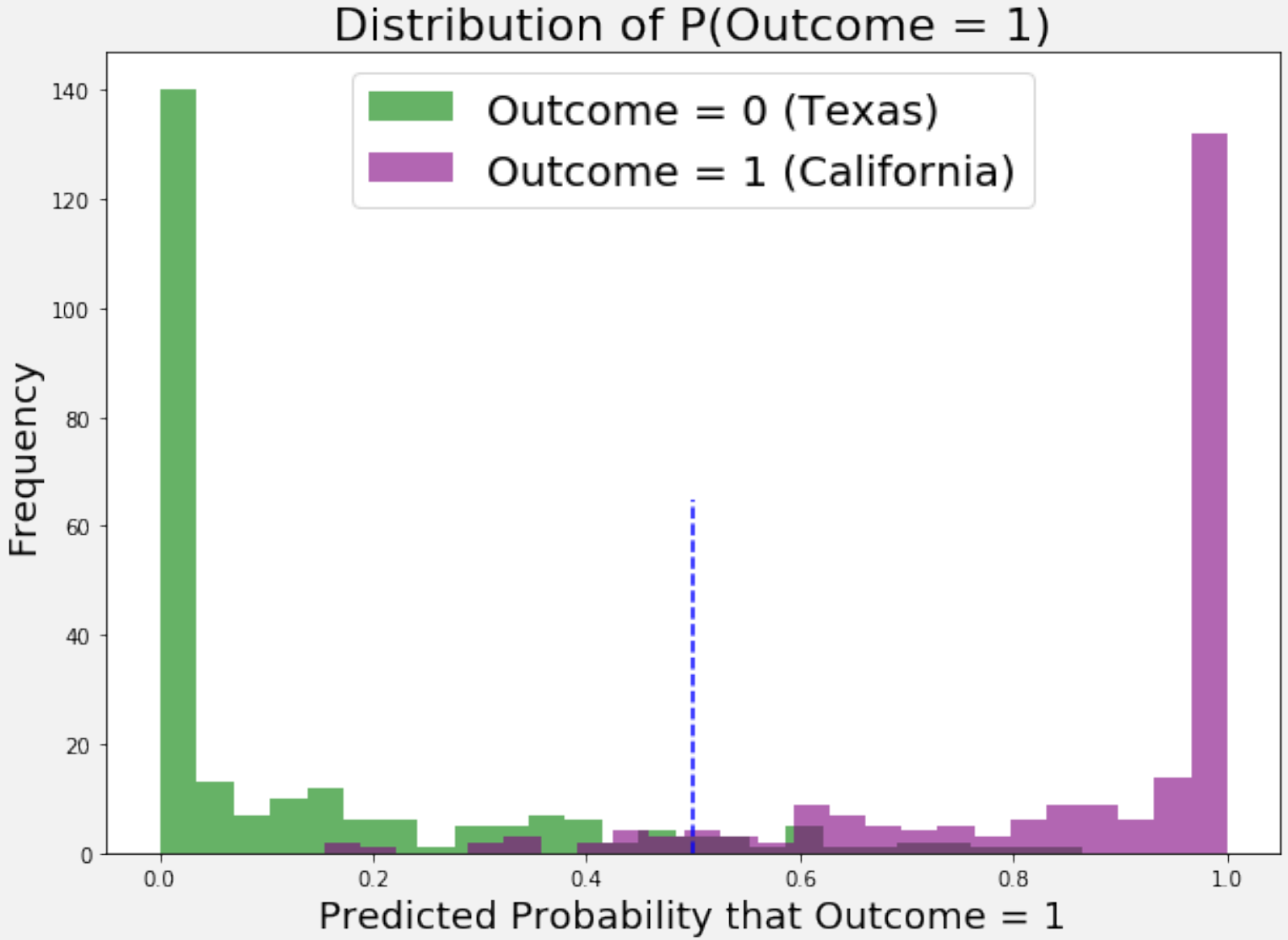
Post 3: "How do decide on how you vote on the judges (Supreme court and 4th district)? Is there somewhere that gives a rundown on all of them?" – False Negative

Post 4: "The Sad State Of Military Housing" – False Positive

Post 5: "Trump Vs. Obama, Texas Vs. California -- Arguing Over Jobs &; The Economy And Why It Matters". - False Positive

Analysis: Probability Distribution

- This gives a clear representation of those values that were classified as:
  - False Positive
  - False Negative
  - True Positive
  - True Negative



Conclusions: Wrapping Up

The NLP was successful

If the two subreddits hadn't contained such high value words it would have made the classification more challenging

Despite removing some of the most high value words, the model still predicted a post with a high rate of accuracy

It would have been nice to continue the data modeling further

Additionally, it would be interesting to pull new posts and see how they are classified.