# Project 3: Web APIs & NLP

Joe Serigano - DSIR 523
1 July 2022

## Our goal:

- Create an NLP model that accurately predicts from which Subreddit a given post originated.

## Problem statement:

- Can we use NLP to discern a good/helpful tip from a bad one?

## Our data:

1. r/LifeProTips
2. r/ShittyLifeProTips

# Our model will compare Subreddit post titles only.

## r/LifeProTips

*“Tips that improve your life in one way or another.”*

- 10,433 posts dating back to April 18, 2022
- Use a heat gun or a blowdryer to help remove stickers easier and faster!
- Check your rental car's trunk for a complete spare and jack kit.

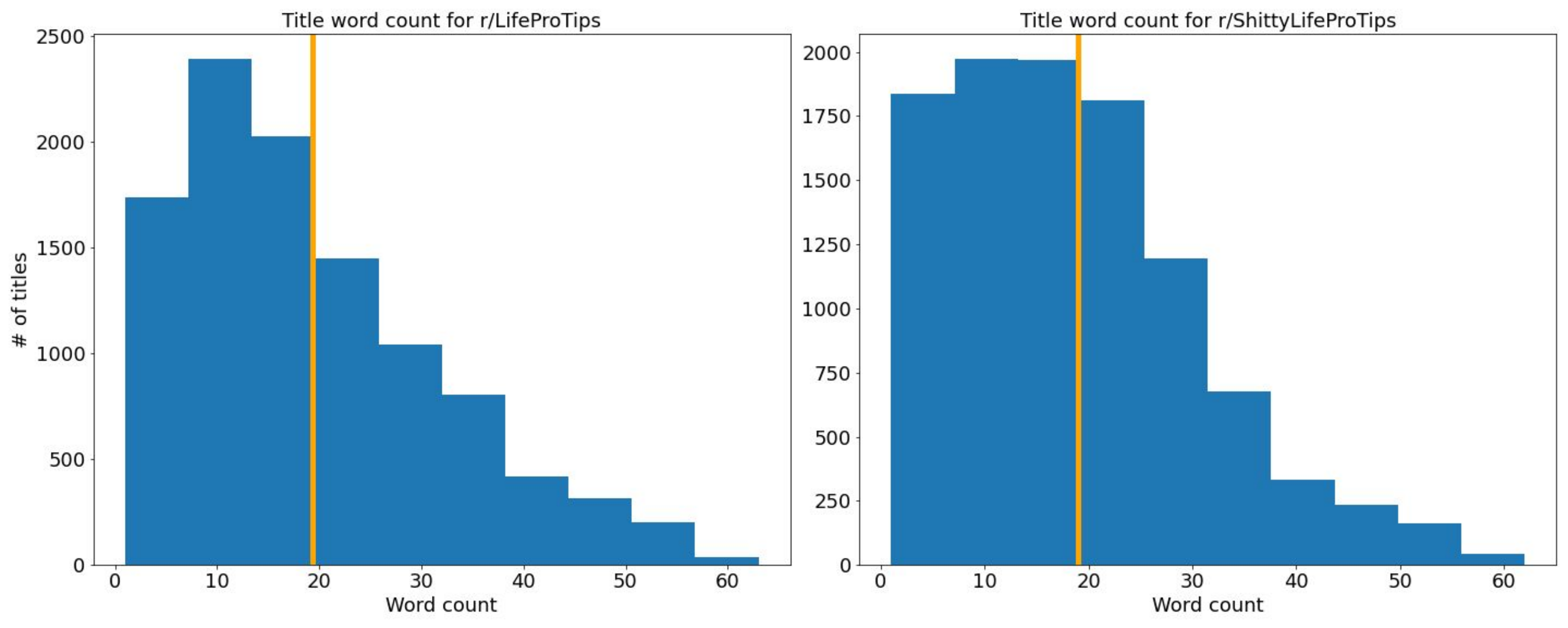
## r/ShittyLifeProTips

*“A place for the shittiest, most mocking "pro-tips" you can think of.”*

- 10,239 posts dating back to Nov 24, 2021
- Walking near a baby in a stroller? Simply pick it up and introduce yourself.
- Charge your phone to 150% and it will last much longer.

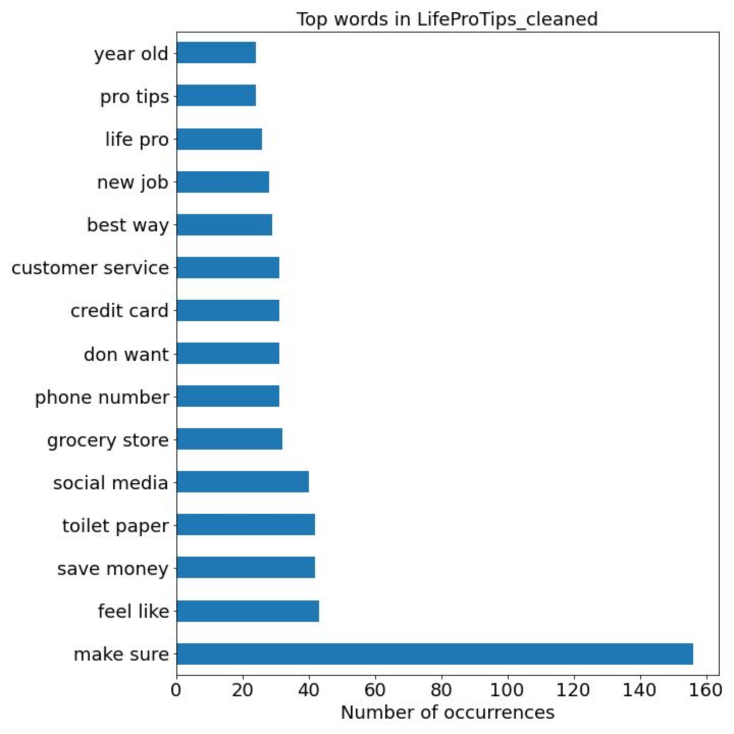
# r/LifeProTips and r/ShittyLifeProTips have very similar posts….

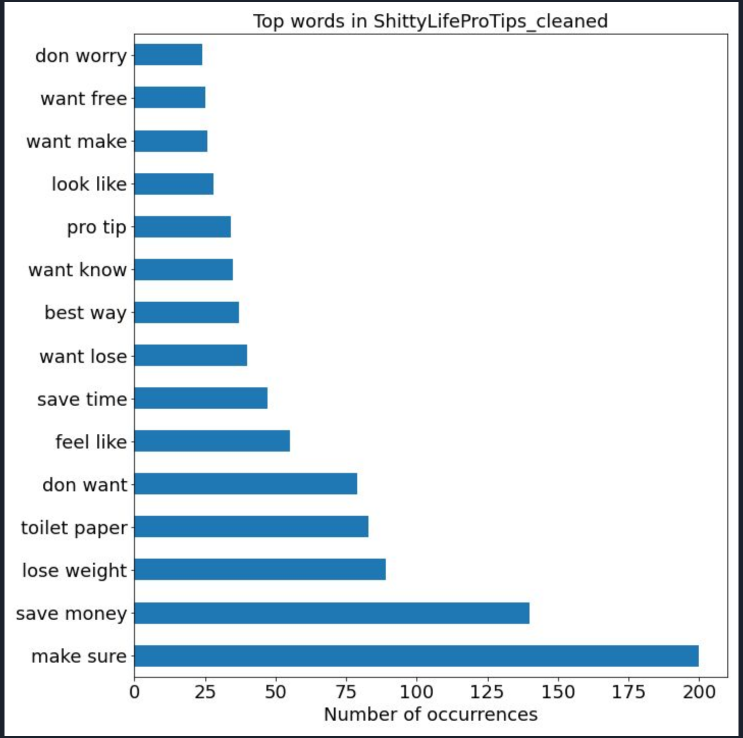
- Average title word count = 19 for both!



# r/LifeProTips and r/ShittyLifeProTips have very similar posts….

- A lot of overlap in the top bigrams





## Model selection:

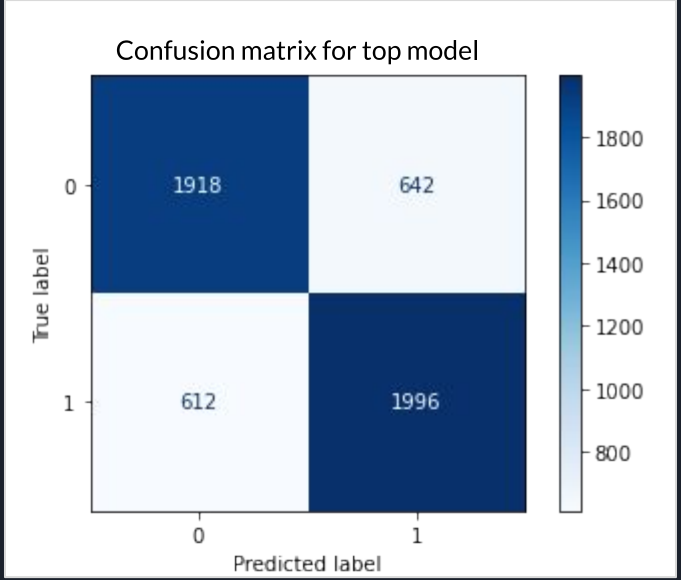
- GridSearchCV to find best parameter values for each combination of vectorizer and classifier
- Vectorizers used:
  - CountVectorizer(), Tfidfvectorizer()
    - Stopwords removed
    - Tokenized, lemmatized, and stemmed versions
- Classifiers used:
  - LogisticRegression(), KNeighborsClassifier(), MultinomialNB(), RandomForestClassifier(), AdaBoostClassifier(), GradientBoostingClassifier()

# Model evaluation:

- Training set: 15,502 posts Test set: 5,168 posts
- Models were each tested individually and evaluated based on various metrics.
    - Top models chosen based on accuracy (.score).
- Baseline accuracy: 50.4%

# Model results:

1. LogisticRegression, TfidfVectorizer:
    - Train: 74.8%  Test: 75.7%
2. MultinomialNB, TfidfVectorizer:
    - Train: 75.4%  Test: 75.6%
3. RandomForest, TfidfVectorizer:
    - Train: 71.0%  Test: 72.7%
4. GradientBoosting, CountVectorizer:
    - Train: 69.4%  Test: 68.8%
5. AdaBoost, TfidfVectorizer:
    - Train: 64.0%  Test: 64.0%
6. KNeighbors, CountVectorizer:
    - Train: 59.5%  Test: 59.8%



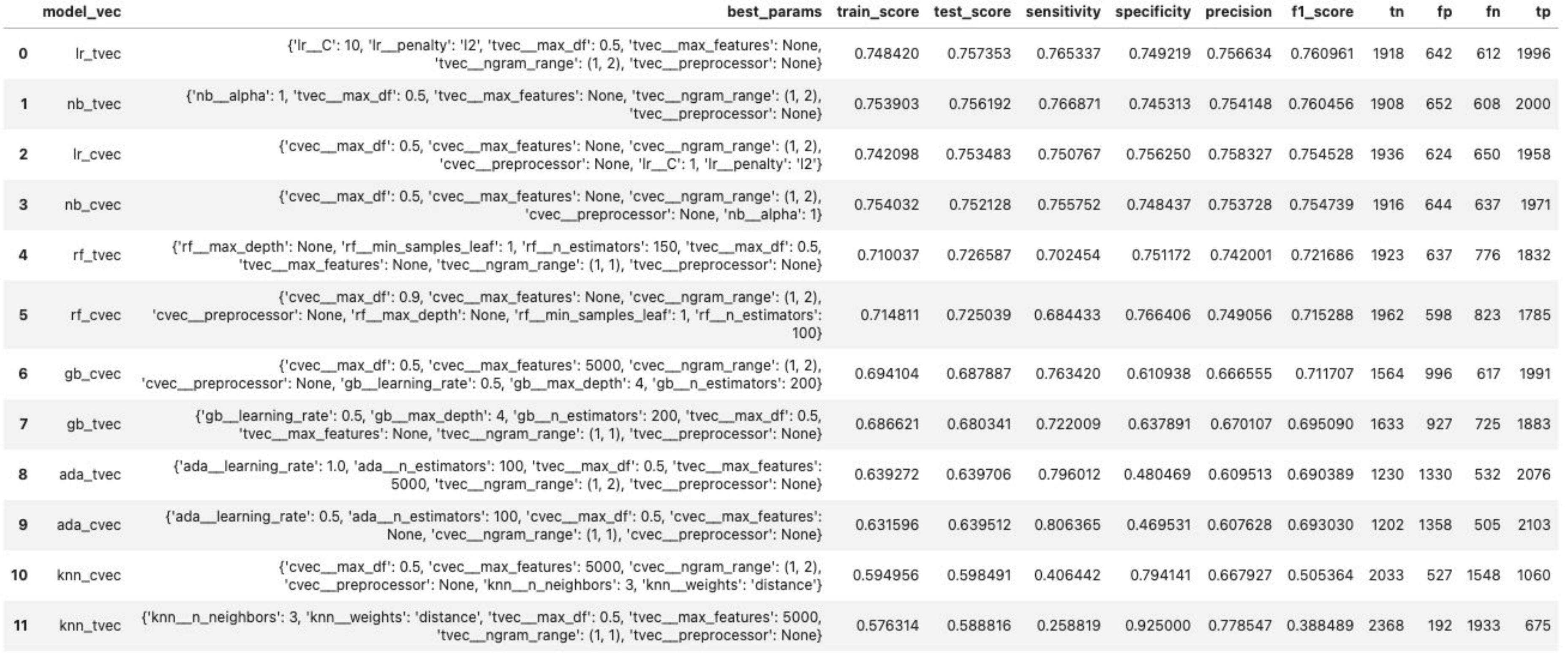
Confusion matrix for top model

## Key takeaways and recommendations:

- Can we discern a good tip from a bad tip with these models?  I wouldn’t bet my life on it….75% accuracy is good but not great!
- All models outperformed the baseline and showed no signs of being overfit.
- Top performing model with a test accuracy of 75.7% is LogisticRegressionClassifier() with TfidfVectorizer().
- A larger data set or an ensemble model could produce models with better performance.

# Supplemental slide:

## Model performances sorted by test score accuracy



| | model_vec | best_params | train_score | test_score | sensitivity | specificity | precision | f1_score | tn | fp | fn | tp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | lr_tvec | {'lr__C': 10, 'lr__penalty': 'l2', 'tvec__max_df': 0.5, 'tvec__max_features': None, 'tvec__ngram_range': (1, 2), 'tvec__preprocessor': None} | 0.748420 | 0.757353 | 0.765337 | 0.749219 | 0.756634 | 0.760961 | 1918 | 642 | 612 | 1996 |
| 1 | nb_tvec | {'nb__alpha': 1, 'tvec__max_df': 0.5, 'tvec__max_features': None, 'tvec__ngram_range': (1, 2), 'tvec__preprocessor': None} | 0.753903 | 0.756192 | 0.766871 | 0.745313 | 0.754148 | 0.760456 | 1908 | 652 | 608 | 2000 |
| 2 | lr_cvec | {'cvec__max_df': 0.5, 'cvec__max_features': None, 'cvec__ngram_range': (1, 2), 'cvec__preprocessor': None, 'lr__C': 1, 'lr__penalty': 'l2'} | 0.742098 | 0.753483 | 0.750767 | 0.756250 | 0.758327 | 0.754528 | 1936 | 624 | 650 | 1958 |
| 3 | nb_cvec | {'cvec__max_df': 0.5, 'cvec__max_features': None, 'cvec__ngram_range': (1, 2), 'cvec__preprocessor': None, 'nb__alpha': 1} | 0.754032 | 0.752128 | 0.755752 | 0.748437 | 0.753728 | 0.754739 | 1916 | 644 | 637 | 1971 |
| 4 | rf_tvec | {'rf__max_depth': None, 'rf__min_samples_leaf': 1, 'rf__n_estimators': 150, 'tvec__max_df': 0.5, 'tvec__max_features': None, 'tvec__ngram_range': (1, 1), 'tvec__preprocessor': None} | 0.710037 | 0.726587 | 0.702454 | 0.751172 | 0.742001 | 0.721686 | 1923 | 637 | 776 | 1832 |
| 5 | rf_cvec | {'cvec__max_df': 0.9, 'cvec__max_features': None, 'cvec__ngram_range': (1, 2), 'cvec__preprocessor': None, 'rf__max_depth': None, 'rf__min_samples_leaf': 1, 'rf__n_estimators': 100} | 0.714811 | 0.725039 | 0.684433 | 0.766406 | 0.749056 | 0.715288 | 1962 | 598 | 823 | 1785 |
| 6 | gb_cvec | {'cvec__max_df': 0.5, 'cvec__max_features': 5000, 'cvec__ngram_range': (1, 2), 'cvec__preprocessor': None, 'gb__learning_rate': 0.5, 'gb__max_depth': 4, 'gb__n_estimators': 200} | 0.694104 | 0.687887 | 0.763420 | 0.610938 | 0.666555 | 0.711707 | 1564 | 996 | 617 | 1991 |
| 7 | gb_tvec | {'gb__learning_rate': 0.5, 'gb__max_depth': 4, 'gb__n_estimators': 200, 'tvec__max_df': 0.5, 'tvec__max_features': None, 'tvec__ngram_range': (1, 1), 'tvec__preprocessor': None} | 0.686621 | 0.680341 | 0.722009 | 0.637891 | 0.670107 | 0.695090 | 1633 | 927 | 725 | 1883 |
| 8 | ada_tvec | {'ada__learning_rate': 1.0, 'ada__n_estimators': 100, 'tvec__max_df': 0.5, 'tvec__max_features': 5000, 'tvec__ngram_range': (1, 2), 'tvec__preprocessor': None} | 0.639272 | 0.639706 | 0.796012 | 0.480469 | 0.609513 | 0.690389 | 1230 | 1330 | 532 | 2076 |
| 9 | ada_cvec | {'ada__learning_rate': 0.5, 'ada__n_estimators': 100, 'cvec__max_df': 0.5, 'cvec__max_features': None, 'cvec__ngram_range': (1, 1), 'cvec__preprocessor': None} | 0.631596 | 0.639512 | 0.806365 | 0.469531 | 0.607628 | 0.693030 | 1202 | 1358 | 505 | 2103 |
| 10 | knn_cvec | {'cvec__max_df': 0.5, 'cvec__max_features': 5000, 'cvec__ngram_range': (1, 2), 'cvec__preprocessor': None, 'knn__n_neighbors': 3, 'knn__weights': 'distance'} | 0.594956 | 0.598491 | 0.406442 | 0.794141 | 0.667927 | 0.505364 | 2033 | 527 | 1548 | 1060 |
| 11 | knn_tvec | {'knn__n_neighbors': 3, 'knn__weights': 'distance', 'tvec__max_df': 0.5, 'tvec__max_features': 5000, 'tvec__ngram_range': (1, 1), 'tvec__preprocessor': None} | 0.576314 | 0.588816 | 0.258819 | 0.925000 | 0.778547 | 0.388489 | 2368 | 192 | 1933 | 675 |