DAT340 - Assignment 3

April 28, 2022

Group PA 47 - Author: Stefano Ribes, ribes@chalmers.se

This Notebook can be viewed online at this link: https://colab.research.google.com/drive/1piorD2pjx2yW4debU0K

# 1 Programming Assignment 3: Text Classification

## 1.1 Setup

[5]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[6]:
```
import os

ASSIGNMENT_ID = 'assignment_3'

data_dir = os.path.join(os.path.abspath(''), 'drive', 'MyDrive')
data_dir = os.path.join(data_dir, 'Colab Notebooks', 'dat340', ASSIGNMENT_ID)
data_dir = os.path.join(data_dir, 'data')
if os.path.exists(data_dir):
    print(f'Directory "{data_dir}" exists')
else:
    print(f'WARNING! Directory "{data_dir}" does not exist!')
```

```
Directory "/content/drive/MyDrive/Colab Notebooks/dat340/assignment_3/data"
exists
```

[7]:
```
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf', 'svg')
```

## 1.2 Cleaning the Data

### 1.2.1 Training Data

Let's start cleaninig the training dataset and resolve conflicting annotations.

First of all, let's read the raw values into lists and convert annotations into integers. Note that there are some empty annotations in the dataset, so I decided to not add their corresponding reviews to the training data.

[8]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

filename = os.path.join(data_dir, 'PA3_train.tsv')
Xtrain = []
Y1train = []
Y2train = []
with open(filename, encoding='utf-8') as f:
    for line in f:
        annotation, review = line.strip().lower().split('\t', maxsplit=1)
        review = review.strip('"')
        a1, a2 = annotation.split('/', maxsplit=1)
        # If any annotation is missing, do not add the corresponding review
        try:
            a1, a2 = int(a1), int(a2)
        except:
            continue
        a1, a2 = int(a1), int(a2)
        Xtrain.append(review)
        Y1train.append(a1)
        Y2train.append(a2)
print(f'len(Xtrain):  {len(Xtrain)}')
print(f'len(Y1train): {len(Y1train)}')
print(f'len(Y2train): {len(Y2train)}')
```

```
len(Xtrain):  7017
len(Y1train): 7017
len(Y2train): 7017
```

The training labels contain values which are not either 0 or 1.

[9]:
```python
np.unique(Y1train, return_counts=True)
```

[9]:
```
(array([-1,  0,  1,  2,  9]), array([ 125, 3243, 3644,    4,    1]))
```

[10]:
```python
np.unique(Y2train, return_counts=True)
```

[10]:
```
(array([0, 1]), array([3373, 3644]))
```

We can see the discording annotations by constructing a dataframe.

[11]:
```
df = pd.DataFrame(list(zip(Y1train, Y2train, Xtrain)), columns=['Annotation 1',␣
↪'Annotation 2', 'Review'])
df = df[df['Annotation 1'] != df['Annotation 2']]
df
```

[11]:
```
      Annotation 1  Annotation 2  \
6                1             0
12               1             0
19              -1             0
45              -1             1
50               0             1
…              …             …
6985             1             0
6993            -1             0
7004             9             1
7007             1             0
7015            -1             0

                                                 Review
6     as a kiwi guy constantly on the hunt for decen…
12    chocolate mousse was with a peppery taste that…
19    restaurant mon ami en las vegas , un verdadero…
45    love love love this place .. its kind of small…
50    hands down the best pizza i’ve ever eaten. and…
…                                                 …
6985  we went there for dinner with of course high e…
6993  some times this place is good today it wasn't …
7004  staff service is good, sushi rice could be try…
7007     often here for lunch. nearly always very good.
7015  the ambiance is ok, the service is slightly sl…

[395 rows x 3 columns]
```

[12]:
```
tmp = df[df['Annotation 1'] != df['Annotation 2']]
print(f'Number of discording annotations: {len(tmp)} ({len(tmp) / len(Xtrain) *␣
↪100:.1f}%)')
```

```
Number of discording annotations: 395 (5.6%)
```

The first step is to threshold positive and negative annotations, assuming that negative values correspond to negative reviews and vice versa.

[13]:
```
# Negative annotations
df.loc[df['Annotation 1'] <= 0, 'Annotation 1'] = 0
df.loc[df['Annotation 2'] <= 0, 'Annotation 2'] = 0
# Positive annotations
```

```
df.loc[df['Annotation 1'] > 0, 'Annotation 1'] = 1
df.loc[df['Annotation 2'] > 0, 'Annotation 2'] = 1
df
```

[13]:
```
      Annotation 1  Annotation 2  \
6                1             0
12               1             0
19               0             0
45               0             1
50               0             1
…              …            …
6985             1             0
6993             0             0
7004             1             1
7007             1             0
7015             0             0

                                                 Review
6     as a kiwi guy constantly on the hunt for decen…
12    chocolate mousse was with a peppery taste that…
19    restaurant mon ami en las vegas , un verdadero…
45    love love love this place .. its kind of small…
50    hands down the best pizza i've ever eaten. and…
…                                                   …
6985  we went there for dinner with of course high e…
6993  some times this place is good today it wasn't …
7004  staff service is good, sushi rice could be try…
7007     often here for lunch. nearly always very good.
7015  the ambiance is ok, the service is slightly sl…

[395 rows x 3 columns]
```

[14]:
```
tmp = df[df['Annotation 1'] != df['Annotation 2']]
print(f'Number of discording annotations: {len(tmp)} ({len(tmp) / len(Xtrain) *␣
 ↪100:.1f}%)')
```

```
Number of discording annotations: 295 (4.2%)
```

The next step is to set to 1 or 0 both conconding reviews, *i.e.* if both positive then set them both to 1.

[15]:
```
# Both positive
df.loc[(df['Annotation 1'] > 0) & (df['Annotation 2'] > 0), 'Annotation 1'] = 1
df.loc[(df['Annotation 1'] > 0) & (df['Annotation 2'] > 0), 'Annotation 2'] = 1
# Both negative
df.loc[(df['Annotation 1'] <= 0) & (df['Annotation 2'] <= 0), 'Annotation 1'] =␣
 ↪0
```

```
df.loc[(df['Annotation 1'] <= 0) & (df['Annotation 2'] <= 0), 'Annotation 2'] =
 ↪0
df
```

[15]:
```
      Annotation 1  Annotation 2  \
6                1             0
12               1             0
19               0             0
45               0             1
50               0             1
…              …           …
6985             1             0
6993             0             0
7004             1             1
7007             1             0
7015             0             0

                                                 Review
6     as a kiwi guy constantly on the hunt for decen…
12    chocolate mousse was with a peppery taste that…
19    restaurant mon ami en las vegas , un verdadero…
45    love love love this place .. its kind of small…
50    hands down the best pizza i’ve ever eaten. and…
…                                                   …
6985  we went there for dinner with of course high e…
6993  some times this place is good today it wasn't …
7004  staff service is good, sushi rice could be try…
7007     often here for lunch. nearly always very good.
7015  the ambiance is ok, the service is slightly sl…

[395 rows x 3 columns]
```

Now we should be seeing a lower or equal amount of discording annotations.

[16]:
```
tmp = df[df['Annotation 1'] != df['Annotation 2']]
print(f'Number of discording annotations: {len(tmp)} ({len(tmp) / len(Xtrain) *
 ↪100:.1f}%)')
```

```
Number of discording annotations: 295 (4.2%)
```

For the remaining discording annotations, a possible approach could be to either manually edit them or to use a classifier to properly label them.

Since they account for only 4.2% of the reviews, I decided to remove them from the training set.

[17]:
```
orig_df = pd.DataFrame(list(zip(Y1train, Y2train, Xtrain)),
 ↪columns=['Annotation 1', 'Annotation 2', 'Review'])
cleaned_df = orig_df.drop(df.index)
```

```
cleaned_df
```

[17]:
```
      Annotation 1  Annotation 2  \
0                0             0
1                1             1
2                0             0
3                0             0
4                0             0
…              …           …
7011             1             1
7012             0             0
7013             0             0
7014             1             1
7016             1             1

                                                 Review
0     ordered my food the hole meal looked dead. pla…
1     we stopped her whilst walking in the haga area…
2     bad experience, on 23/03/19 myself and my part…
3     extremely underwhelming experience here last n…
4     waited 30 minutes to get a table…that was ok. …
…                                                 …
7011  we recently dined at ma cuisine, and enjoyed e…
7012                              bad service, stay away
7013  old school, but not always in a good way. lots…
7014  top 5 allergen free restaurant and the food is…
7016  the food was yummy and the drinks excellent. w…

[6622 rows x 3 columns]
```

Finally, `Xtrain` and `Ytrain` can be extracted by the dataframe columns.

[18]:
```python
Xtrain = list(cleaned_df['Review'])
Ytrain = list(cleaned_df['Annotation 1']) # Either ann1 or ann2 would work now
[(x, y) for x, y in zip(Xtrain[:3], Ytrain[:3])]
```

[18]:
```
[('ordered my food the hole meal looked dead. plain cold and looked horrible the
woman shouted at me as i complained about it and threatened to throw a chair at
me',
  0),
 ('we stopped her whilst walking in the haga area. the cafe is well recommended.
good service and we enjoyed our teas and a cinamon roll. the latter was large
but so good that between us we finished it! recommended stop off.',
  1),
 ('bad experience, on 23/03/19 myself and my partner arrived at 20.00 and were
promptly sent to the bar and told it would be roughly a 30 minute wait by the
blonde lady at the front desk, she was impolite to start. after 50 minutes(and
```

```
when we noticed the couple that arrived after us were seated before us) i went
to talk again and she said they will be as quick as possible. another 15 minutes
passed during which another couple who arrived after us were seated. this
happened a third time and then we left after paying for our drinks feeling like
we were ignored. the bar tenders were fantastic however the floor greeting staff
were rude and self righteous. won't be going back. bad experience.',
  0)]
```

[19]:
```python
np.unique(Ytrain, return_counts=True)
```

[19]:
```
(array([0, 1]), array([3126, 3496]))
```

#### 1.2.2 Test Data

[20]:
```python
filename = os.path.join(data_dir, 'PA3_test_clean.tsv')
Xtest = []
Ytest = []
with open(filename, encoding='utf-8') as f:
    for line in f:
        annotation, review = line.strip().lower().split('\t', maxsplit=1)
        review = review.strip('"')
        Xtest.append(review)
        Ytest.append(int(annotation))
```

[21]:
```python
np.unique(Ytest, return_counts=True)
```

[21]:
```
(array([0, 1]), array([866, 886]))
```

Guide on text feature extraction.

### 1.3 Vectorizer

#### 1.3.1 Tokenizer

[22]:
```python
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer

nltk.download('punkt')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[22]: True

Ideally, a custom Tokenizer might be useful to group together words with the same stemm, *i.e.* "cooking", "cooked", "cooker" = "cook", potentially reducing the amount of features to consider.

[23]:
```
class LemmaTokenizer(object):
    def __init__(self):
        self.wnl = WordNetLemmatizer()

    def __call__(self, articles):
        return [self.wnl.lemmatize(t) for t in word_tokenize(articles)]
```

Unfortunately, the above tokenizer seems to be performing very badly (in the tests performed below), and I don't have time to fix it. Hence, I'm dropping it from the analysis.

### 1.3.2 Tfidf Vectorizer

Before setting the parameter grid for the Vectorizer, let's play a bit with its parameters to see which features are extracted.

[25]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(strip_accents='unicode',
                             stop_words='english',
                             min_df=0.003,
                            #  max_features=1000,
                             ngram_range=(1, 2),
                             lowercase=True)
vectorizer.fit(Xtrain)
features_names = vectorizer.get_feature_names_out()
# Logging
table_data = []
for elem in features_names:
    table_data.append(elem)
pd.DataFrame(table_data, columns=['Feature Name'])
```

[25]:

| | Feature Name |
|---|---|
| 0 | 00 |
| 1 | 10 |
| 2 | 10 10 |
| 3 | 10 minutes |
| 4 | 100 |
| … | … |
| 1201 | yes |
| 1202 | york |
| 1203 | young |
| 1204 | yummy |

```
1205         zero

[1206 rows x 1 columns]
```

[26]:

```
table_data = []
for elem in vectorizer.get_stop_words():
    table_data.append(elem)
pd.DataFrame(table_data, columns=['Stop Words'])
```

[26]:

```
    Stop Words
0         will
1         side
2     yourself
3         whom
4        noone
..         ...
313  meanwhile
314    further
315         us
316     though
317         re

[318 rows x 1 columns]
```

Finally, set a TfIdf Vectorizer with a parameter grid common to all classifiers.

[27]:

```
tfidf_parameters = {
    # 'tfidf__tokenizer': (None, LemmaTokenizer()), # Not working properly
    # 'tfidf__max_df': (0.4, 1.0), # No difference in the tests
    # 'tfidf__min_df': (1.0, 0.0001), # 1.0: ~100k, 0.001: ~3k, 0.005: ~1.2k
    'tfidf__ngram_range': ((1, 1), (1, 2)),  # Unigrams or bigrams
    'tfidf__use_idf': (True, False),
    'tfidf__max_features': (None, 1000, 5000),
    # 'tfidf__norm': ('l1', 'l2')
}

def add_parameters(params_old, params_in):
    params_new = params_old.copy()
    for k in params_in.keys():
        params_new[k] = params_in[k]
    return params_new

tfidf = TfidfVectorizer(strip_accents='unicode',
                        stop_words='english',
                        lowercase=True)
tfidf
```

[27]: TfidfVectorizer(stop_words='english', strip_accents='unicode')

[31]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

tfidf_tester = GridSearchCV(Pipeline([('tfidf', tfidf)]), tfidf_parameters,
                            scoring='accuracy',
                            cv=5, error_score=0, n_jobs=-1, verbose=3)
# tfidf_tester.fit(Xtrain, Ytrain)
```

## 1.4 Models

[32]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

# Features preprocessing
from sklearn.preprocessing import MaxAbsScaler # Unused...
# Linear classifiers
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.svm import LinearSVC
# Bernoulli Model (requires binary vectorizer)
from sklearn.naive_bayes import BernoulliNB
# Tree Classifiers
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# Neural network classifier (will take longer time to train)
from sklearn.neural_network import MLPClassifier
scoring_metrics = ['accuracy', 'roc_auc']
models = {}
```

### 1.4.1 Perceptron (Baseline)

[33]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', Perceptron())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__penalty': ('l1', 'l2'),
    'clf__alpha': (0.0001, 0.001),
```

```
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['Perceptron'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.2 Bernoulli

[34]:
```
# Setup the pipeline with the vectorizer and the classifier
# NOTE: With Bernoulli, the vectorizer must produce a binarized output.
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(strip_accents='unicode', stop_words='english',
                              binary=True, lowercase=True)),
    ('clf', BernoulliNB())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {

}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['BernoulliNB'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.3 Decision Tree

[35]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', DecisionTreeClassifier())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__max_depth': (32, 128),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['DecisionTreeClassifier'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.4 Random Forest

[36]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', RandomForestClassifier())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__n_estimators': (16, 64, 100),
    'clf__max_depth': (32, 128),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['RandomForestClassifier'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.5 LogisticRegression

[37]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', LogisticRegression())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    # 'clf__penalty': ('l1', 'l2'), # Not all sovers support L1
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['LogisticRegression'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.6 SGDClassifier

[38]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', SGDClassifier())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__penalty': ('l1', 'l2'),
```

```python
    'clf__loss': ('hinge', 'modified_huber'),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['SGDClassifier'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.7 LinearSVC

[39]:
```python
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', LinearSVC())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__penalty': ('l1', 'l2'),
    'clf__C': (1.0, 0.8),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['LinearSVC'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.8 Ridge Classifier

[40]:
```python
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', RidgeClassifier())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__alpha': (0.9, 0.8),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['RidgeClassifier'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

### 1.4.9 Multi-layer Perceptron

[41]:
```
# Setup the pipeline with the vectorizer and the classifier
pipeline = Pipeline([
    ('tfidf', tfidf),
    ('clf', MLPClassifier())
])
# Add the classifier configurations to the grid of parameters
clf_parameters = {
    'clf__hidden_layer_sizes': ((128,), (128, 128,), (256, 256, 256,)),
}
parameters = add_parameters(tfidf_parameters, clf_parameters)
# Add the grid-search to the list of models
models['MLPClassifier'] = GridSearchCV(pipeline, parameters,
                                    scoring=scoring_metrics, refit='accuracy',
                                    cv=5, error_score=0, n_jobs=-1, verbose=3)
```

## 1.5 Cross-Validation and Training

[42]:
```
from sklearn.utils.validation import check_is_fitted
from joblib import dump, load

SCORE_THRESHOLD = 0.80

scores = {}
for model_type in models.keys():
    modelfile = os.path.join(data_dir, f'{model_type}.joblib')
    # Check if model exists on disk and eventually load it.
    if os.path.exists(modelfile):
        models[model_type] = load(modelfile)
        model_saved = True
    else:
        model_saved = False
    # Check if model is fitted and if so, that its score is above a threshold.
    try:
        check_is_fitted(models[model_type])
        model_trained = models[model_type].score(Xtest, Ytest) > SCORE_THRESHOLD
    except:
        model_trained = False
    # Fit if the performance of the model are low or if it has not been loaded
    # from disk.
    if not model_trained or not model_saved:
        models[model_type].fit(Xtrain, Ytrain)
        dump(models[model_type], modelfile)
    scores[model_type] = models[model_type].best_score_
```

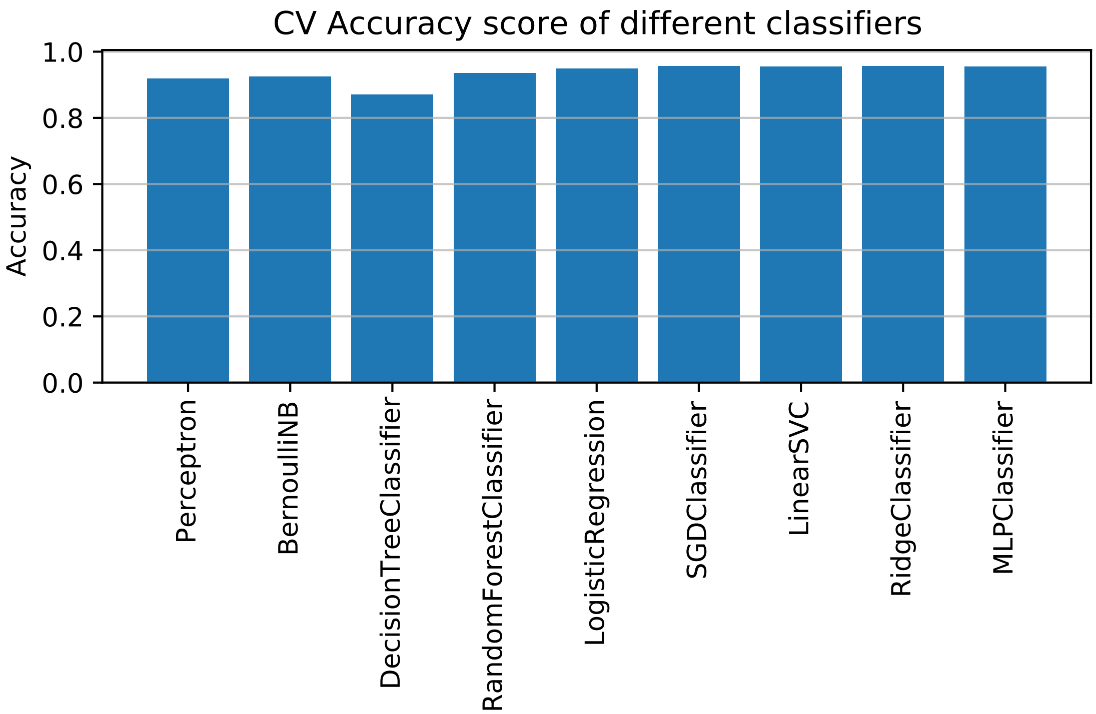
```
    print(f'{model_type} model reached CV score of: {models[model_type].
 ↪best_score_:.3f}')
```

```
Perceptron model reached CV score of: 0.919
BernoulliNB model reached CV score of: 0.925
DecisionTreeClassifier model reached CV score of: 0.871
RandomForestClassifier model reached CV score of: 0.935
LogisticRegression model reached CV score of: 0.949
SGDClassifier model reached CV score of: 0.956
LinearSVC model reached CV score of: 0.956
RidgeClassifier model reached CV score of: 0.957
MLPClassifier model reached CV score of: 0.955
```

[63]:

```
import matplotlib.pyplot as plt

for model_type in models.keys():
    print(f'{model_type} cross-validation best score: {models[model_type].
 ↪best_score_:.3f}')

linspace = [x for x in range(len(models.keys()))]
plt.bar(linspace, scores.values())
plt.xticks(linspace, [f'{m}' for m in models.keys()], rotation=90)
plt.grid(which='both', axis='y', alpha=0.7, zorder=1)
plt.ylabel('Accuracy')
plt.title('CV Accuracy score of different classifiers')
plt.tight_layout()
plt.savefig(os.path.join(data_dir, f'cv_accuracy.pdf'))
plt.show()
```

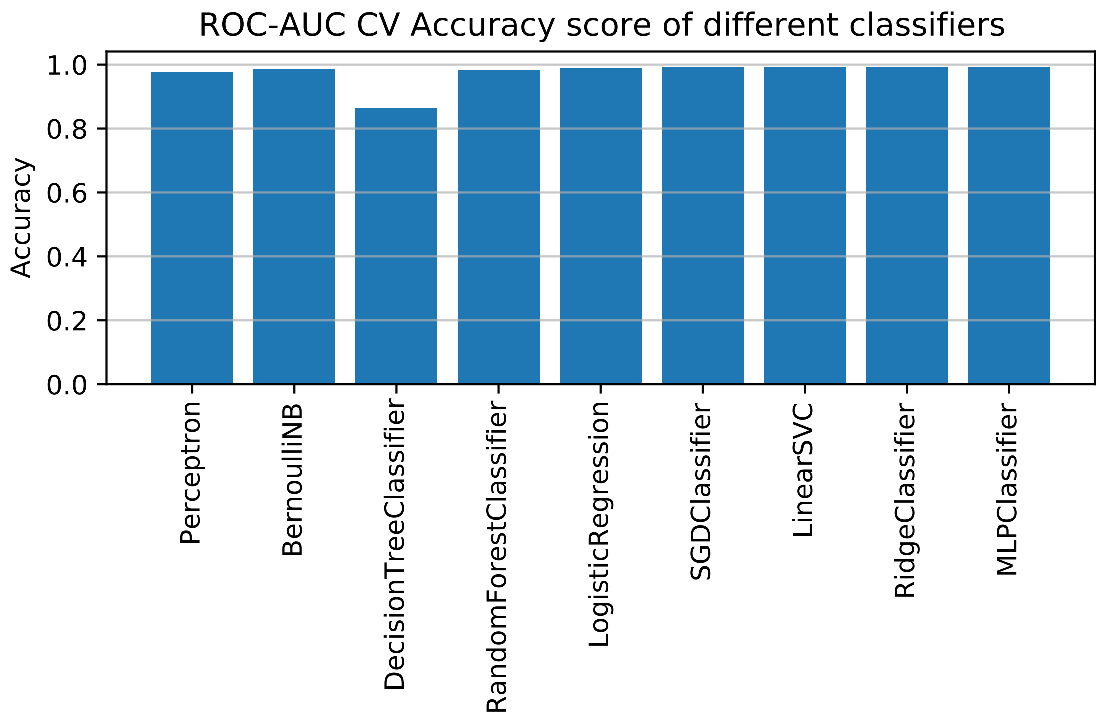
```
Perceptron cross-validation best score: 0.919
BernoulliNB cross-validation best score: 0.925
DecisionTreeClassifier cross-validation best score: 0.871
RandomForestClassifier cross-validation best score: 0.935
LogisticRegression cross-validation best score: 0.949
SGDClassifier cross-validation best score: 0.956
LinearSVC cross-validation best score: 0.956
RidgeClassifier cross-validation best score: 0.957
MLPClassifier cross-validation best score: 0.955
```




[64]:
```python
def get_roc_auc(model_type):
    i = np.argmin(models[model_type].cv_results_['rank_test_roc_auc'])
    return models[model_type].cv_results_['mean_test_roc_auc'][i]

roc_auc_scores = {}
for model_type in models.keys():
    roc_auc_scores[model_type] = get_roc_auc(model_type)
    print(f'{model_type} ROC-AUC CV score: {get_roc_auc(model_type):.3f}')

linspace = [x for x in range(len(models.keys()))]
plt.bar(linspace, roc_auc_scores.values())
plt.xticks(linspace, [f'{m}' for m in models.keys()], rotation=90)
plt.grid(which='both', axis='y', alpha=0.7, zorder=1)
plt.ylabel('Accuracy')
plt.title('ROC-AUC CV Accuracy score of different classifiers')
plt.savefig(os.path.join(data_dir, f'cv_roc_auc.pdf'))
plt.tight_layout()
plt.show()
```

```
Perceptron ROC-AUC CV score: 0.976
BernoulliNB ROC-AUC CV score: 0.985
DecisionTreeClassifier ROC-AUC CV score: 0.862
RandomForestClassifier ROC-AUC CV score: 0.984
LogisticRegression ROC-AUC CV score: 0.988
```

```
SGDClassifier ROC-AUC CV score: 0.990
LinearSVC ROC-AUC CV score: 0.991
RidgeClassifier ROC-AUC CV score: 0.991
MLPClassifier ROC-AUC CV score: 0.991
```




[45]:

```python
table_data = []
for model_type in models.keys():
    vectorizer_params = models[model_type].best_estimator_.steps[0][1].
 ↪get_params()
    table_data.append(
        (model_type,
         vectorizer_params['use_idf'],
         'Unigrams' if vectorizer_params['ngram_range'] == (1,1) else 'Bigrams',
         'All' if vectorizer_params['max_features'] == None else␣
 ↪vectorizer_params['max_features'],
         scores[model_type]
        )
    )
pd.DataFrame(table_data, columns=['Classifier', 'Use idf', 'Uni/Bi-grams', 'Max␣
 ↪Features', 'CV Accuracy'])
```

[45]:

| | Classifier | Use idf | Uni/Bi-grams | Max Features | CV Accuracy |
|---|---|---|---|---|---|
| 0 | Perceptron | False | Bigrams | 5000 | 0.918756 |
| 1 | BernoulliNB | True | Bigrams | 5000 | 0.925097 |

| | | | | | |
|---|---|---|---|---|---|
| 2 | DecisionTreeClassifier | True | Unigrams | 1000 | 0.870584 |
| 3 | RandomForestClassifier | False | Bigrams | All | 0.935368 |
| 4 | LogisticRegression | True | Unigrams | All | 0.948958 |
| 5 | SGDClassifier | True | Bigrams | All | 0.956206 |
| 6 | LinearSVC | True | Bigrams | All | 0.955602 |
| 7 | RidgeClassifier | True | Bigrams | All | 0.957112 |
| 8 | MLPClassifier | False | Bigrams | All | 0.954696 |

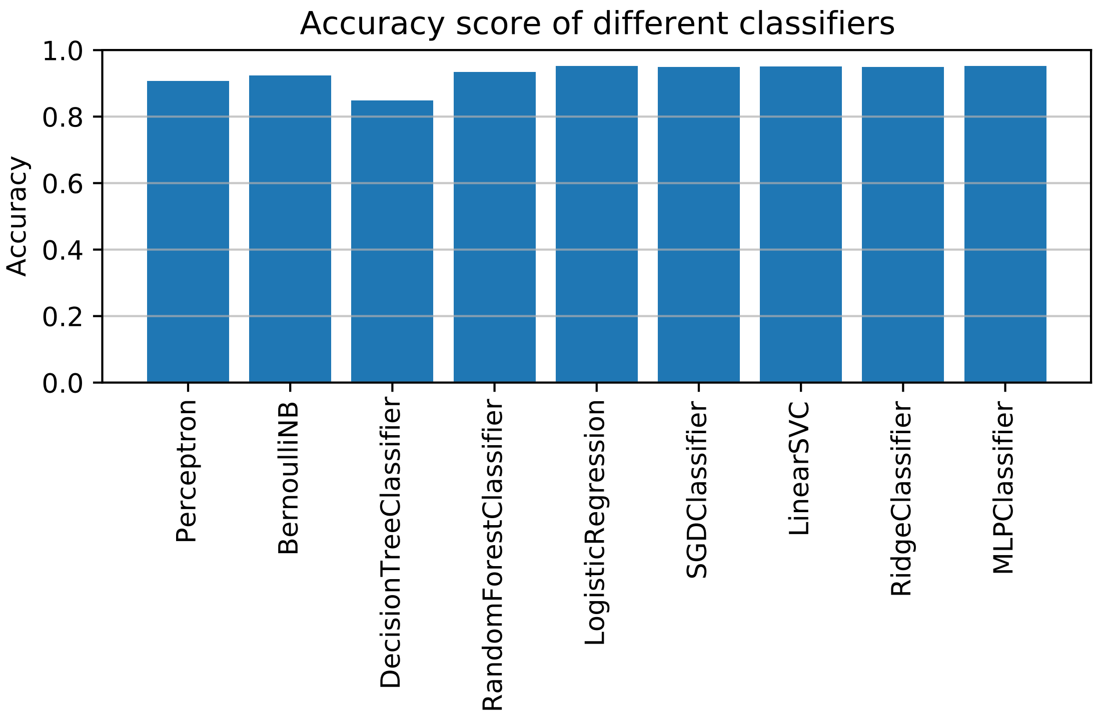
Most of the best classifiers exploit a Vectorizer with all the available features. I suspect I might find a sweet spot if I increase the number of max features allowed.

## 1.6 Evaluation

### 1.6.1 Accuracy

[66]:
```
eval_scores = {}
for model_type in models.keys():
    eval_scores[model_type] = models[model_type].score(Xtest, Ytest)
    print(f'{model_type} score: {eval_scores[model_type]:.3f}')

linspace = [x for x in range(len(models.keys()))]
plt.bar(linspace, eval_scores.values())
plt.xticks(linspace, [f'{m}' for m in models.keys()], rotation=90)
plt.grid(which='both', axis='y', alpha=0.7, zorder=1)
plt.ylabel('Accuracy')
plt.title('Accuracy score of different classifiers')
plt.tight_layout()
plt.savefig(os.path.join(data_dir, f'eval_accuracy.pdf'))
plt.show()
```

```
Perceptron score: 0.906
BernoulliNB score: 0.924
DecisionTreeClassifier score: 0.848
RandomForestClassifier score: 0.934
LogisticRegression score: 0.951
SGDClassifier score: 0.949
LinearSVC score: 0.950
RidgeClassifier score: 0.949
MLPClassifier score: 0.953
```




[47]:
```
table_data = []
for model_type in models.keys():
    vectorizer_params = models[model_type].best_estimator_.steps[0][1].
 ↪get_params()
    table_data.append(
        (model_type,
         vectorizer_params['use_idf'],
         'Unigrams' if vectorizer_params['ngram_range'] == (1,1) else 'Bigrams',
         'All' if vectorizer_params['max_features'] == None else␣
 ↪vectorizer_params['max_features'],
         eval_scores[model_type]
        )
    )
pd.DataFrame(table_data, columns=['Classifier', 'Use idf', 'Uni/Bi-grams', 'Max␣
 ↪Features', 'Eval Accuracy'])
```

[47]:

| | Classifier | Use idf | Uni/Bi-grams | Max Features | Eval Accuracy |
|---|---|---|---|---|---|
| 0 | Perceptron | False | Bigrams | 5000 | 0.906393 |
| 1 | BernoulliNB | True | Bigrams | 5000 | 0.923516 |
| 2 | DecisionTreeClassifier | True | Unigrams | 1000 | 0.848174 |
| 3 | RandomForestClassifier | False | Bigrams | All | 0.933790 |
| 4 | LogisticRegression | True | Unigrams | All | 0.951484 |
| 5 | SGDClassifier | True | Bigrams | All | 0.948630 |

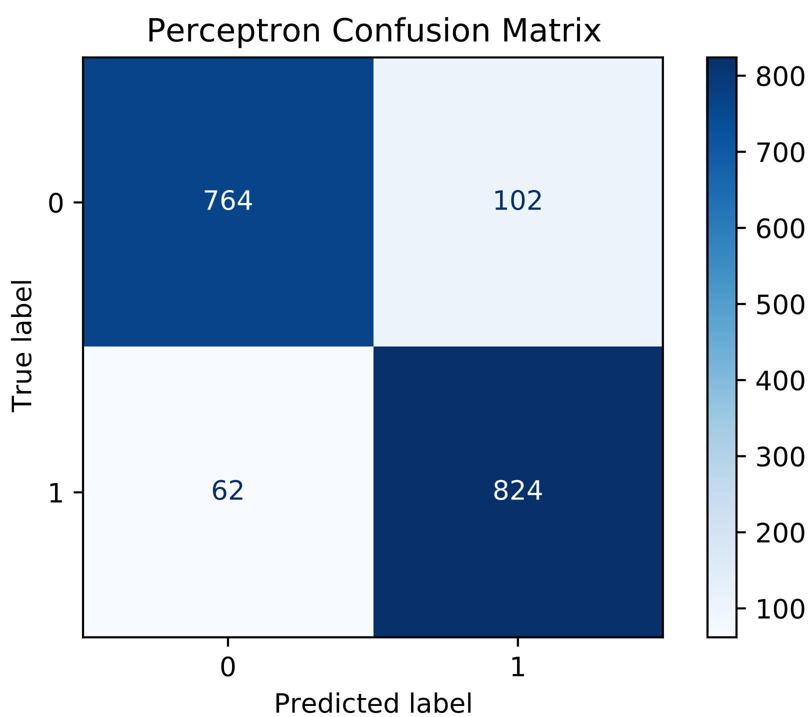
```
6              LinearSVC    True      Bigrams          All       0.950342
7        RidgeClassifier    True      Bigrams          All       0.948630
8          MLPClassifier   False      Bigrams          All       0.952626
```

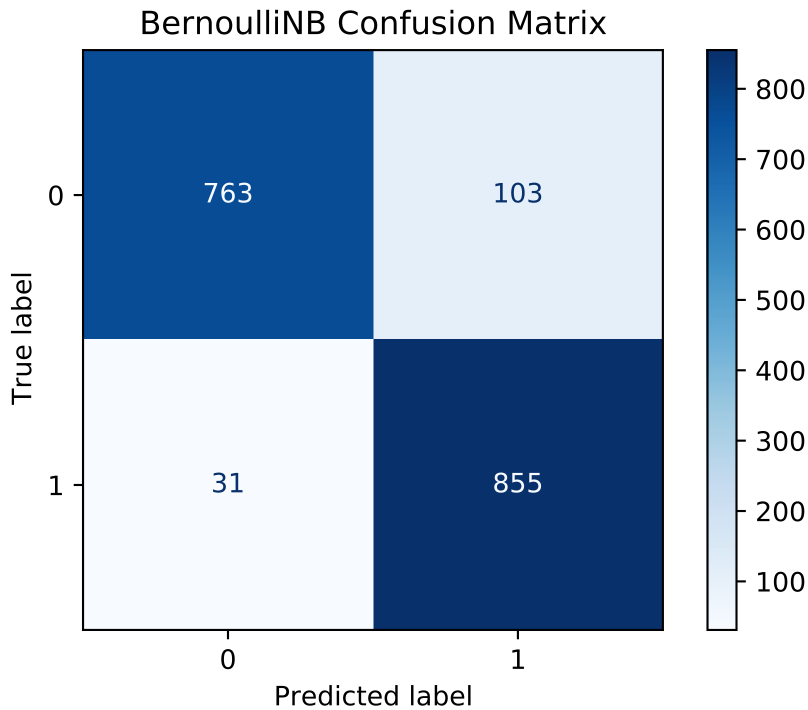
[60]:
```
models['MLPClassifier'].best_estimator_
```

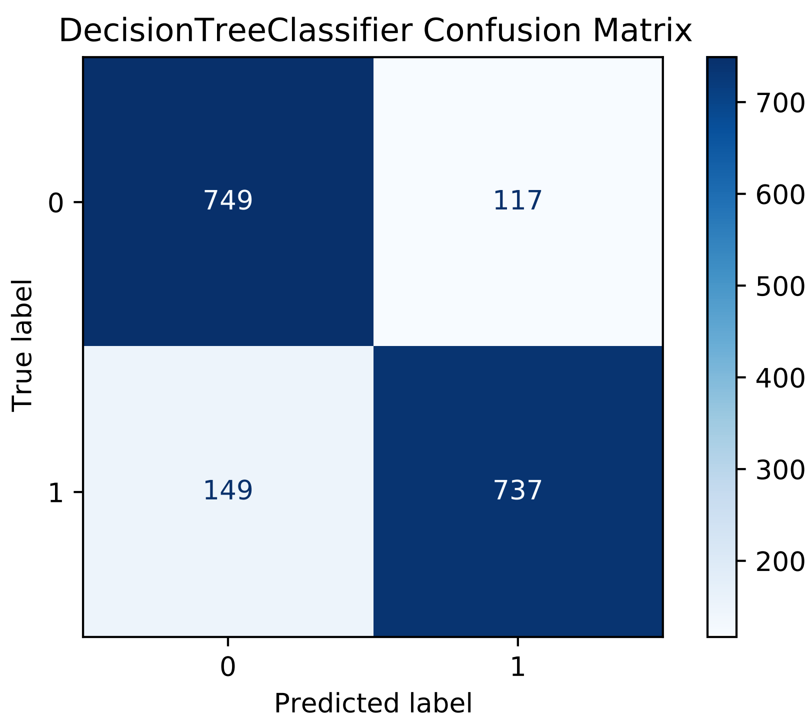
[60]:
```
Pipeline(steps=[('tfidf',
                 TfidfVectorizer(ngram_range=(1, 2), stop_words='english',
                                 strip_accents='unicode', use_idf=False)),
                ('clf', MLPClassifier(hidden_layer_sizes=(256, 256, 256)))])
```

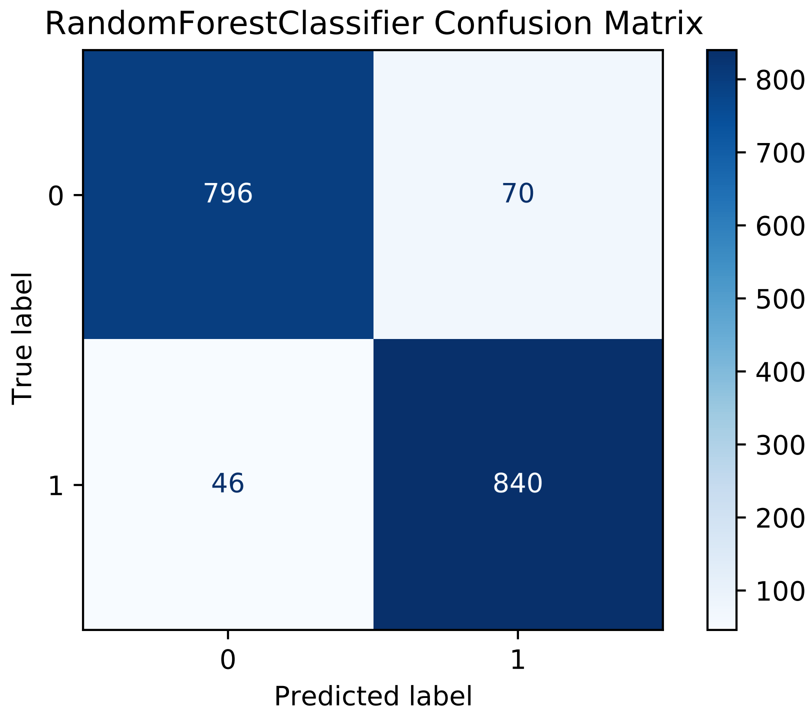
### 1.6.2 Confusion Matrix

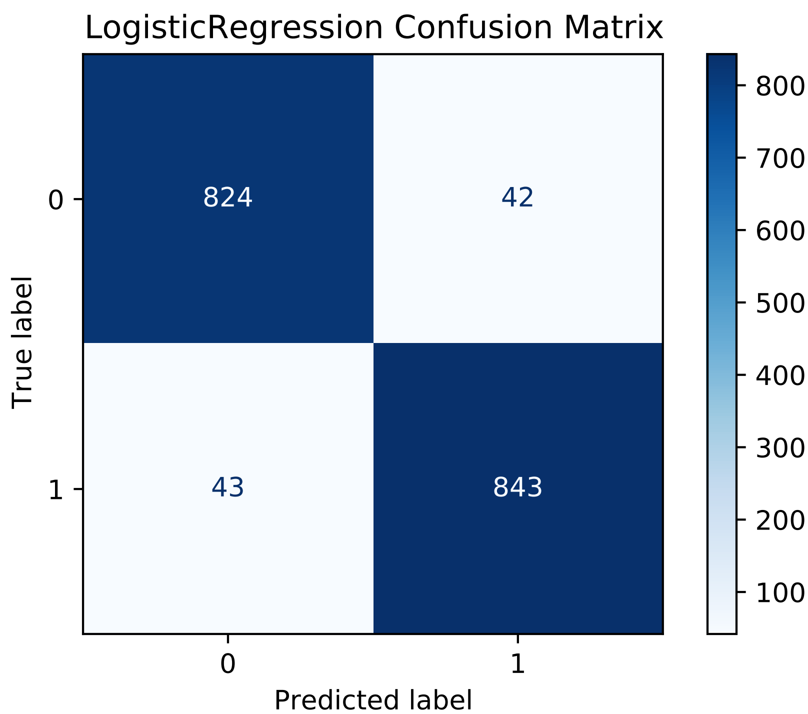
[57]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

for model_type in models.keys():
    disp = ConfusionMatrixDisplay.from_estimator(models[model_type],
                                                 Xtest, Ytest,
                                                 ␣
 ↪display_labels=models[model_type].classes_,
                                                 cmap=plt.cm.Blues)
    disp.ax_.set_title(f'{model_type} Confusion Matrix')
    plt.savefig(os.path.join(data_dir, f'conf_mat_{model_type}.pdf'))
plt.show()
```

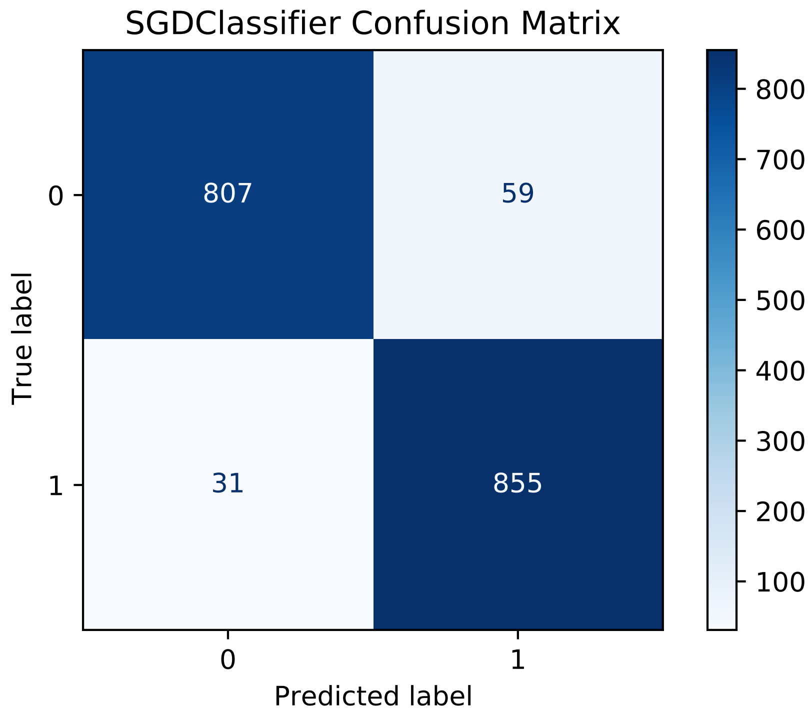


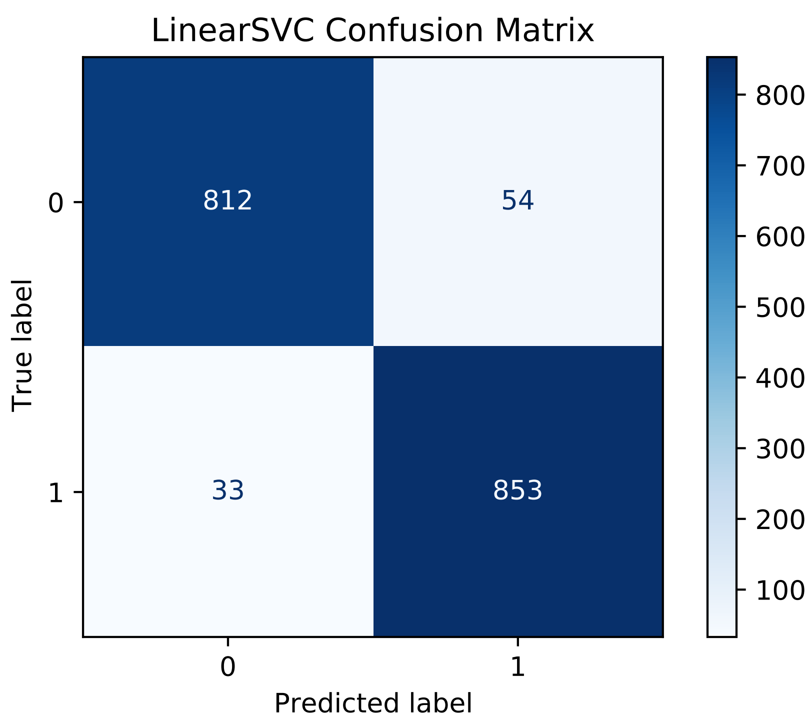

BernoulliNB Confusion Matrix
763
103
31
855
True label
Predicted label
0
1
0
1
800
700
600
500
400
300
200
100

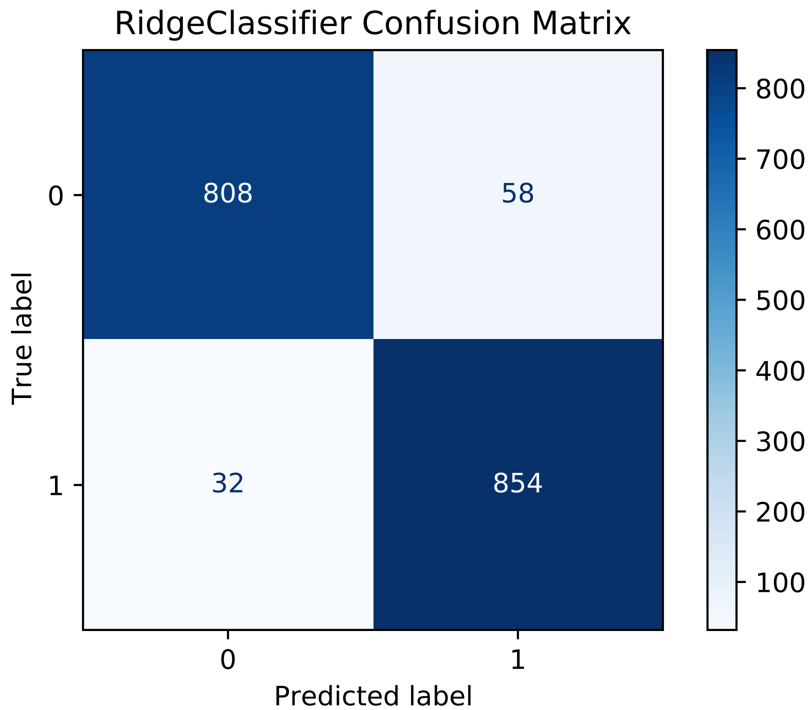

DecisionTreeClassifier Confusion Matrix
749
117
149
737
True label
Predicted label
0
1
0
1
700
600
500
400
300
200

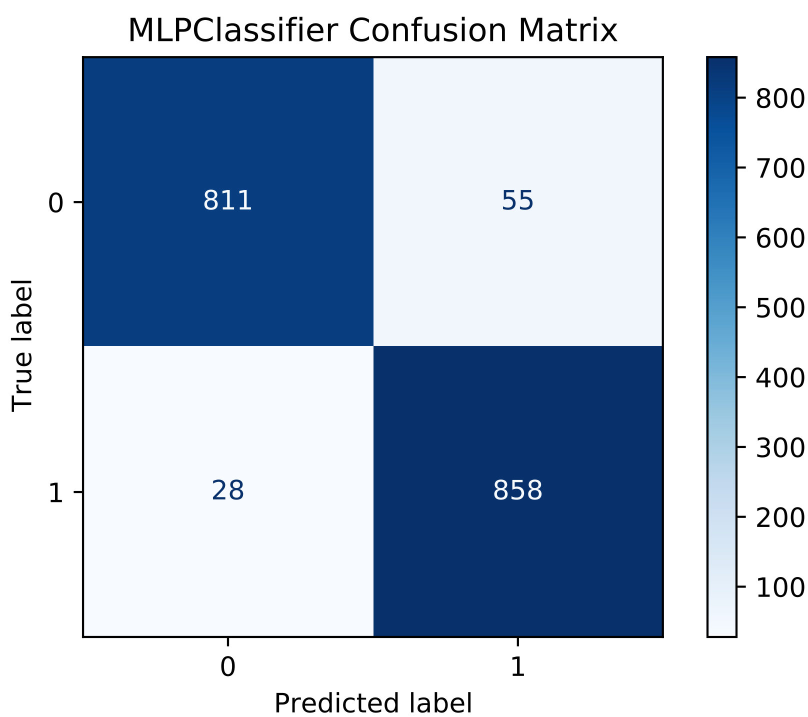

RandomForestClassifier Confusion Matrix
796
70
46
840
True label
Predicted label
0
1
0
1
800
700
600
500
400
300
200
100


LogisticRegression Confusion Matrix
824
42
43
843
True label
Predicted label
0
1
0
1
800
700
600
500
400
300
200
100


SGDClassifier Confusion Matrix
True label
Predicted label
0
1
0
1
807
59
31
855
800
700
600
500
400
300
200
100


LinearSVC Confusion Matrix
True label
Predicted label
0
1
0
1
812
54
33
853
800
700
600
500
400
300
200
100


RidgeClassifier Confusion Matrix
True label
Predicted label
0
1
808
58
32
854
800
700
600
500
400
300
200
100


MLPClassifier Confusion Matrix
True label
Predicted label
0
1
811
55
28
858
800
700
600
500
400
300
200
100

### 1.6.3 Mislabeled Reviews

Let's look at some of the errors the baseline and other classifiers make.

NOTE: Confidence is only available for models that output a probability distribution such as the logistic regression, *e.g.* not the Perceptron model.

[49]:
```
predictions = models['Perceptron'].predict(Xtest)
display_cnt = 0
table_data = []
for i, (y, y_pred) in enumerate(zip(Ytest, predictions)):
    if y_pred != y and display_cnt < 10:
        # print(f'{i:3d}) pred/true/review: {y_pred}/{y} -- "{Xtest[i]}"')
        display_cnt += 1
        table_data.append((y, y_pred, Xtest[i], i))
pd.DataFrame(table_data, columns=['True Label', 'Predicted Label', 'Review',␣
 ↪'Review ID'])
```

[49]:
```
   True Label  Predicted Label  \
0           0                1
1           0                1
2           0                1
3           0                1
4           0                1
5           0                1
6           1                0
7           1                0
8           0                1
9           0                1

                                              Review  Review ID
0  desert was great, starter and main (steak) wou…         11
1  ervice very slow and disjointed and not very f…         12
2  no real atmosphere and definitely not very fre…         42
3                no reasonably priced set lunch menu.         43
4                                    not recommended         44
5                         ood was bland service not great         45
6      outstanding customer service and gorgeous food!         46
7                         sticky fingers is the greatest!         64
8                     the envrionment is not clean enough         69
9                               the food is not fresh.         71
```

[50]:
```
predictions = models['DecisionTreeClassifier'].predict(Xtest)
probs = models['DecisionTreeClassifier'].predict_proba(Xtest)
```

```
display_cnt = 0
table_data = []
for i, (y, y_pred) in enumerate(zip(Ytest, predictions)):
    if y_pred != y and display_cnt < 10:
        # print(f'{i:3d}) pred/true/confidence/review: {y_pred}/{y}/
 ↪{max(probs[i]):.2f} -- "{Xtest[i]}"')
        table_data.append((y, y_pred, max(probs[i]), Xtest[i], i))
        display_cnt += 1
pd.DataFrame(table_data, columns=['True Label', 'Predicted Label',␣
 ↪'Confidence', 'Review', 'Review ID'])
```

[50]:

```
   True Label  Predicted Label  Confidence  \
0           1                0    1.000000
1           0                1    0.992330
2           1                0    0.924347
3           1                0    1.000000
4           1                0    0.924347
5           1                0    0.924347
6           0                1    0.992330
7           1                0    0.924347
8           1                0    0.500000
9           1                0    1.000000

                                              Review  Review ID
0                            a wonderful experience!          1
1  desert was great, starter and main (steak) wou…         11
2  food is better than our expectation. would lik…         19
3  i felt like i was in good hands and was able t…         30
4  i would definitely feel comfortable eating her…         32
5  incredible food, and visual presentation, the …         34
6                    ood was bland service not great          45
7                    sticky fingers is the greatest!          64
8                                         very good.          89
9  waiter was excellent. manager, chef, kitchen a…         91
```

The 11th review is mislabeled as a good comment despite being a negative one. I believe that the word ”great” fools the model into believing it’s a positive review, while instead referring to just one food, not to the overall restaurant experience.

[72]:

```
test_review = 44
table_data = []
for model_type in models.keys():
    y_pred = models[model_type].predict([Xtest[test_review]])[0]
    try:
        confidence = max(models[model_type].
 ↪predict_proba([Xtest[test_review]])[0])
    except:
```

```
        confidence = 'N.A.'
    table_data.append((model_type, Ytest[test_review], y_pred, confidence,␣
 ↪Xtest[test_review]))
pd.DataFrame(table_data, columns=['Classifier', 'True Label', 'Predicted␣
 ↪Label', 'Confidence', 'Review'])
```

[72]:

```
               Classifier  True Label  Predicted Label Confidence  \
0              Perceptron           0                1       N.A.
1             BernoulliNB           0                1   0.992279
2  DecisionTreeClassifier           0                0   0.924347
3  RandomForestClassifier           0                0   0.532642
4      LogisticRegression           0                1   0.710735
5           SGDClassifier           0                1       N.A.
6               LinearSVC           0                1       N.A.
7         RidgeClassifier           0                1       N.A.
8           MLPClassifier           0                0   0.965501

            Review
0  not recommended
1  not recommended
2  not recommended
3  not recommended
4  not recommended
5  not recommended
6  not recommended
7  not recommended
8  not recommended
```

### 1.6.4 Feature Importance

Let's anylize the feature importance by analyzing the Random Forest classifier.

[52]:

```
vectorizer = models['RandomForestClassifier'].best_estimator_.steps[0][1]
randforest = models['RandomForestClassifier'].best_estimator_.steps[1][1]
# Get indeces of sorted importance values, then the sorted feature names
sorted_idx = (-randforest.feature_importances_).argsort()
features_names = vectorizer.get_feature_names_out()[sorted_idx]
features_vals = randforest.feature_importances_[sorted_idx]
# Logging
table_data = []
for elem in zip(features_names, features_vals):
    table_data.append(elem)
pd.DataFrame(table_data, columns=['Feature Name', 'Importance'])
```

[52]:

```
        Feature Name  Importance
0              great    0.022768
```

```
1          delicious    0.014705
2           friendly    0.014235
3            amazing    0.011841
4              worst    0.010895
…                  …           …
99577       house ny    0.000000
99578     house nice    0.000000
99579  house negroni    0.000000
99580    house place    0.000000
99581      50 person    0.000000

[99582 rows x 2 columns]
```

Not surprisingly, in the top 5 features we find 1-gram adjectives: "great", "delicious", "amazing", "excellent", "worst".

I expected negated verbs like "don", "wasn" and "didn" to be ranked higher, but they are still in the top-50 positions.

At the bottom there are very "strong" bigrams such as "disgusting behaviour", but apparently they are also very rare in the documents and so ranked low.

---

For the Decision Tree classifier, despite exploiting lesser number of features (1000 versus ~10000), the most important features remain the positive adjectives.

At the bottom are lesser important features like food names like "gyoza", which is a reasonable and intuitive assumption.

[53]:
```python
vectorizer = models['DecisionTreeClassifier'].best_estimator_.steps[0][1]
randforest = models['DecisionTreeClassifier'].best_estimator_.steps[1][1]
# Get indeces of sorted importance values, then the sorted feature names
sorted_idx = (-randforest.feature_importances_).argsort()
features_names = vectorizer.get_feature_names_out()[sorted_idx]
features_vals = randforest.feature_importances_[sorted_idx]
# Logging
table_data = []
for elem in zip(features_names, features_vals):
    table_data.append(elem)
pd.DataFrame(table_data, columns=['Feature Name', 'Importance'])
```

[53]:
```
    Feature Name  Importance
0          great    0.131911
1      delicious    0.089102
2      excellent    0.075286
3        amazing    0.072588
4           good    0.050104
..           …           …
995        guess    0.000000
```

```
996           guy    0.000000
997         gyoza    0.000000
998      gorgeous    0.000000
999          zero    0.000000

[1000 rows x 2 columns]
```

### 1.6.5 ROC-AUC (TODO)

[ ]:
```python
from sklearn.metrics import roc_curve, auc

# # Calculate the FPR and TPR for all thresholds of the classification
# for model_type in models.keys():
#     probs = models[model_type].predict_proba(Xtest)
#     preds = probs[:,1]
#     fpr, tpr, threshold = roc_curve(Ytest, preds)
#     roc_auc = auc(fpr, tpr)
#     plt.plot(fpr, tpr, 'b', label=f'{model_type} AUC={roc_auc:0.2f}')

# plt.title('Receiver Operating Characteristic')
# plt.legend(loc='lower right')
# plt.plot([0, 1], [0, 1], 'r--')
# plt.grid()
# plt.xlim([0, 1])
# plt.ylim([0, 1])
# plt.ylabel('True Positive Rate')
# plt.xlabel('False Positive Rate')
# plt.show()
```

## 1.7 Save and Load Models

[ ]:
```python
from joblib import dump, load

for model_type in models.keys():
    dump(models[model_type], os.path.join(data_dir, f'{model_type}.joblib'))
print('All models saved to disk.')
```

[ ]:
```python
for model_type in models.keys():
    models[model_type] = load(os.path.join(data_dir, f'{model_type}.joblib'))
print('All models loaded from disk.')
```

## 2 Converting Notebook to PDF

The following two cells can be ignored for grading, as they just convert this notebook into a PDF file.

[ ]:
```
%%capture
!apt-get update
!apt-get install -y texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
!apt-get install -y inkscape
!add-apt-repository -y universe
!add-apt-repository -y ppa:inkscape.dev/stable
!apt-get update -y
!apt install -y inkscape
```

[61]:
```
%%capture
import re

ASSIGNMENT_NAME = 'DAT340 - Assignment ' + ASSIGNMENT_ID.split('_')[1]
pdf_dir = os.path.join(os.path.abspath(''), 'drive', 'MyDrive')
pdf_dir = os.path.join(pdf_dir, 'Colab Notebooks', 'dat340', ASSIGNMENT_ID)
pdf_filename = re.escape(os.path.join(pdf_dir, ASSIGNMENT_NAME)) + '.ipynb'

!jupyter nbconvert --to pdf --TemplateExporter.exclude_input=False $pdf_filename
```

[ ]:
```

```