DNSC 6279 Data Mining Final Report

Online news popularity prediction

Report Index

1. Problem Description

1.1. Dataset Review

Our group chose to use the *Online News Popularity Data Set* from UCI Machine Learning Repository. This dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. The goal is to predict the number of shares in social networks (popularity).

1.2. Problem Description

Our main goal is to predict the popularity of news or articles through the given variables using different algorithms. During this period, we tried to use regression algorithms to predict the number of shares. Unfortunately, every model we tried does not perform well on predicting the number of shares: the evaluation metrics, R squared is pretty close to 0, meaning that the response cannot be explained by our models at all, and the lowest test RMSE we achieved is greater than 7000. Besides, the higher the Regularization parameter was, the better the model performs, which indicates that the model did such a poor job that even using the mean of the number of shares to make predictions can result in a better outcome. Instead, we convert the regression problem into a classification problem and will develop some learning algorithms that classify news as popular or unpopular.

1.3. Pair plot of predictors

We also visualized the correlation between predictors.

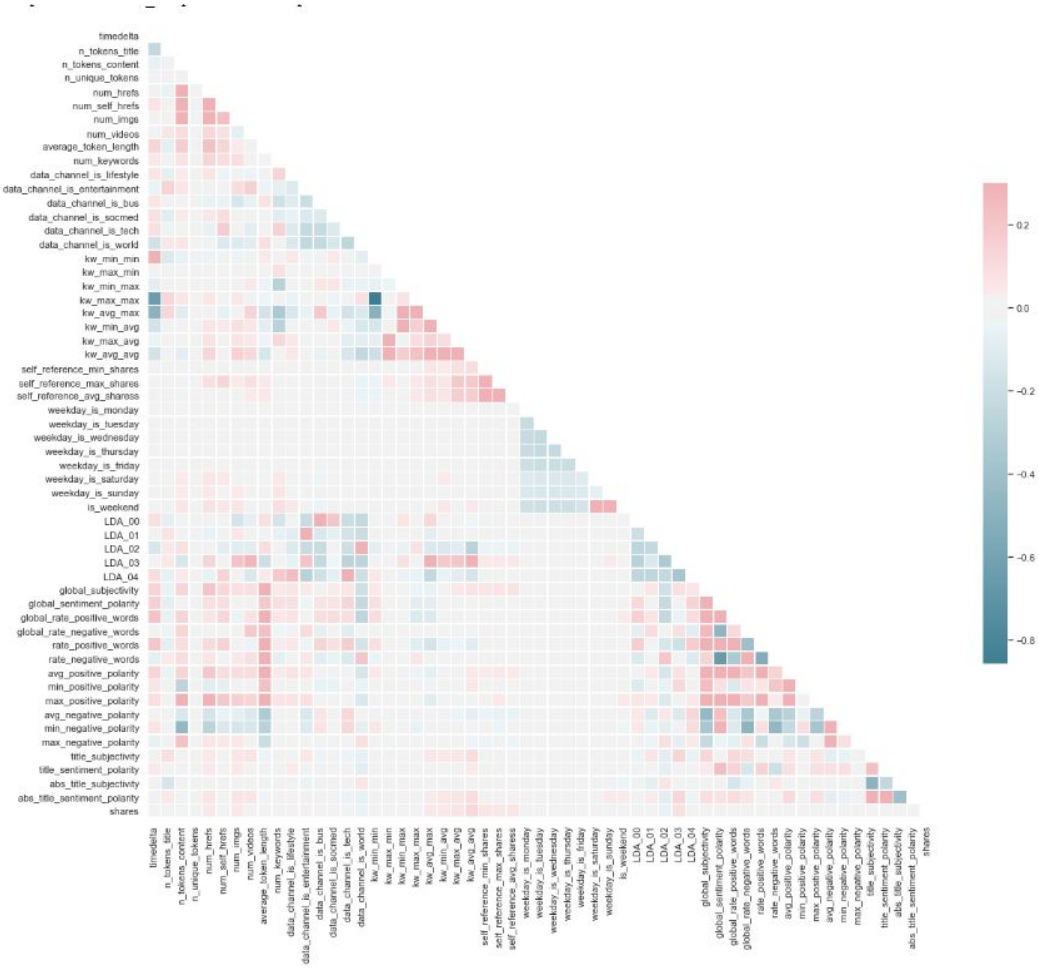

Figure 1.1 Pearson Correlation

As we can see from the plot, most of the predictors are uncorrelated, but some of them are highly correlated. We removed highly correlated predictors accordingly.

2. Pre-selection of feature

The dataset contains 61 columns, including one URL column, one goal column (popularity) and 59 variables. Thus, due to the large number of features we have, before applying algorithms to obtain a higher prediction rate, we pre-select some features that have a higher prediction rate using a forward stepwise method and XGboost.

2.1. Forward Stepwise with BIC

To simplify the problem, we apply *the stepwise algorithm* to the dataset. Using a 5-fold *Cross-Validation* Approach, we obtain 5 groups of train and test sets. Applying each group with the *forward stepwise algorithm* choosing nvmax=60, we obtain n, which is the best number of variables that should be chosen, by calculating the minimum value of BIC. As shown below, in the forward stepwise algorithm, we obtain 11, 9, 9, 11, 11 respectively for the *5-fold cross-validation.* Thus, from this method, we chose 11 as the number of variables used.

This feature selection approach may not be considered appropriate because the best number of variables is dependent on the learning algorithm we applied to the dataset. However, it will be computational intensive if we are trying to find the optimal combination of the number of variables to use and the set of hyper-parameters of a specific learning algorithm. Thus, the naive approach can make our life easier and will provide us with a good starting point. In the future study, we will move from the naive approach to some more appropriate approaches and do further investigations.

```
> n_choose_f
[1] 11  9  9 11 11
```

Below shows the 11 variables we chose from the 59 variables in the dataset.

```
> coefficients(md.fit1, id = 11)
                  (Intercept)                     timedelta
                -2.221250e+03                 1.873890e+00
               n_tokens_title                     num_hrefs
                 1.146622e+02                 3.466283e+01
average_token_length data_channel_is_entertainment
                -4.630659e+02                -6.170197e+02
                  kw_min_avg                     kw_max_avg
                -4.720347e-01                -2.083471e-01
                  kw_avg_avg     self_reference_min_shares
                 1.854099e+00                 2.614821e-02
 global_subjectivity         max_negative_polarity
                2.801300e+03                 -1.338366e+03
```

Figure 2.1 Stepwise feature selection

2.2. Shapley Values

Another method to decide which variables to consider is based on Xgboost and Shapley values. The Shapley value is the average marginal contribution of a feature value across all possible coalitions, which

provides our insights on how important one variable can affect our target value.
But please note that Shapley values cannot achieve feature selection, what data scientists usually do is to rank the Shapley values, then decide to choose Top X variables to put in the next step's prediction.

We firstly applied the Xgboost method on the whole dataset, then obtained the Shapley values. (Here Xgboost is only the base model to calculate the Shapley values, thus we don't report the model details here. How we construct Xgboost model, please see 3.3.5)

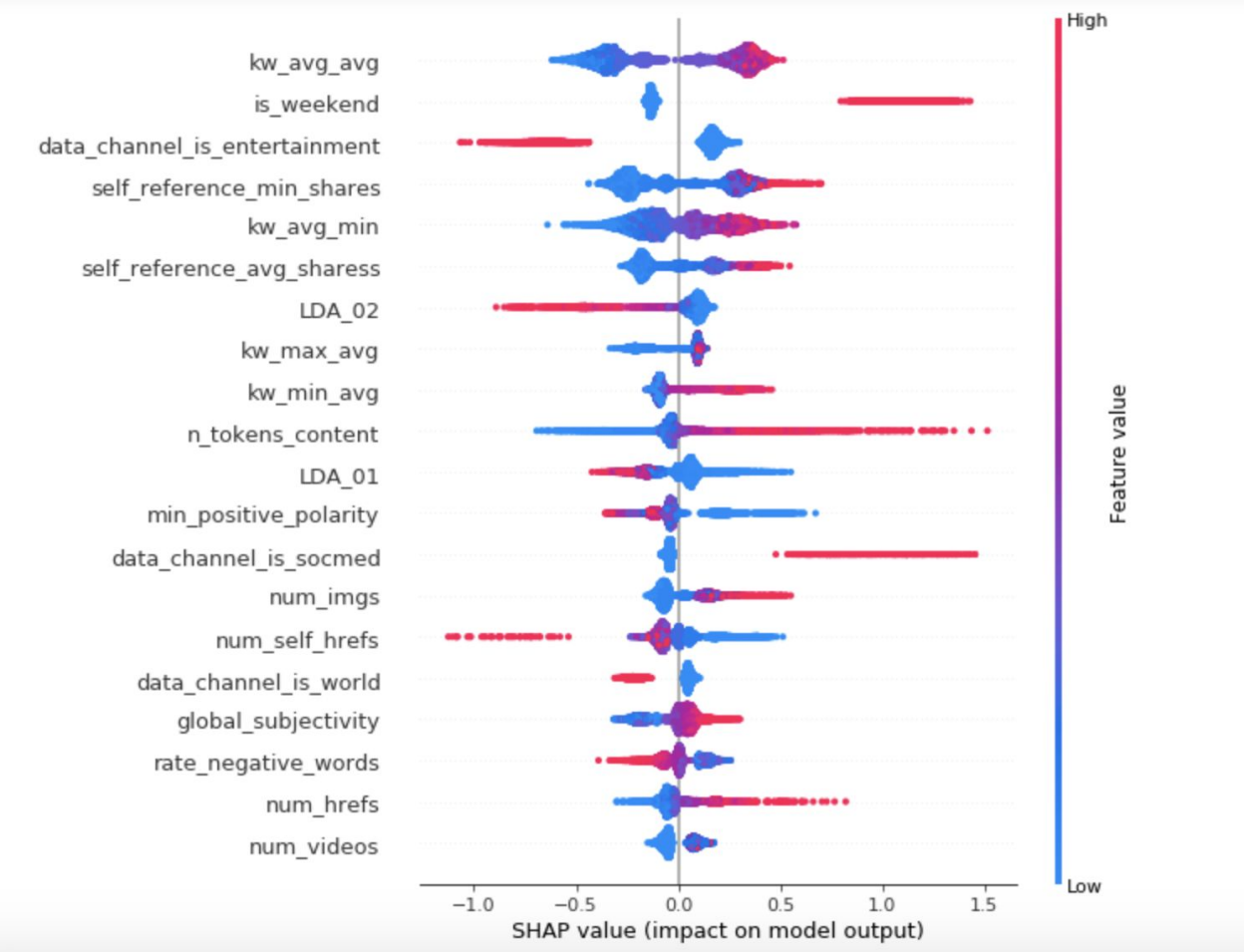

Figure 2.2 Shapley value

In order to see the variables' impact more clearly, we plot each variable's Shapley value, this graph shows the top20 variables with highest Shapley values.

Compared to the forward stepwise selection result, we found that all of 11 variables selected by forward stepwise are in the top20 Shapley values list. Which means that the forward stepwise result is reliable.

3. Algorithms

3.1. Overview

After feature selection, we will use these 11 features to do our data analysis. Our analysis can be split into two parts, “regression” and “classification”.

For regression, we use different regression methods to evaluate whether our target variable “Share” can be exactly predicted. Linear Regression, Ridge Regression and Lasso Regression were chosen to fit the model. And we use R-Square and RMSE to evaluate their goodness.

For the classification problem, we encoded our target, that is, number of shares, into binary format. Observations labeled with 1 are considered popular, while observations labeled with 0 are considered unpopular.

We will elaborate how we label our target variable later.

3.2. Regression

3.2.1. Linear Regression

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression.

As shown above, the linear regression method has poor performance. The root mean squared error is above 12000, while R square is smaller than 0.02. Thus, linear regression is proved to have poor predictive power on the data.

3.2.2. Ridge Regression

Ridge regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables). In general, the method provides improved efficiency in parameter estimation problems in exchange for a tolerable amount of bias.

```
In [32]: #try ridge regression. Do not run this cell if you're not confident with the computational power of your machine.

ridge = RidgeRegression(degree = 1,alpha = 1) #It's computationally intensive if degree is higer than 2 since we have c
ridge.fit(X_train,y_train)
para = {
        'poly__degree':[1,2],
    'ridge_reg__alpha':[0.1,0.5,1,2,5]
}
ridge_GS = GridSearchCV(ridge,param_grid=para,cv=5,scoring='r2',n_jobs=-1)
ridge_GS.fit(X_test,y_test)
test_RMSE(ridge_GS,X_test,y_test) #poor performance

Out[32]: 8225.785232176675

In [33]: predicY=ridge_GS.predict(X_test)
testY=y_test
Rsquare()

R2: 0.03956033075307697
```

As shown above, Ridge regression also has a poor performance to interpret the dataset. Test root mean square error is large and R square is small. Thus Ridge regression has a poor performance on the dataset given.

3.2.3. Lasso Regression

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models.

```
In [34]: #Do not run this cell if you're not confident with the computational power of your machine.
lasso = LassoRegression(degree = 1,alpha = 1)
lasso.fit(X_train,y_train)
para = {
        'poly__degree':[1,2],
    'lasso_reg__alpha':[0.1,0.5,1,2,5]
}
lasso_GS = GridSearchCV(lasso,param_grid=para,cv=5,scoring='r2',n_jobs=-1)
lasso_GS.fit(X_test,y_test)
test_RMSE(lasso_GS,X_test,y_test) #poor performance

Out[34]: 8226.114756471543

In [35]: predicY=lasso_GS.predict(X_test)
testY=y_test
Rsquare()

R2: 0.03948337893994702
```

Similarly, the test Root Mean Squared Error is above 8000 in the test set, while R square is smaller than 0.05. Thus Lasso regression has a poor performance on the dataset given.

3.2.4. Poisson Regression

Poisson regression assumes the response variable Y has a Poisson distribution, and assumes the logarithm of its expected value can be modeled by a linear combination of unknown parameters. We also apply poisson regression on our dataset.

Generalized Linear Model Regression Results

Dep. Variable:	shares	No. Observations:	31715
Model:	GLM	Df Residuals:	31705
Model Family:	Poisson	Df Model:	9
Link Function:	log	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-1.1355e+08
Date:	Mon, 13 Apr 2020	Deviance:	2.2680e+08
Time:	14:05:14	Pearson chi2:	1.78e+09
No. Iterations:	15		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
x1	1.7270	0.001	1915.468	0.000	1.725	1.729
x2	0.7719	0.001	580.825	0.000	0.769	0.774
x3	1.3773	0.001	1062.211	0.000	1.375	1.380
x4	-0.2032	0.000	-743.883	0.000	-0.204	-0.203
x5	-0.3433	0.000	-950.000	0.000	-0.344	-0.343
x6	-24.9187	0.011	-2251.067	0.000	-24.940	-24.897
x7	19.5935	0.006	3389.253	0.000	19.582	19.605
x8	2.2756	0.001	1643.054	0.000	2.273	2.278
x9	1.7731	0.001	1671.686	0.000	1.771	1.775
x10	5.1969	0.001	6597.469	0.000	5.195	5.198

```
In [17]: poisson_predictions=poisson_training_results.get_prediction(X_train_standard)
         predictions_summary_frame=poisson_predictions.summary_frame()

In [19]: y_predict=predictions_summary_frame['mean']
         #RMSE
         np.sqrt(mean_squared_error(y_train,y_predict))

Out[19]: 12796.929520747984
```

As shown above, the log-likelihood parameter of the training model is negative 1*10^8, which indicates its deviance goodness of fit is also positive 2*10^8 (deviance=-2*log-likelihood). Since the deviance of the model is pretty large, it has a poor performance. Moreover, the RMSE is even larger than 12000. This indicates that Poisson regression has a poor performance on the Online News Popularity dataset.

3.2.5. Result

Thus, to summarize, regression has very poor performance in interpreting the online news popularity dataset.

3.3. Classification

3.3.1. Data preprocessing for classification problems

We manually label these observations to be popular and unpopular based on the number of shares. At beginning the cut off value is the median, however, we figured out that by doing so, our positive observations and negative observations are mixed together, and apparently we are not able to implement a classification algorithm to divide them. The solution to the problem is to label observations with the number of shares higher than 75% quantile as positive, and the number of shares lower than 25% quantile as negative, and then get rid of other observations in the training set so as to make our classifiers more sensitive. Another thing that we want to mention here, is that for classification problems we did not do feature pre-selection since after doing so, the performance of our classifiers even became worse. Thus keeping all the features here may be a good choice.

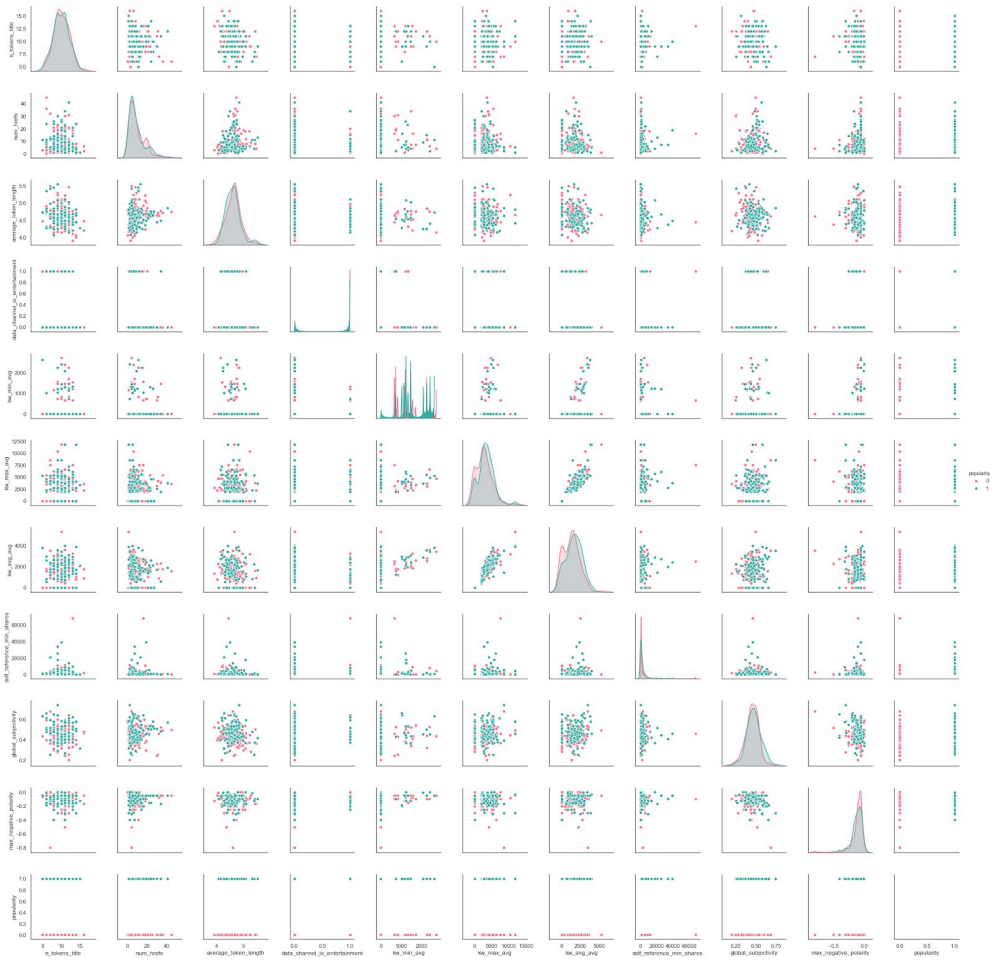

Figure 3.1 Pair plots of some predictors after data processing

3.3.2. Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The Naive Bayes classifier tries to approximate the Bayes classifier, which is considered to be the optimal classifier.

$$\Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}.$$

However, in the real-world, estimating fk(x) tends to be more challenging, unless we assume some simple forms for these densities. In the setting of Naive Bayes classifier, we assume that all the predictors follow a normal distribution. Although It is not a reasonable assumption, it still might be useful when we are trying to make things easier.

```
from sklearn.model_selection import cross_val_score
from numpy import *
nb=naive_bayes.GaussianNB()
nb.fit(X_train_standard, y_train)
roc_scores = cross_val_score(nb, X_train_standard, y_train, cv=5, scoring='roc_auc')
acc_scores = cross_val_score(nb, X_train_standard, y_train, cv=5, scoring='accuracy')
f1_scores = cross_val_score(nb, X_train_standard, y_train, cv=5, scoring='f1')
print('Best validation ROC_AUC score: ',mean(roc_scores))
print('Best validation Accuracy score: ',mean(acc_scores))
print('Best validation f1 score: ',mean(f1_scores))
```

```
Best validation ROC_AUC score:  0.7380295585398189
Best validation Accuracy score:  0.633527798512881
Best validation f1 score:  0.5367153592948014
```

This model gives us an accuracy score of 0.6335 and an AUC score of 0.738 and f1 of 0.537. The metrics shown above indicate that it is highly likely that our assumptions are violated. Since

3.3.3. K Nearest Neighbors

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of the 1970's as a non-parametric technique.

One of the advantages of the non-parametric method is that we do not need to make assumptions in terms of what is the true relationship between the predictors and the response. Instead, the KNN algorithm uses the majority voting mechanism to decide which category that the data point should belong to. Even though KNN is a non-parametric method, we still need to choose the set of hyperparameters to achieve the best result.

In order to find the best hyperparameters of the model, for example, the number of neighbors we used to make predictions, we used grid search cross-validation to search for the best set of hyper-parameters.

```
knn_roc,knn_estimator = knn_grid_search.best_score_,knn_grid_search.best_estimator_
print('Best parameters: ',knn_grid_search.best_params_)
print('Best validation ROC_AUC score: ',knn_roc)
knn_grid_search.score
```

```
Best parameters:  {'n_neighbors': 35, 'weights': 'distance'}
Best validation ROC_AUC score:  0.7486195831280712
```

In the result, the best k number is 35.

```
from sklearn.model_selection import cross_val_score
from numpy import  *
roc_scores = cross_val_score(knn_grid_search, X_train_standard, y_train, cv=5, scoring='roc_auc')
acc_scores = cross_val_score(knn_grid_search, X_train_standard, y_train, cv=5, scoring='accuracy')
f1_scores = cross_val_score(knn_grid_search, X_train_standard, y_train, cv=5, scoring='f1')
print('Best validation ROC_AUC score: ',mean(roc_scores))
print('Best validation Accuracy score: ',mean(acc_scores))
print('Best validation f1 score: ',mean(f1_scores))
```

```
Best validation ROC_AUC score:  0.7486195831280712
Best validation Accuracy score:  0.6859616155955187
Best validation f1 score:  0.6748627330214064
```

This model gives us an accuracy score of 0.6860, an AUC score of 0.7486 and f1 of 0.6749.

3.3.4. Logistic regression

Logistic regression is a commonly used supervised machine learning algorithm to classify data points into different categories. The name Logistic Regression indicates that the underlying

technique is derived from linear regression, namely, logistic regression is a generalized linear regression.

$$\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}.$$

In logistic regression, we assume that the log-odds have some linear relationship with the predictors.

We start with a simple model, assuming that the underlying decision boundary is linear. All parameters remain default.
One problem of this model is that we take all variables into account, and it may make the model have a large variance. In order to balance the variance and bias, we decided to add regularization terms to our model and use cross-validation to measure which approach (L1 or L2) does a better job.

```
%%time
parameters = {
    'penalty':['l2','l1'],
    'C':[10, 100, 200, 500]
}
log_roc, log_estimator = classifier(logistic, X_train_under, y_train_under, X_test_standard, y_test, gridsearch = True, parameters = parameters)
```

```
Best parameters:  {'C': 100, 'penalty': 'l1'}
Best validation ROC_AUC score:  0.7860564192260355
Best validation F1 score:  0.7179741244583706
Best validation accuracy score:  0.7179741244583706
```

As we can see from the screenshot above, using L1 regularization with a penalty parameter C (referred to as lambda in our textbook) of 100 gives us the cross-validation score. Since the value of the optimal penalty parameter does not lie on the upper or lower limit of our search space, we are safe to say that the set of parameters is optimal.

In the real world, the linear assumption is always violated. Unfortunately, even though we would like to go beyond the linearity and to improve the performance of the logistic regression classifier, by adding polynomial features to this model, we have too many features to fit the polynomial model and doing so always makes our machine crash down. Thus, this topic will not be covered in this report.

3.3.5. Support vector classifier(SVC)

Another classification algorithm to be used in this project is support vector classifier. The idea behind SVC is that the classifier is trying to find a hyperplane that separates data points that belong to different categories.

The RBF kernel function is strictly superior to other kernel functions in practice. It will project our training samples to a high dimensional space and try to find out a hyperplane that separates them apart. Usually, there is not a hyperplane that can do the perfect separation, therefore we also set the tolerance parameter C that allows our support vector classifier to make some mistakes. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close' (from the homepage of sklearn), and we are also going to tune this parameter.

We still tried to use grid search cross-validation to find the optimal set of hyper-parameters.

After tuning hyper parameters, the best result we got is shown below.

```
%%time
svc = SVC(gamma='auto',cache_size= 11000)
parameters = {'gamma':[0.1,1,5,10],
    'kernel':['rbf'],
    'C':[1,5,10,20,50],
}
svc_roc,svc_estimator = classifier(svc,X_train_under,y_train_under,X_test_standard,y_test,gridsearch = True, parameters = parameters )
```

```
Best parameters:  {'C': 20, 'gamma': 0.1, 'kernel': 'rbf'}
Best validation ROC_AUC score:  0.7844940349471665
Best validation F1 score:  0.716596247340014
Best validation accuracy score:  0.716596247340014
```

We select the optimal set of hyperparameters with C being 20 and gamma being 0.1. The ROC score on the validation set is about 0.78, the accuracy is 0.72 and the F1 score is 0.72.

3.3.6. XGboost

Inspired by the guest speaker after our Midterm exam, we tried XGboost with Monotonic constraints as another classification

method. Monotonic relationships can prevent overfitting and excess error due to variance for new data.

According to the material provided by Prof. Patrick Hall, we can find the monotonic relationship by calculating the pairwise Pearson correlations between variables and our target - whether an online news is popular.

We obtained the Pearson correlation by Pandas .corr() function.

```
pd.DataFrame(data[X + [y]].corr()[y]).iloc[:-1]
```

	shares
timedelta	0.008662
n_tokens_title	0.008783
n_tokens_content	0.002459
n_unique_tokens	0.000806
n_non_stop_words	0.000443
n_non_stop_unique_tokens	0.000114
num_hrefs	0.045404
num_self_hrefs	-0.001900
num_imgs	0.039388
num_videos	0.023936
average_token_length	-0.022007
num_keywords	0.021818
data_channel_is_lifestyle	0.005831
data_channel_is_entertainment	-0.017006
data_channel_is_bus	-0.012376
data_channel_is_socmed	0.005021
data_channel_is_tech	-0.013253
data_channel_is_world	-0.049497

From the correlation results, we can see that average_token_length,data_channel_is_entertainment, data_channel_is_tech, data_channel is bus.etc are negatively related to our response. Then we convert the correlation results into (-1,1) mono_constraints - If the Pearson correlation between an input variable and ‘shares’ is positive, a positive monotonic relationship constraint is specified for that variable using 1. If the correlation is negative, a negative monotonic constraint is specified using -1.

```
mono_constraints = tuple([int(i) for i in np.sign(data[X + [y]].corr()[y].values[:-1])])
mono_constraints
```

```
(1,
 1,
 1,
 1,
 1,
 1,
 1,
 -1,
 1,
 1,
 -1,
 1,
 1,
 -1,
 -1,
 1,
 -1,
 -1,
 -1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
```

Next step, we passed the mono_constraint to XGBoost when the GBM model is trained.

```
# used to calibrate predictions to mean of y
base_y = y_train_under.mean()

# tuning parameters
params = {
    'objective': 'binary:logistic',             # produces 0-1 probabilities for binary classification
    'booster': 'gbtree',                        # base learner will be decision tree
    'eval_metric': 'auc',                       # stop training based on maximum AUC, AUC always between 0-1
    'eta': 0.08,                                # learning rate
    'subsample': 0.9,                           # use 90% of rows in each decision tree
    'colsample_bytree': 0.9,                    # use 90% of columns in each decision tree
    'max_depth': 15,                            # allow decision trees to grow to depth of 15
    'monotone_constraints':mono_constraints,    # 1 = increasing relationship, -1 = decreasing relationship
    'base_score': base_y,                       # calibrate predictions to mean of y
    'seed': 666                                # set random seed for reproducibility
}

# watchlist is used for early stopping
watchlist = [(dtrain, 'train'), (dtest, 'eval')]

# train model
xgb_model = xgb.train(params,                   # set tuning parameters from above
                      dtrain,                   # training data
                      1000,                     # maximum of 1000 iterations (trees)
                      evals=watchlist,          # use watchlist for early stopping
                      early_stopping_rounds=50, # stop after 50 iterations (trees) without increase in AUC
                      verbose_eval=True)
```

```
[191]   train-auc:0.894062      eval-auc:0.712957
[192]   train-auc:0.894185      eval-auc:0.712879
[193]   train-auc:0.894322      eval-auc:0.712748
[194]   train-auc:0.894439      eval-auc:0.712757
[195]   train-auc:0.894589      eval-auc:0.712797
[196]   train-auc:0.894739      eval-auc:0.712809
[197]   train-auc:0.894867      eval-auc:0.712791
[198]   train-auc:0.895065      eval-auc:0.712861
[199]   train-auc:0.895257      eval-auc:0.712867
[200]   train-auc:0.895323      eval-auc:0.712856
[201]   train-auc:0.895517      eval-auc:0.712861
[202]   train-auc:0.895833      eval-auc:0.71283
[203]   train-auc:0.896066      eval-auc:0.712809
Stopping. Best iteration:
[153]   train-auc:0.885447      eval-auc:0.7142
```

We manually set the algorithm to stop training after 50 iterations without increasing in AUC. In the end the model returned the best AUC on fitting the test dataset is 0.7142.

Obtaining the confusion matrix and reporting the accuracy rate, it's 65.74%.

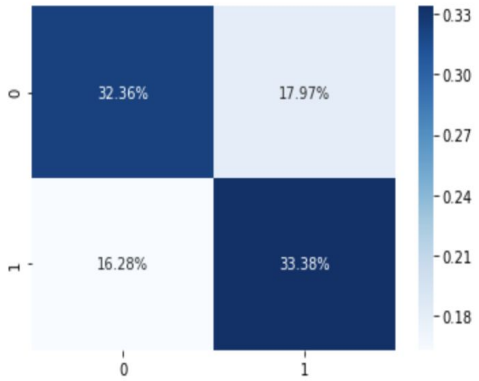

Figure 3.2 Confusion Matrix

4. Summary and Conclusion

4.1. Regression

	RMSE	R^2
Linear Regression	12192.2	0.018
Ridge Regression	8225.8	0.039
Lasso Regression	8226.1	0.039
Poisson Regression	12796.9	N/A

4.2. Classification

	Accuracy	AUC	F1
Naive Bayes	0.633	0.738	0.537
KNN	0.686	0.749	0.675
Logistic Regression	0.718	0.786	0.718
SVM	0.717	0.784	0.717
XGboost	**0.769**	**0.885**	**0.769**

According to the summary, XGboost performs superior to any other algorithms used in this project.

4.3. Overall

After we tried all these models, now it's time to put all the results together and pick the best model. On the left side is a table for regression, as we can see the model with best performance is the Lasso regression. However, the R-square is too low. Maybe it's because the actual relation between these variables and our response is not linear. The right side is our classification results on the training dataset, as you can see the XGboost model outperforms.

There are published articles studying the exact same dataset as ours, we have different ways to process data, we did different features selection and applied different models. We had similar accuracy results as other

existing research, we even have higher AUC results. Thus, we can fairly say our classification models are considered good.

4.4. Conclusion and model interpretation

So as Xgboost is our best model, we tried to interpret our model and give some practical advice to news writers or the news platform. **Figure 2.2** shows top20 important features. The ranking result tells us when news writers want to write popular news, they need to pay attention to features like average number of keywords,publish the news over the weekend or not, what's the publish category.

Then in order to understand how one feature impacts the probability of being popular specifically, we then calculated partial dependence and plotted out the ICE curve for some important variables.

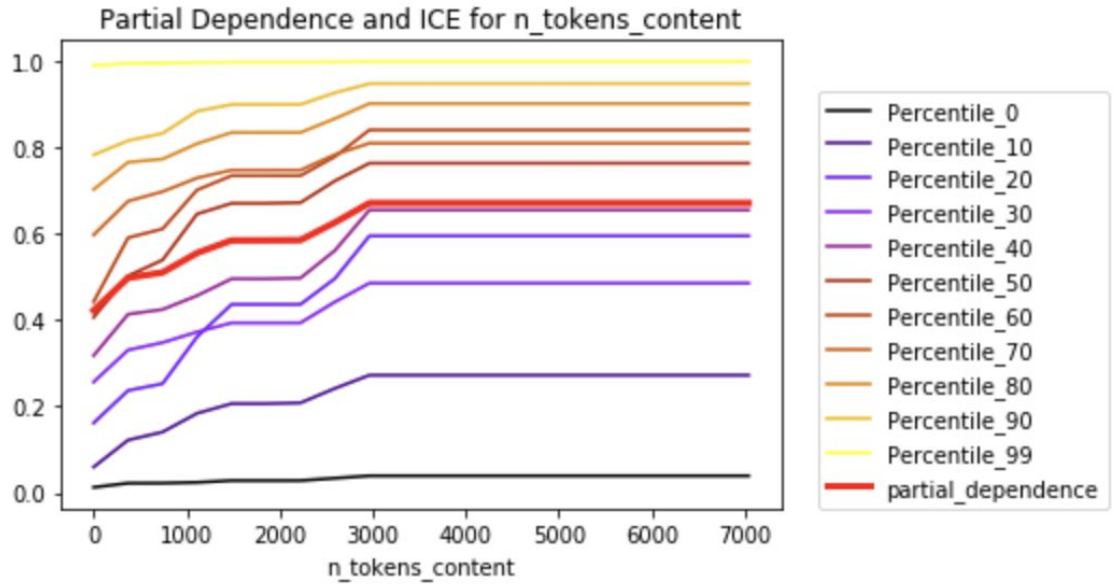

Figure 4.1 Partial dependence and ICE plot

For example, **Figure 4.1** is for the feature called number of words in the content, the y-axis is the probability of being popular, the x-axis is the number of words. We can see that first as the number of words increases, the probability of being popular also increases. This is consistent with our monotonic constraint result. The plot also tells us one more thing that, when the number of words increases from 0 to 3000, the probability

increases significantly but after 3000, the probability remains the same. So we can therefore say that if you want to make your news popular, you need to at least write 3000 words long.

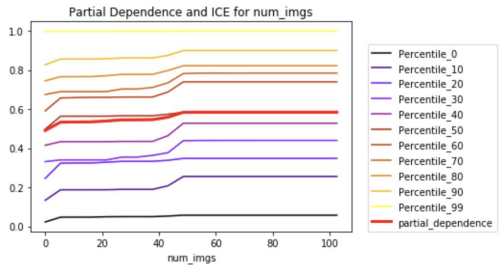

Figure 4.2

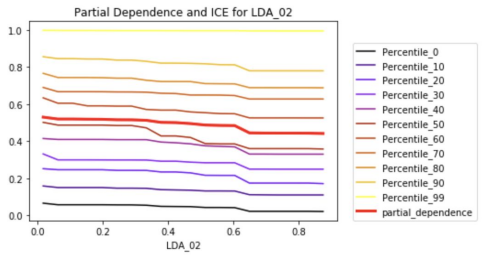

Figure 4.3

Similarly, we can draw some conclusions for other important features, like it's a good choice to include some images in your content but make sure to include around 45 images. When it comes to topic choice for the editor, they may want to avoid news with a closeness score to LDA topic 2 greater than 0.6.