L2: Text classification

Text classification is the task of sorting text documents into predefined classes. The concrete problem you will be working on in this lab is the classification of texts with respect to their political affiliation. The specific texts you are going to classify are speeches held in the Riksdag, the Swedish national legislature.

Data set

The raw data for this lab comes from The Riksdag's Open Data. We have tokenized the speeches and put them into two compressed JSON files:

- `speeches-201718.json.bz2` (speeches from the 2017/2018 parliamentary session)
- `speeches-201819.json.bz2` (ditto, from the 2018/2019 session)

We start by loading these files into two separate data frames.

In [1]:
```
import pandas as pd
import bz2

with bz2.open('speeches-201718.json.bz2') as source:
    speeches_201718 = pd.read_json(source)

with bz2.open('speeches-201819.json.bz2') as source:
    speeches_201819 = pd.read_json(source)
```

When you inspect the two data frames, you can see that there are three labelled columns: `id` (the official speech ID), `words` (the space-separated words of the speech), and `party` (the party of the speaker, represented by its customary abbreviation).

In [2]:
```
speeches_201718.head()
```

Out[2]:

	id	words	party
0	H5-002-004	eders majestäter eders kungliga högheter herr ...	S
1	H5-003-001	aktuell debatt om situationen för ensamkommand...	V
2	H5-003-002	herr talman och ledamöter jag vill börja med a...	S
3	H5-003-003	herr talman åhörare den här debatten handlar a...	M
4	H5-003-004	herr talman ansvar och rättssäkerhet är två or...	SD

Throughout the lab, we will be using the speeches from 2017/2018 as our training data, and the speeches from 2018/2019 as our test data.

In [3]:
```
training_data, test_data = speeches_201718, speeches_201819
```

For later reference, we store the sorted list of party abbreviations.

In [4]:
```
parties = sorted(training_data['party'].unique())
print(parties)
```

```
['C', 'KD', 'L', 'M', 'MP', 'S', 'SD', 'V']
```

Problem 1: Visualization

Your first task is to get to know the data better by producing a simple visualization.

If you are not familiar with the Swedish political system and the parties represented in the Riksdag in particular, then we suggest that you have a look at the Wikipedia article about the 2018 Swedish general election.

For the lab, we ask you to compare the two data frames with respect to the distribution of the speeches over the different parties. Write code to generate two bar plots that visualize this information, one for the 2017/2018 speeches and one for the 2018/2019 speeches. Inspect the two plots, and compare them

- to each other
- to the results of the 2014 and the 2018 general elections

Summarize your observations in a short text in the cell below.

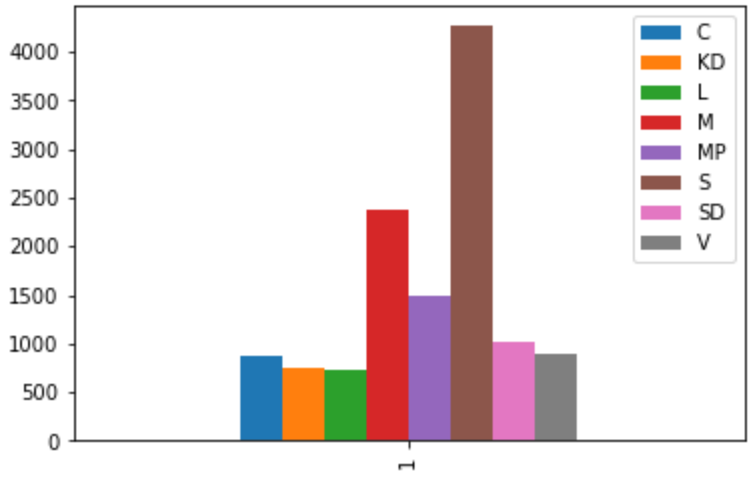
Tip: If you need help with creating bar plots, Bar Plot using Pandas provides a useful tutorial.

In [5]:
```
from collections import Counter
s_1718=Counter(speeches_201718["party"])
s_1819=Counter(speeches_201819["party"])

df=pd.DataFrame(dict(s_1718),columns=parties,index=[1])
df=df.append(pd.DataFrame(dict(s_1819),columns=parties,index=[2]))

df[0:1].plot(kind="bar")
df[1:2].plot(kind="bar")
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21d80a0ba48>
```



TODO: Enter your summary here

Decreased speechess for M and S were distributed to other parties, which also reflect on the seats lost in parliament in 2018 general election.

Problem 2: Naive Bayes classifier

You are now ready to train and evaluate a classifier. More specifically, we ask you to train a Multinomial Naive Bayes classifier. You will have to

1. vectorize the speeches in the training data
2. instantiate and fit the Naive Bayes model
3. evaluate the model on the test data

The scikit-learn library provides a convenience class Pipeline that allows you to solve the first two tasks with very compact code. For the evaluation you can use the function `classification_report` , which will report per-class precision, recall and F1, as well as overall accuracy.

In [6]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
# TODO: Write code here to train and evaluate a Multinomial Naive Bayes clas

#
# vectorizer = CountVectorizer()
# X = vectorizer.fit_transform(training_data["words"])
# clf = MultinomialNB()
# clf.fit(X,training_data["party"])
# test_X=vectorizer.transform(test_data["words"])
# res=clf.predict(test_X)
pipe=Pipeline([("vectorize",CountVectorizer()),("MultinomialNB",Multinomial

pipe.fit(training_data["words"],training_data["party"])

res=pipe.predict(test_data["words"])
print(classification_report(test_data["party"],res))
```

```
              precision    recall  f1-score   support

           C       0.63      0.04      0.07       671
          KD       0.70      0.02      0.03       821
           L       0.92      0.02      0.04       560
           M       0.36      0.68      0.47      1644
          MP       0.36      0.25      0.29       809
           S       0.46      0.84      0.59      2773
          SD       0.57      0.12      0.20      1060
           V       0.59      0.15      0.24       950

    accuracy                           0.43      9288
   macro avg       0.57      0.26      0.24      9288
weighted avg       0.52      0.43      0.34      9288
```

Would you have expected the results that you got?

The prediction is worse than we expected. Only 43% of accruacy over all classes.

Problem 3: Baselines

Evaluation metrics such as accuracy should not be understood as absolute measures of performance, but should be used only to compare different classifiers. When other classifiers are not available, a simple baseline is a classifier that generates predictions by random sampling, respecting the training set’s class distribution. This baseline is implemented by the class DummyClassifier. What is the performance of the random baseline on the test data?

In [7]:
```
# TODO: Write code here to evaluate the random baseline
from sklearn.dummy import DummyClassifier

pipe=Pipeline([("vectorize",CountVectorizer()),("MultinomialNB",DummyClassif
# dummy_clf=DummyClassifier(strategy="uniform")
# dummy_clf.fit(X,training_data["party"])
pipe.fit(training_data["words"],training_data["party"])
res=pipe.predict(test_data["words"])
```

```
C:\Users\Administrator\anaconda3\envs\kaggle\lib\site-packages\sklearn\dumm
y.py:132: FutureWarning: The default value of strategy will change from stra
tified to prior in 0.24.
  "stratified to prior in 0.24.", FutureWarning)
```

Comparing the result with naive bayes classifier, result from dummy classifier is much worse only 18% of accuracy.

An even dumber baseline is to predict, for every document, that class which appears most often in the training data. This baseline is also called the most frequent class baseline. What is the accuracy of that baseline on the test data?

In [8]:
```
# TODO: Write code here to print the accuracy of the most frequent class bas
print(classification_report(test_data["party"],res))
```

```
              precision    recall  f1-score   support

           C       0.06      0.06      0.06       671
          KD       0.07      0.05      0.06       821
           L       0.06      0.05      0.06       560
           M       0.19      0.21      0.20      1644
          MP       0.08      0.11      0.09       809
           S       0.30      0.35      0.32      2773
          SD       0.11      0.08      0.09      1060
           V       0.12      0.08      0.10       950

    accuracy                           0.18      9288
   macro avg       0.12      0.12      0.12      9288
weighted avg       0.17      0.18      0.17      9288
```

Problem 4: Creating a balanced data set

As you saw in Problem 1, the distribution of the speeches over the eight different parties (classes) is imbalanced. One technique used to alleviate this is **undersampling**, in which one randomly removes samples from over-represented classes until all classes are represented with the same number of samples.

Implement undersampling to create a balanced subset of the training data. Rerun the evaluation from Problem 2 on the balanced data and compare the results. Discuss your findings in a short text. Would you argue that undersampling make sense for the task of predicting the party of a speaker?

Hint: Your balanced subset should consist of 5,752 speeches.

In [9]:
```
from random import choices
# TODO: Write code here to implement undersampling

Counter(training_data["party"])

# 719
random_ind=[]
for par in parties:
    ind=training_data[training_data["party"]==par].index
    random_ind=random_ind+(choices(ind,k=719))

undersampling=training_data.iloc[random_ind]

pipe=Pipeline([("vectorize",CountVectorizer()),("MultinomialNB",MultinomialN

pipe.fit(undersampling["words"],undersampling["party"])

res=pipe.predict(test_data["words"])

print(classification_report(test_data["party"],res))
```

```
              precision    recall  f1-score   support

           C       0.25      0.40      0.30       671
          KD       0.35      0.32      0.34       821
           L       0.24      0.40      0.30       560
           M       0.37      0.53      0.44      1644
          MP       0.34      0.34      0.34       809
           S       0.79      0.32      0.45      2773
          SD       0.44      0.37      0.40      1060
           V       0.38      0.53      0.44       950

    accuracy                           0.40      9288
   macro avg       0.39      0.40      0.38      9288
weighted avg       0.48      0.40      0.40      9288
```

TODO: Enter your answer here

No, we don't think that undersampling would be a good idea for this case, with undersampling, our model would tend to be overfitting, it learned more specific information than general.

Problem 5: Confusion matrix

A **confusion matrix** is a specific table that is useful when analysing the performance of a classifier. In this table, both the rows and the columns correspond to classes, and each cell (i, j) states how many times a sample with gold-standard class i was predicted as belonging to class j.

In scitkit-learn, the confusion matrix of a classifier is computed by the function `confusion_matrix` . If you would rather see a visual representation, you can also use `plot_confusion_matrix` .

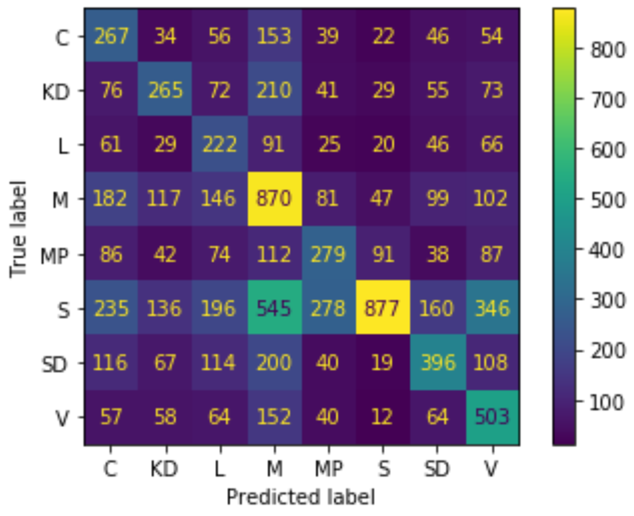
Your task is to use the confusion matrix in order to find, for each given party p in the Riksdag, that other party p' which the classifier that you trained in Problem 4 most often confuses with p when it predicts the party of a speaker.

In [10]:
```
# TODO: Write code here to solve Problem 5
from sklearn.metrics import confusion_matrix,plot_confusion_matrix

confusion_matrix(test_data["party"],res)

plot_confusion_matrix(pipe,test_data["words"],test_data["party"])
```

Out[10]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21d8350e
d08>
```



Take a minute to reflect on the question whether your results make sense.

Problem 6: Grid search

Until now, you have been using the vectorizer and the Naive Bayes classifier with their default hyperparameters. When working with real-world applications, you would want to find settings for the hyperparameters that maximize the performance for the task at hand.

Manually tweaking the hyperparameters of the various components of a vectorizer–classifier pipeline can be cumbersome. However, scikit-learn makes it possible to run an exhaustive search for the best hyperparameters over a grid of possible values. This method is known as **grid search**.

The hyperparameters of a pipeline should never be tuned on the final test set. (Why would that be a bad idea?) Instead, one should either use a separate validation set, or run cross-validation over different folds. Here we will use cross-validation.

Implement a grid search with 5-fold cross-validation to find the optimal parameters in a grid defined by the following choices for the hyperparameters:

- In the vectorizer, try a set-of-words model in addition to the default bag-of-words model (two possible parameter values).
- Also in the vectorizer, try extracting bigrams in addition to unigrams (two possible parameter

values).
- In the Naive Bayes classifier, try using additive smoothing with $\alpha \in \{1, 0.1\}$ (two possible parameter values).

Use the class GridSearchCV from the scikit-learn library. Print the results of your best model, along with the parameter values that yielded these results.

In [1]:
```
# TODO: Write code here to implement the grid search
from sklearn.model_selection import GridSearchCV

pipe=Pipeline([("vectorize",CountVectorizer()),("MultinomialNB",MultinomialN

parameter={"vectorize__binary":(True,False),
           "vectorize__ngram_range":((1,1),(1,2),(2,2)),
           "MultinomialNB__alpha":(1,0.1)}
clf=GridSearchCV(pipe,parameter)

clf.fit(training_data["words"],training_data["party"])

res=clf.predict(test_data["words"])

sorted(clf.cv_results_.keys())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-54ee65353583> in <module>
      2 from sklearn.model_selection import GridSearchCV
      3 
----> 4 pipe=Pipeline([("vectorize",CountVectorizer()),("MultinomialNB",Mult
inomialNB())])
      5 
      6 parameter={"vectorize__binary":(True,False),

NameError: name 'Pipeline' is not defined
```

In [27]:
```
clf.best_params_
print(classification_report(test_data["party"],res))
```

```
              precision    recall  f1-score   support

           C       0.39      0.27      0.32       671
          KD       0.45      0.24      0.31       821
           L       0.37      0.26      0.30       560
           M       0.44      0.58      0.50      1644
          MP       0.32      0.46      0.38       809
           S       0.61      0.65      0.63      2773
          SD       0.49      0.43      0.45      1060
           V       0.50      0.42      0.46       950

    accuracy                           0.48      9288
   macro avg       0.45      0.41      0.42      9288
weighted avg       0.48      0.48      0.48      9288
```

Problem 7: Try to improve your results

Scikit-learn makes it easy to test different vectorizer–classifier pipelines – among other things, it includes different types of logistic regression classifiers, support vector machines, and decision trees. Browse the library to see which methods are supported.

Build a pipeline that you find interesting, and use grid search to find optimal settings for the hyperparameters. Print the results of your best model. Did you manage to get better results than the ones that you obtained in Problem 6? Answer with a short text.

In [24]:
```
# TODO: Write code here to search for a better model
from sklearn.linear_model import LogisticRegression
pipe=Pipeline([("vectorize",CountVectorizer(binary=False,ngram_range=(1,1))
parameter={"LR__C":(0.1,0.5,1)}
clf=GridSearchCV(pipe,parameter,n_jobs=-1)
clf.fit(training_data["words"],training_data["party"])
res=clf.predict(test_data["words"])
```

```
C:\Users\Administrator\anaconda3\envs\kaggle\lib\site-packages\sklearn\linea
r_model\_logistic.py:764: ConvergenceWarning: lbfgs failed to converge (stat
us=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

TODO: Enter your answer here In this exercise, using CV with parameters C=0.1,0.5,1 and logistic regression as classifier, we found that the best model is with 0.1 as penalty factor, it return 50% accuracy which is higher than the best accuracy of SVM and Naive Bayes.

In [25]:
```
print(clf.best_params_)
print(classification_report(test_data["party"],res))
```

```
{'LR__C': 0.1}
              precision    recall  f1-score   support

           C       0.45      0.37      0.41       671
          KD       0.47      0.20      0.28       821
           L       0.42      0.28      0.34       560
           M       0.49      0.55      0.52      1644
          MP       0.28      0.42      0.34       809
           S       0.57      0.75      0.65      2773
          SD       0.52      0.40      0.45      1060
           V       0.57      0.30      0.39       950

    accuracy                           0.50      9288
   macro avg       0.47      0.41      0.42      9288
weighted avg       0.50      0.50      0.48      9288
```

Please read the section ‘General information’ on the ‘Labs’ page of the course website before submitting this notebook!