COMP4900 Assignment 2: Text Analysis

Alexander Kuhn

March 3 2020

Abstract

This project was intended to compare the performance of different classifiers on a set of movie reviews. Bernoulli Naive Bayes, Multinomial Naive Bayes, and an SVM with a linear kernel were selected as the classifiers. First, preprocessing was applied to the reviews to standardize capitalization and filter out common words that were irrelevant, then they were subsequently vectorized for use by the classifiers. Next, parameter optimization was performed on the classifiers. The penultimate step was to investigate their performance in 5-fold cross-validation vs. 10-fold. Though all classifiers performed in the 80% range in validation testing, the linear SVM was selected as the optimal classifier, with an average accuracy of 86% $\pm$ 0.5% on the validation set. However, this failed to generalize to test data, ultimately resulting in an accuracy of 51.6% on the test set. Attempts to further reduce the number of features proved somewhat successful at a test accuracy of 67.33%.

1 Introduction

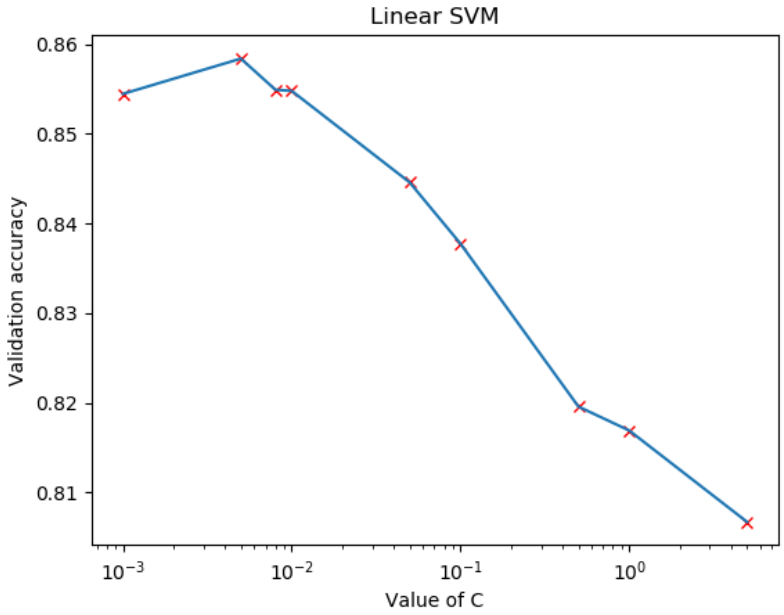
The project was performed in the following stages: initial exploration of dataset, preprocessing of data, hyperparameter optimization of Naive Bayes smoothing and SVM regularization, investigation of different values of k for k-fold cross-validation, and execution of classification tasks. The initial exploration of data involved perusing the train and test .csv files, resulting in the decision to pre-process both by converting to uppercase, removing common words and HTML formatting, and converting to a more streamlined bag of words model separated by spaces. Feature selection was performed using sklearn's CountVectorizer, so that each model only considered the top 5000 words in the dataset. Hyperparameter optimization was completed through a gridsearch of multiple values for alpha on Multinomial Naive Bayes and C on the Linear SVM. Next, a comparison was made between average validation accuracy on 5 folds of cross-validation vs. 10. During this comparison, the average validation accuracy of each classifier was taken into account in order to select the best model for the classification task, which proved to be the linear SVM. The classification task was completed by a search to find the best SVC over 5 folds of cross-validation, and this model was preserved and used to classify the test data.

2 Dataset

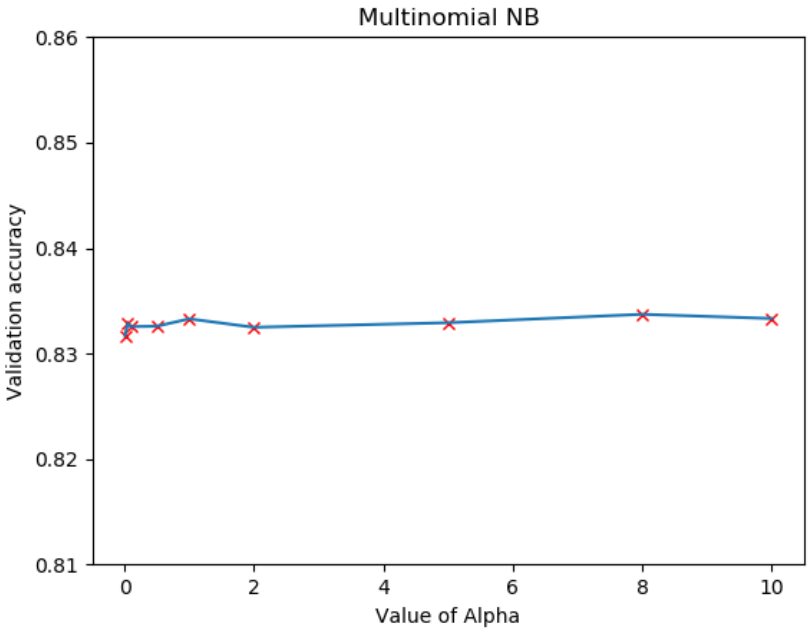
The dataset is a collection of movie reviews. Each sample is a snippet of one review, classified in binary terms of sentiment, positive or negative. The training set is almost perfectly balanced, with a narrow majority (51%) of reviews being positive. For my purposes, each review was broken down in terms of word presence. Then, all words were converted to upper-case, to avoid words with different capitalization being treated differently, and common words (as found in nltk.corpus.stopwords.words [1]) were removed, to reduce noise. As suggested in the specification, the top 5000 words were then selected to use as the features for the classifiers. Bernoulli Naive Bayes and the Linear SVM were fitted to binarized versions of this data, whereas Multinomial Naive Bayes was given the full frequency count of each word.

3 Proposed Approach

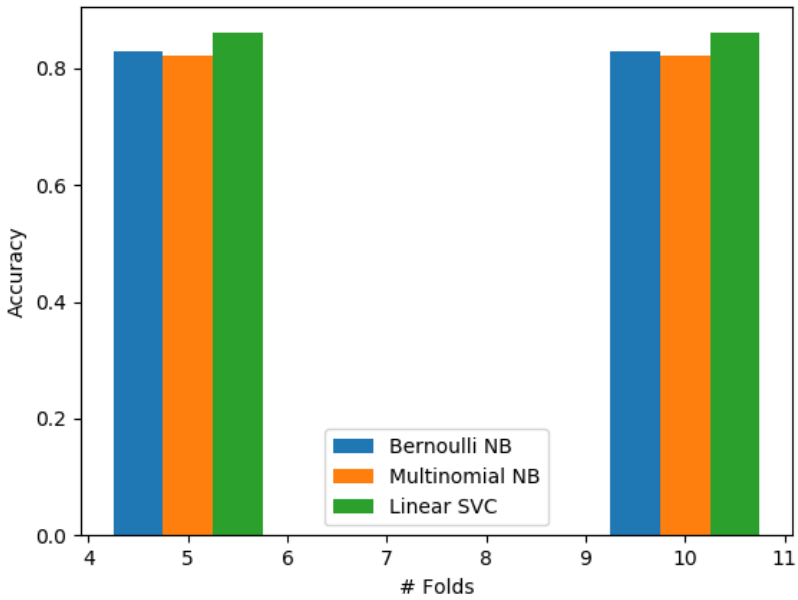
I considered 3 additional classifiers. The first was a multinomial Naive Bayes classifier, the theory being that using the frequency of the words rather than their presence [5] might yield an increase in accuracy. Another was an SVM using a radial basis function kernel, as text classification problems are generally a good fit for SVM approaches [4]. However, this proved to be a poor fit for the data, and it was also taking hours to train. Adjusting to use ScikitLearn's LinearSVC proved superior in terms of both runtime and accuracy. In calibrating the models, two hyperparameters were investigated.



The first (above) was the value of C in the Linear SVM. C helps to regularize the classifier, preventing its overfitting to the test data. At the default value of C, the SVM's performance is disappointing, converging at an accuracy of 0.82 (on the validation set; its performance on the training set is over 99% accurate, indicating an overfitting issue). Through a search on values ranging from 0.001 to 10, I discovered the ideal value was around 0.005, which produced an accuracy of 86% on the validation set and 90% on the training set.
Next, I investigated the value of alpha (the Laplace smoothing parameter) [2] in Multinomial Naive Bayes.



As is evident from the graph, the value of alpha had no impact whatsoever on the performance of the classifier. I therefore decided to leave it at the default in both Naive Bayes models.
With the hyperparameters set, all that remained was to decide on the train/test split for cross-validation. k = 5 and k = 10 "have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance" [3], so I elected to track the average validation accuracy of the models using both values of k.



There was no discernible advantage for either, which resulted in a value of 5 folds being chosen for the final model, as an 80/20 train/test split is faster to train. Initially, I intended to evaluate the models on train time as well. However, no classifier had a train time of greater than 60 seconds per iteration, which I felt rendered the measure irrelevant; the trade-off between a 40-second train time and a one-minute train time was not significant enough to merit a compromise on accuracy. As a result, I selected the Linear SVC as the model of choice for the test set predictions.

4 Results

As seen above, the Linear SVC model had a distinct advantage over its competitors. My final selection for the test set was the best in terms of validation accuracy over 5 iterations of 5-fold cross-validation, at 86.88%. However, this model utterly failed to replicate its success on the test set, with an accuracy of 51.6%. An attempt was made to do better with my Bernoulli Naive Bayes classifier using unprocessed features, which produced a test accuracy of 56%.

5 Discussion and Conclusion

Any conclusions drawn from this report must be viewed with suspicion, considering its failure to produce an accurate reading of the test set. There are a number of potential explanations for this. It is possible that the ID indices

of the test set are being shuffled somehow in the export to a .csv file, or in the prediction process. However, this is unlikely, as when the same process is repeated using the training set, it produces a prediction file with an accuracy of approximately 90%. There is also the potential that this is an extreme incidence of overfitting; on the other hand, this would not explain why the validation accuracy is so low. The most plausible explanation I can think of is that there is an issue with the parameters. Perhaps the features being selected are too specific to the training set, and they are too weak as predictors to handle truly unseen data. Some last-minute submissions lend credence to this hypothesis: with 1000 features, the SVM gets a test accuracy of 67.33%.

If we are to look purely at the results on the validation set, the results are mostly positive. All 3 classifiers are viable options for text classification. The weakest is Multinomial Naive Bayes, which is to be expected, considering the relatively small amount of samples and features NB. The linear SVM is the most effective by a decent margin, although it must be said that getting it to that point required the most work of any classifier. Its C-value at the end, 0.005, is multiple orders of magnitude off from its default value of 1, indicating that it requires a greatly expanded hyperplane to avoid overfitting to the training set. The Bernoulli Naive Bayes model is the most effective "out-of-the-box" model: using only the standard smoothing value of 1.0, it produced a stable accuracy around 83%. However, if one is willing to spend time searching for the proper value of C, the linear support vector machine approach is the best.

6 Statement of Contributions

Alexander Kuhn wrote the code and the report in its entirety. Jimmy Woo is listed as a member of the group, but contributed nothing.

References

[1] Accessing Text Corpora and Lexical Resources, www.nltk.org/book/ch02.html.

[2] C.D. Manning, P. Raghavan and M. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, p. 260.

[3] James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R. Springer, 2017.

[4] Joachims, Thorsten. "Text Categorization with Support Vector Machines: Learning with Many Relevant Features." SpringerLink, Springer, Berlin, Heidelberg, 21 Apr. 1998, link.springer.com/chapter/10.1007/BFb0026683.

[5] "Naive Bayes and Text Classification." Dr. Sebastian Raschka, 4 Oct. 2014, sebastianraschka.com/Articles/2014_naive_bayes_1.html.

```
#nb.py
import␣pandas␣as␣pd
import␣numpy␣as␣np
from␣sklearn.feature_extraction.text␣import␣CountVectorizer
from␣sklearn.model_selection␣import␣KFold




class␣my_Naive_Bayes:
def␣fit(self,␣X,␣y):
num_samples␣=␣X.shape[0]
separated␣=␣[[x␣for␣x,␣a␣in␣zip(X,␣y)␣if␣a␣==␣b]␣for␣b␣in␣np.unique(y)]

numer␣=␣np.array([np.array(i).sum(axis=0)␣for␣i␣in␣separated])␣+␣1
denom␣=␣np.array([len(i)␣+␣2␣for␣i␣in␣separated])

self.probs␣=␣numer␣/␣denom[np.newaxis].T

def␣predict_log_proba(self,␣X):
deltas␣=␣[]

for␣x␣in␣X:
deltas.append((np.log(self.probs)␣*␣x␣+␣\
␣np.log(1␣-␣self.probs)␣*␣np.abs(x␣-␣1)
).sum(axis=1))

return␣deltas

def␣predict(self,␣X):
pred␣=␣np.argmax(self.predict_log_proba(X),␣axis=1)
return␣pred

def␣score(self,␣y_pred,␣y):
count␣=␣0

for␣i␣in␣range(len(y)):
if␣y_pred[i]␣==␣y[i]:
count␣+=␣1

return␣count/len(y)
```

```
#tests.py
import␣pandas␣as␣pd
import␣numpy␣as␣np
from␣nb␣import␣my_Naive_Bayes
from␣sklearn.feature_extraction.text␣import␣CountVectorizer
from␣sklearn.model_selection␣import␣KFold
from␣sklearn.naive_bayes␣import␣MultinomialNB
from␣sklearn.svm␣import␣LinearSVC
import␣matplotlib.pyplot␣as␣plt
from␣bs4␣import␣BeautifulSoup
import␣re
import␣nltk

count␣=␣0
def␣preprocess(data):
global␣count
count␣+=␣1

plain_text␣=␣BeautifulSoup(data,"html.parser").get_text()
letters␣=␣re.sub("[^a-zA-Z]",␣"␣",␣plain_text).upper()
tokens␣=␣nltk.word_tokenize(letters)
stopwords␣=␣set(nltk.corpus.stopwords.words("english"))
words␣=␣[w␣for␣w␣in␣tokens␣if␣not␣w␣in␣stopwords]
words␣=␣[nltk.stem.SnowballStemmer("english").stem(w)␣for␣w␣in␣words]
cleaned␣=␣"␣".join(words)
if␣(count␣%␣5000␣==␣0):
print("cleaned!",␣count)

return␣cleaned

df␣=␣pd.read_csv('train.csv')
df['sentiment']␣=␣np.where(df['sentiment'].str.contains('positive'),␣1,␣0)
y_train␣=␣np.array(df['sentiment'])

df['clean']␣=␣df['review'].apply(preprocess)

#bernoulli␣NB␣classifiers␣require␣that␣features␣be␣binary,␣an␣indicator␣of␣whether␣the␣word␣
vectorizer␣=␣CountVectorizer(max_features=5000,␣binary=True)
X_train␣=␣vectorizer.fit_transform(df['clean']).toarray()
X_train␣=␣np.array(X_train)
print(vectorizer.get_feature_names())

#here␣i␣create␣an␣alternate␣version␣of␣the␣training␣data,␣retaining␣frequencies␣of␣words,␣fo
vectorizer_alt␣=␣CountVectorizer(max_features=5000)
X_train_alt␣=␣vectorizer_alt.fit_transform(df['clean']).toarray()
X_train_alt␣=␣np.array(X_train_alt)
```

```
def svc_opt():
kf = KFold(n_splits=5, shuffle=True)
scores_svm = []
Cvals = [0.001, 0.005, 0.008, 0.01, 0.05, 0.1, 0.5, 1, 5]
for c in Cvals:
scores = []
clf_svm = LinearSVC(C=c, dual = False)
for train_index, test_index in kf.split(X_train):
X_tr, X_val = X_train[train_index], X_train[test_index]
y_tr, y_val = y_train[train_index], y_train[test_index]
clf_svm.fit(X_tr, y_tr)

scores.append(clf_svm.score(X_val, y_val))
scores_svm.append(np.mean(scores))
print("score of", np.mean(scores), "obtained at c of", c)

plt.plot(Cvals, scores_svm, 'rx')
plt.plot(Cvals, scores_svm)
plt.title("Linear SVM")
plt.xscale('log')
plt.xlabel("Value of C")
plt.ylabel("Validation accuracy")
plt.show()

def multi_opt():
kf = KFold(n_splits=5, shuffle=True)
scores_multi = []
alphas = [0.01, 0.05, 0.1, 0.5, 1, 2, 5, 8, 10]
for a in alphas:
scores = []
clf_multi = MultinomialNB(alpha=a)
for train_index, test_index in kf.split(X_train_alt):
X_tr, X_val = X_train_alt[train_index], X_train_alt[test_index]
y_tr, y_val = y_train[train_index], y_train[test_index]
clf_multi.fit(X_tr, y_tr)

scores.append(clf_multi.score(X_val, y_val))
scores_multi.append(np.mean(scores))
print("score of", np.mean(scores), "obtained at alpha of", a)

plt.plot(alphas, scores_multi, 'rx')
plt.plot(alphas, scores_multi)
plt.title("Multinomial NB")
#plt.xscale('log')
plt.xlabel("Value of Alpha")
```

```
plt.ylabel("Validation␣accuracy")
plt.ylim(0.81,0.86)
plt.show()

def␣kfold_opt():
folds␣=␣np.array([5,10])
clf_bernoulli␣=␣my_Naive_Bayes()
clf_multi␣=␣MultinomialNB()
clf_svm␣=␣LinearSVC(C=0.005,␣dual=False)
loops␣=␣5

avg_scores_5␣=␣[0,0,0]
avg_scores_10␣=␣[0,0,0]

for␣i␣in␣range␣(loops):
for␣fold␣in␣folds:
kf␣=␣KFold(n_splits=fold,␣shuffle=True)
for␣train_index,␣test_index␣in␣kf.split(X_train_alt):
X_tr,␣X_val␣=␣X_train[train_index],␣X_train[test_index]
y_tr,␣y_val␣=␣y_train[train_index],␣y_train[test_index]
X_tr_alt,␣X_val_alt␣=␣X_train_alt[train_index],␣X_train_alt[test_index]

clf_bernoulli.fit(X_tr,␣y_tr)
clf_multi.fit(X_tr_alt,␣y_tr)
clf_svm.fit(X_tr,␣y_tr)

y_pred␣=␣clf_bernoulli.predict(X_val)
score_bernoulli=␣clf_bernoulli.score(y_pred,␣y_val)
score_multi␣=␣clf_multi.score(X_val_alt,␣y_val)
score_svm␣=␣clf_svm.score(X_val,␣y_val)
#print(score_svm)

if␣(fold==5):
avg_scores_5[0]␣+=␣score_bernoulli
avg_scores_5[1]␣+=␣score_multi
avg_scores_5[2]␣+=␣score_svm
else:
avg_scores_10[0]␣+=␣score_bernoulli
avg_scores_10[1]␣+=␣score_multi
avg_scores_10[2]␣+=␣score_svm
print␣("iteration",␣i,␣"complete")

avg_scores_5␣=␣np.array(avg_scores_5)/(loops*folds[0])
avg_scores_10␣=␣np.array(avg_scores_10)/(loops*folds[1])

bernoulli_scores␣=␣[avg_scores_5[0],avg_scores_10[0]]
```

```
multi_scores = [avg_scores_5[1],avg_scores_10[1]]
svm_scores = [avg_scores_5[2],avg_scores_10[2]]

ax = plt.subplot(111)
plt.bar(x=(folds-0.5), height=bernoulli_scores, label='Bernoulli NB', width=0.5)
plt.bar(x=folds, height=multi_scores, label='Multinomial NB', width=0.5)
plt.bar(x=(folds+0.5), height=svm_scores, label='Linear SVC', width=0.5)
plt.xlabel('# Folds')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


#svc_opt()
#multi_opt()
#kfold_opt()

clf = LinearSVC(C=0.005, dual=False)
kf = KFold(n_splits=5, shuffle=True)
bestScore = 0

for i in range(5):
for train_index, test_index in kf.split(X_train):
X_tr, X_val = X_train[train_index], X_train[test_index]
y_tr, y_val = y_train[train_index], y_train[test_index]

clf.fit(X_tr, y_tr)

currScore = clf.score(X_val, y_val)

if (currScore > bestScore):
best = clf
bestScore = currScore
print("done iteration")
print("Training complete")

print (bestScore)
clf = best

df_test = pd.read_csv('test.csv')
df_test['clean'] = df_test['review'].apply(preprocess)
vectorizer = CountVectorizer(max_features=5000, binary=True)
X_test = vectorizer.fit_transform(df_test['clean']).toarray()

X_test = np.array(X_test)
y_pred = clf.predict(X_test)
```

```
y_pred␣=␣y_pred.astype('str')
y_pred␣=␣np.where(y_pred␣==␣'1',␣'positive',␣'negative')
df_test['sentiment']␣=␣y_pred

df_test.index.name='id'
df_test['sentiment'].to_csv('predictions.csv',␣index=True,␣header=True)
```