# HW 4: Classification </center>

Each assignment needs to be completed independently. Never ever copy others' work (even with minor modification, e.g. changing variable names). Anti-Plagiarism software will be used to check all submissions.

In this assignment, we build a classifier to identify ChatGPT-generated or human answers. The dataset is taken from https://huggingface.co/datasets/Hello-SimpleAI/HC3. Two files have been prepared for you for training and testing

- hw4_train.csv: dataset for training
- hw4_test.csv: dataset for testing.

The label column indicates if the answer is ChatGPT-generated (label 1) or human answer

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

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
data_train = pd.read_csv("hw4_train.csv")
data_train.head()
```

Out[2]:

| | question | answer | label |
|---|---|---|---|
| 0 | Rent or buy with 0 down | In the situation you describe, I would strongl... | 0 |
| 1 | Why has technology advanced so much in the las... | Technology has advanced a lot in the last 100 ... | 1 |
| 2 | Why can famous celebrities get away with doing... | It 's not enough to " know " , you need actual... | 0 |
| 3 | How do people become introverts / extroverts ?... | Introverts and extroverts are just different t... | 1 |
| 4 | Why are most of the foods that taste good bad ... | Well, foods that taste good are often high in ... | 1 |

In [3]:
```
data_test = pd.read_csv("hw4_test.csv")
data_test.head()
```

Out[3]:

| | question | answer | label |
|---|---|---|---|
| 0 | Why wo n't cats walk on aluminum foil ? I 've ... | Cats are just like people in the sense that no... | 0 |
| 1 | Nationalism and Globalism Hi ! I 'm wondering ... | Sure! Nationalism is a belief that people who ... | 1 |
| 2 | Why are there so many homeless in America ? I ... | There are many reasons why there are more home... | 1 |
| 3 | I'm thinking about selling some original artwo... | In the United States, sales tax is typically c... | 1 |
| 4 | Why does my TV still put out sound when I hit ... | When you hit the "mute" button on your TV remo... | 1 |

# Q1 Classification

- Define a function `create_model(train_docs, train_y, test_docs, test_y, model_type='svm', stop_words='english', min_df = 1, print_result = True)`, where
  - `train_docs`: is a list of documents for training
  - `train_y`: is the ground-truth labels of the training documents
  - `test_docs`: is a list of documents for test
  - `test_y`: is the ground-truth labels of the test documents
  - `model_type`: two options: `nb` (Multinomial Naive Bayes) or `svm` (Linear SVM)
  - `stop_words`: indicate whether stop words should be removed. The default value is 'english', i.e. remove English stopwords.
  - `min_df`: only word with document frequency above this threshold can be included. The default is 1.
  - `print_result`: controls whether to show classification report or plots. The default is True.

- This function does the following:
  - Fit a `TfidfVectorizer` using `train_docs` with options `stop_words, min_df` as specified in the function inputs. Extract features from `train_docs` using the fitted `TfidfVectorizer`.
  - Train `linear SVM` or `Multinomial Naive Bayes` model as specified by `model_type` using the extracted features and `train_y`.
  - Tranform `test_docs` by the fitted `TfidfVectorizer` (hint: use function `transform` not `fit_transform`).
  - Predict the labels for `test_docs`. If `print_result` is True, print the classification report.
  - Calculate the AUC score and PRC scores for class 1 on the test dataset. If `print_result` is True, plot the ROC and PRC curves. **Hint**:
    - `sklearn.svm.LinearSVM` does not provide `predict_proba` function.
    - Instead, you can use its `decision_function` (see some reference code)

- Another option is to use `sklearn.svm.SVC` with `kernel='linear' and probability=False` (see reference. This option can be very slow.)
  - Return the trained model, the fitted `TfidfVectorizer`, and the AUC and PRC scores.

- Test your function with the `answer` column as the input text in following cases:
  - model_type='svm', stop_words = 'english', min_df = 1
  - model_type='nb', stop_words = 'english', min_df = 1

In [4]:
```
from sklearn.model_selection import cross_validate
from sklearn.metrics import precision_recall_fscore_support, \
classification_report, roc_curve, auc, precision_recall_curve
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import svm
import numpy as np
import scipy
from matplotlib import pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics.pairwise import cosine_similarity
```

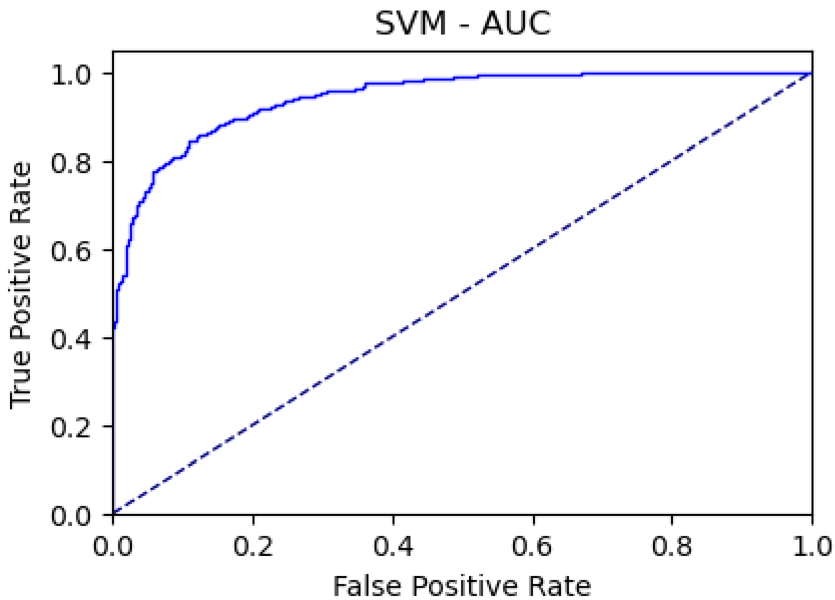
In [5]:
```
def create_model(train_docs, train_y, test_docs, test_y, \
              model_type='svm', stop_words=True, min_df = 1, print_result = Tru

    model, tfidf_vect, auc_score, prc_score = None, None, None, None

    # indicate whether stop words should be removed
    if stop_words:
        tfidf_vect = TfidfVectorizer(stop_words="english", min_df = min_df)
    else:
        tfidf_vect = TfidfVectorizer(min_df = min_df)

    # generate tfidf matrix
    X_train = tfidf_vect.fit_transform(train_docs)
    # generate tifid for new documents
    X_test = tfidf_vect.transform(test_docs)

    # to set model to train
    if model_type == "nb":
        model = MultinomialNB().fit(X_train, train_y)
        predict_p=model.predict_proba(X_test)[:,1]

    else:
        model = svm.LinearSVC().fit(X_train, train_y)
        predict_p =model.decision_function(X_test)

    # compute fpr/tpr by different thresholds
    fpr, tpr, thres = roc_curve(test_y, predict_p,pos_label=1)
    # compute precision/recall by different thresholds
    precision, recall, thresholds = precision_recall_curve(test_y,predict_p, po

    auc_score = auc(fpr, tpr)
    prc_score = auc(recall, precision)
```

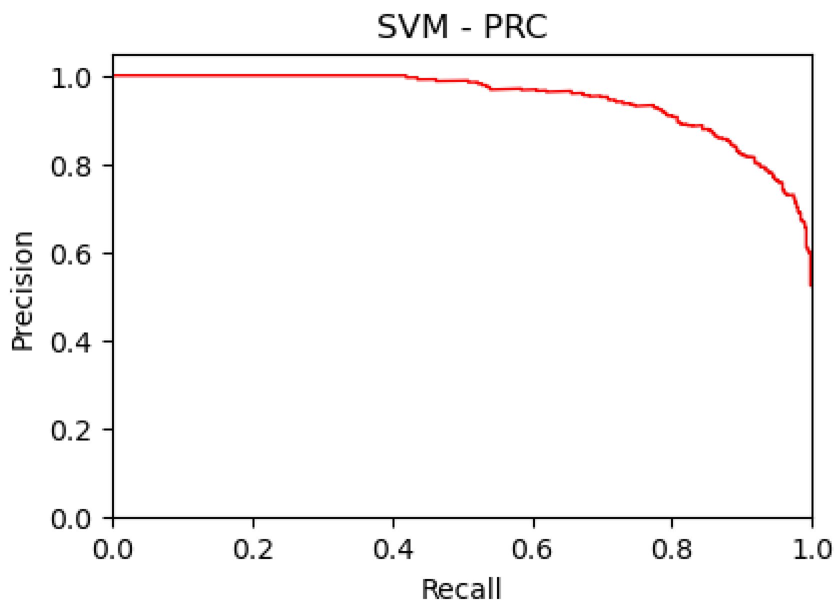
```
    if print_result:
    # to print the classification report
        y_pred =model.predict(X_test)
        print(classification_report(test_y, y_pred))
    # plot the ROC
        plt.figure(figsize=(4.5,3));
        plt.plot(fpr, tpr, color='blue', lw=1);
        plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--');
        plt.xlim([0.0, 1.0]);
        plt.ylim([0.0, 1.05]);
        plt.xlabel('False Positive Rate');
        plt.ylabel('True Positive Rate');
        plt.title('NB - AUC' if model_type=='nb' else 'SVM - AUC');
        plt.show();
    # plot the PRC
        plt.figure(figsize=(4.5,3));
        plt.plot(recall, precision, color='red', lw=1);
        plt.xlim([0.0, 1.0]);
        plt.ylim([0.0, 1.05]);
        plt.xlabel('Recall');
        plt.ylabel('Precision');
        plt.title('NB - PRC' if model_type=='nb' else 'SVM - PRC');
        plt.show();

        print("AUC: {:.2%}".format(auc_score),", PRC: {:.2%}".format(prc_score)

    return model, tfidf_vect, auc_score, prc_score
```

In [6]:

```
train_docs = data_train["answer"]
train_y = data_train["label"]
test_docs = data_test["answer"]
test_y = data_test["label"]
result = create_model(train_docs, train_y, test_docs, test_y, \
              model_type='svm', stop_words=True, min_df = 1, print_result = Tru
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86       500
           1       0.86      0.86      0.86       500

    accuracy                           0.86      1000
   macro avg       0.86      0.86      0.86      1000
weighted avg       0.86      0.86      0.86      1000
```

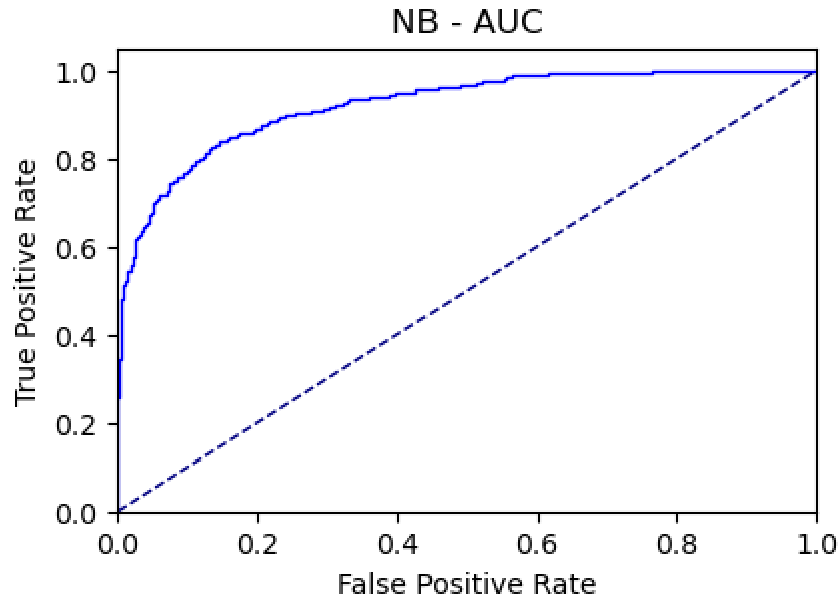


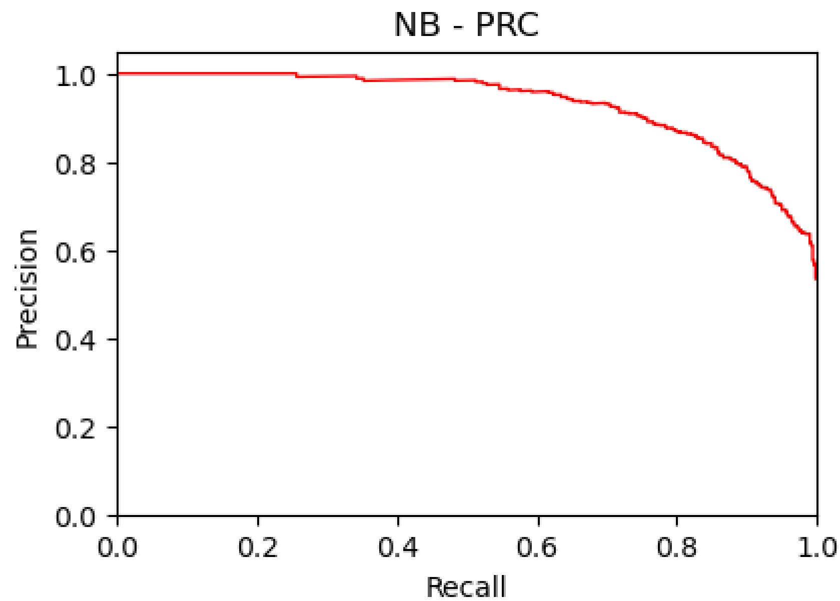




```
AUC: 94.36% , PRC: 94.78%
```

In [7]:
```
train_docs = data_train["answer"]
train_y = data_train["label"]
test_docs = data_test["answer"]
test_y = data_test["label"]
result = create_model(train_docs, train_y, test_docs, test_y, \
              model_type='nb', stop_words=True, min_df = 1, print_result = True
```

```
              precision    recall  f1-score   support

           0       0.91      0.67      0.77       500
           1       0.74      0.93      0.82       500

    accuracy                           0.80      1000
   macro avg       0.82      0.80      0.80      1000
weighted avg       0.82      0.80      0.80      1000
```




```
AUC: 92.36% , PRC: 93.08%
```

# Q2: Search for best parameters

From Task 1, you may find there are many possible ways to configure parameters. Next, let's use grid search to find the optimal parameters

- Define a function `search_para(docs, y, model_type = 'svm')` where
  - `docs` are training documents
  - `y` is the ground-truth labels
  - `model_type`: either SVM or Naive Bayes classifier
- This function does the following:
  - Create a pipleline which integrates `TfidfVectorizer` and the classifier
  - Define the parameter ranges as follow:
    - `stop_words': [None, 'english']`

- `min_df: [1,2,3, 5]`
  - Set the scoring metric to "f1_macro"
  - Use `GridSearchCV` with `5-fold cross validation` to find the best parameter values based on the training dataset.
  - Print the best parameter values
- For each SVM or Naive Bayes model, call the function `create_model` defined in Task 1 `with the best parameter values`.

- `Analysis`: Please briefly answer the following:
  - Compare with the model in Task 1, how is the performance improved on the test dataset?
  - Why do you think the new parameter values help sentiment classification?

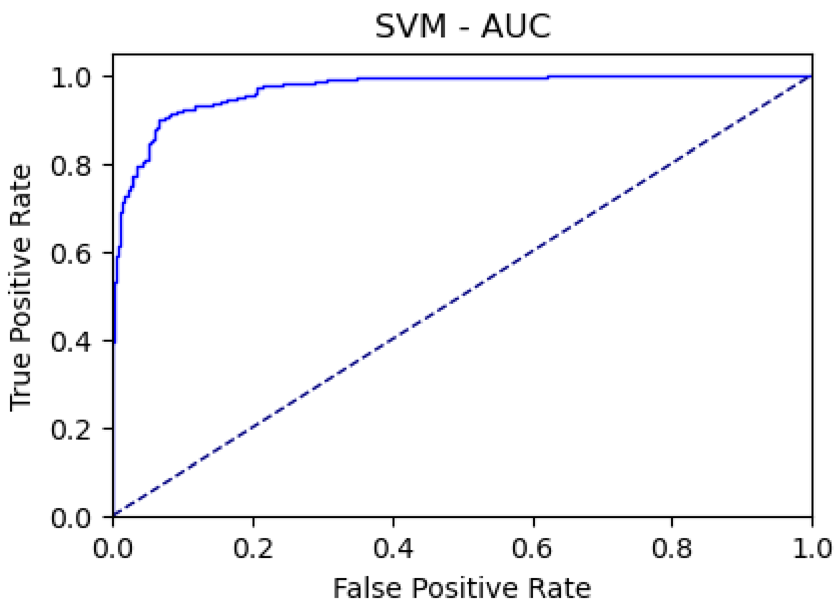
In [8]:

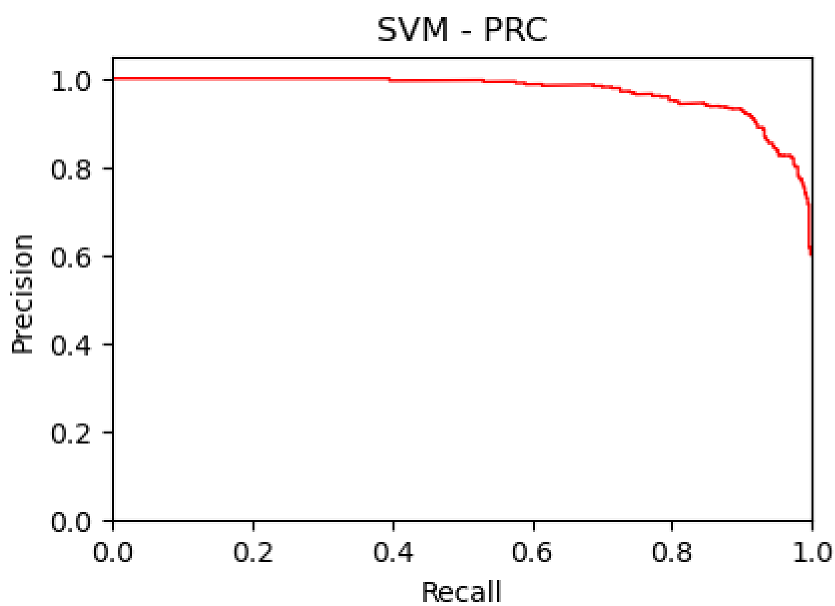
```
def search_para(docs, y, model_type = 'svm'):
    # generate a pipeline
    text_clf = Pipeline([('tfidf', TfidfVectorizer()),
                         ('clf', svm.LinearSVC() if model_type == 'svm'\
                          else MultinomialNB())
                        ])
    parameters = {'tfidf__min_df':[1,2,3, 5],
                 'tfidf__stop_words':[None,"english"],}
    metric =  "f1_macro"

    gs_clf = GridSearchCV(text_clf, param_grid=parameters, \
                          scoring=metric, cv=5)
    gs_clf = gs_clf.fit(docs, y)

    return gs_clf
```

In [9]:

```
docs = data_train["answer"]
y = data_train["label"]
gs_clf = search_para(docs, y, model_type = 'svm')
for param_name in gs_clf.best_params_:
    print("{0}:\t{1}".format(param_name,\
                              gs_clf.best_params_[param_name]))

print("best f1 score: {:.3f}".format(gs_clf.best_score_))

result = create_model(train_docs, train_y, test_docs, test_y, \
              model_type='svm', stop_words=gs_clf.best_params_["tfidf__stop_wor
                      min_df = gs_clf.best_params_["tfidf__min_df"], print_resu
```

```
tfidf__min_df:  2
tfidf__stop_words:      None
best f1 score: 0.902
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       500
           1       0.90      0.92      0.91       500

    accuracy                           0.91      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```



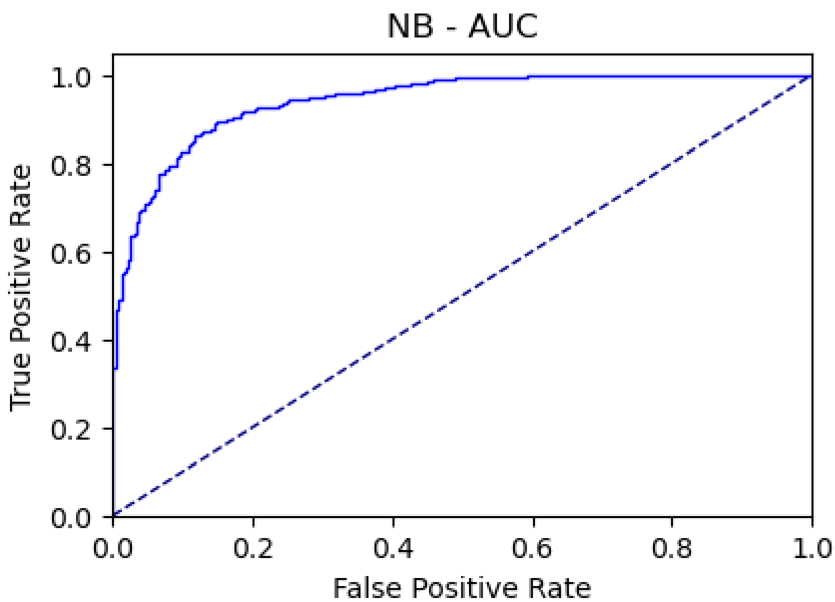



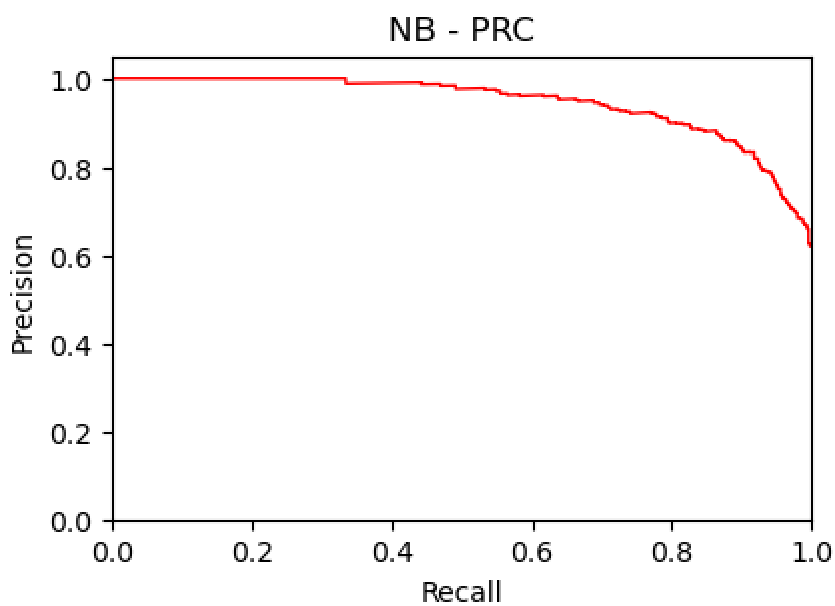

```
AUC: 97.03% , PRC: 97.15%
```

In [10]:

```
# Best parameters for Naive Bayes model

# Retrain Naive Bayes model with the best parameters

# Add your code here
gs_clf = search_para(docs, y, model_type = 'nb')
for param_name in gs_clf.best_params_:
    print("{0}:\t{1}".format(param_name,\
                             gs_clf.best_params_[param_name]))

print("best f1 score: {:.3f}".format(gs_clf.best_score_))

result = create_model(train_docs, train_y, test_docs, test_y, \
              model_type='nb', stop_words=gs_clf.best_params_["tfidf__stop_word
                      min_df = gs_clf.best_params_["tfidf__min_df"], print_resu
```

```
tfidf__min_df:  5
tfidf__stop_words:      None
best f1 score: 0.851
              precision    recall  f1-score   support

           0       0.89      0.83      0.86       500
           1       0.84      0.90      0.87       500

    accuracy                           0.86      1000
   macro avg       0.87      0.86      0.86      1000
weighted avg       0.87      0.86      0.86      1000
```




```
AUC: 94.39% , PRC: 94.59%
```

Compared to the model in Question 1, we observe the following in Question 2:

- The SVM model achieves a higher accuracy for the f1_score, 5 points. Similarly, the NB model achieves a higher accuracy, 6 points. Furthermore, the AUC and PRC scores in Q2 are also better than Q1, with the curves closer to the corner.

- The new parameter values help sentiment classification because we did not remove stopword. When we remove stopwords it could cause a shift in the balance between positive and negative words and removing stopwords maybe remove important information from the text and affect the performance of the sentiment classification model.

## Q3: Improved Classifier

So far we only considered the TFIDF weights as features. Can you considered other features, e.g. the differences you noticed in HW3, and incorporate these features into your classifier? Your target is to `improve the F1 macro score of your SVM or Naive Bayes by at least 2%`.

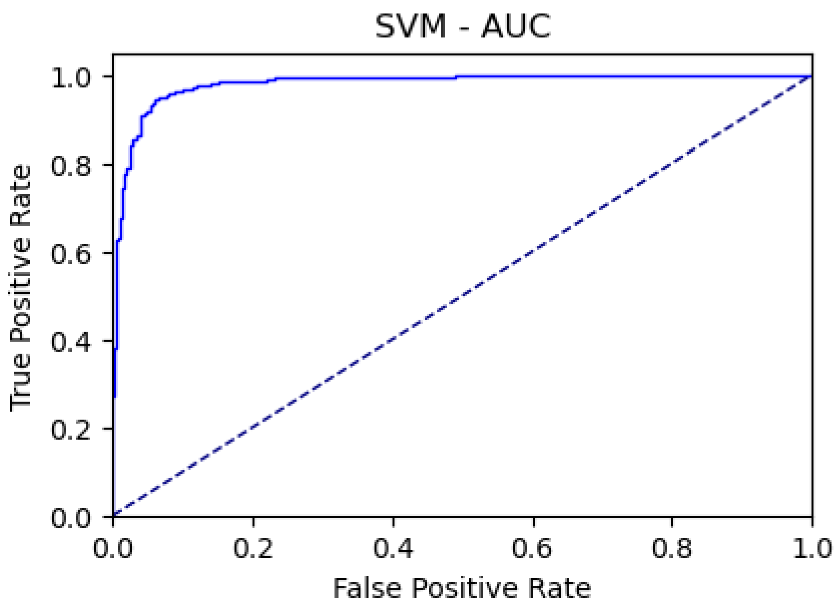
In [11]:
```
def create_advanced_model(train_df, train_y, test_df, test_y,\
                          stop_words=None, min_df = 1, print_result = True):

    model, tfidf_vect, auc_score, prc_score = None, None, None, None

    if stop_words:
        tfidf_vect = TfidfVectorizer(stop_words="english", min_df = min_df)
        vectorizer = CountVectorizer(stop_words="english", min_df = min_df)
    else:
        tfidf_vect = TfidfVectorizer(min_df = min_df)
        vectorizer = CountVectorizer(min_df = min_df)

    X_train = tfidf_vect.fit_transform(train_df)
    X_test = tfidf_vect.transform(test_df)
    # use CountVectorizer() to count words in each document we consider for moc
    data = pd.concat([train_df, test_df])
    count = vectorizer.fit_transform(data)
    lenght = count.sum(axis = 1)
    # create a new feature with log(length each doccument)
    lenght_ans_train=np.log(lenght[:len(train_df),:])
    lenght_ans_test = np.log(lenght[len(train_df):,:])

    # to combine into a new dataset with TF-IDF matrix and the log of length dc

    dataset_train = scipy.sparse.hstack([X_train,lenght_ans_train])

    dataset_test = scipy.sparse.hstack([X_test,lenght_ans_test])


    # I use SVM to improve the F1 macro score

    model = svm.LinearSVC().fit(dataset_train, train_y)
    predict_p = model.decision_function(dataset_test)

    y_pred =model.predict(dataset_test)


    fpr, tpr, thres = roc_curve(test_y, predict_p, pos_label=1)

    precision, recall, thresholds = precision_recall_curve(test_y,predict_p, pc
    auc_score = auc(fpr, tpr)
```

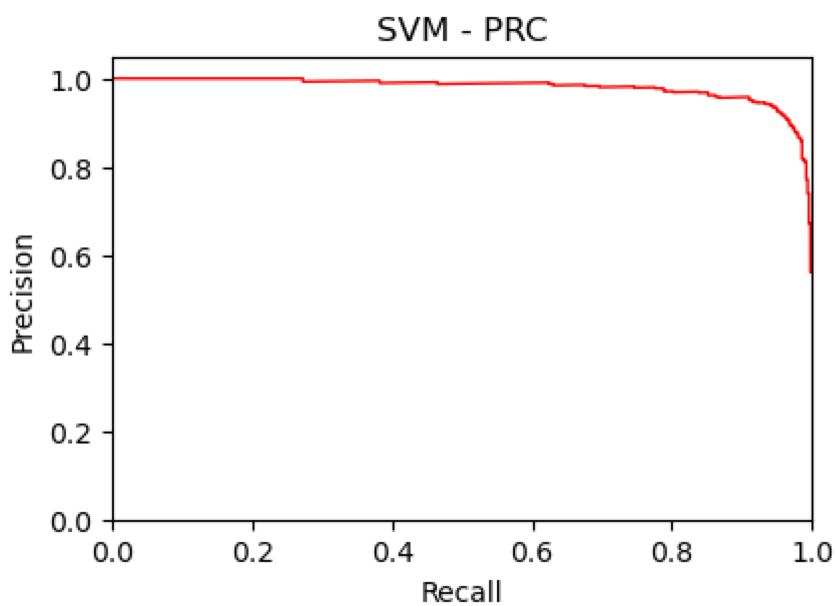
```
    prc_score = auc(recall, precision)

    if print_result:
        print(classification_report(test_y, y_pred))
        plt.figure(figsize=(4.5,3));
        plt.plot(fpr, tpr, color='blue', lw=1);
        plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--');
        plt.xlim([0.0, 1.0]);
        plt.ylim([0.0, 1.05]);
        plt.xlabel('False Positive Rate');
        plt.ylabel('True Positive Rate');
        plt.title('SVM - AUC');
        plt.show();

        plt.figure(figsize=(4.5,3));
        plt.plot(recall, precision, color='red', lw=1);
        plt.xlim([0.0, 1.0]);
        plt.ylim([0.0, 1.05]);
        plt.xlabel('Recall');
        plt.ylabel('Precision');
        plt.title('SVM - PRC');
        plt.show();

        print("AUC: {:.2%}".format(auc_score),", PRC: {:.2%}".format(prc_score)
    return model, tfidf_vect, auc_score, prc_score
```

In [12]:

```
train_df = data_train["answer"]
train_y = data_train["label"]
test_df = data_test["answer"]
test_y = data_test["label"]
result = create_advanced_model(train_df, train_y, test_df, test_y, \
             stop_words=None, min_df = 3, print_result = True)
```

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       500
           1       0.94      0.94      0.94       500

    accuracy                           0.94      1000
   macro avg       0.94      0.94      0.94      1000
weighted avg       0.94      0.94      0.94      1000
```

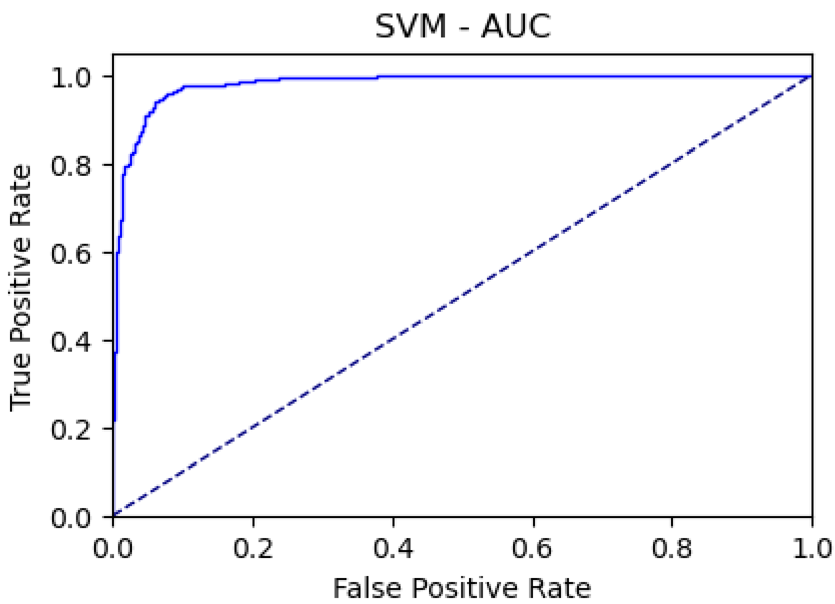


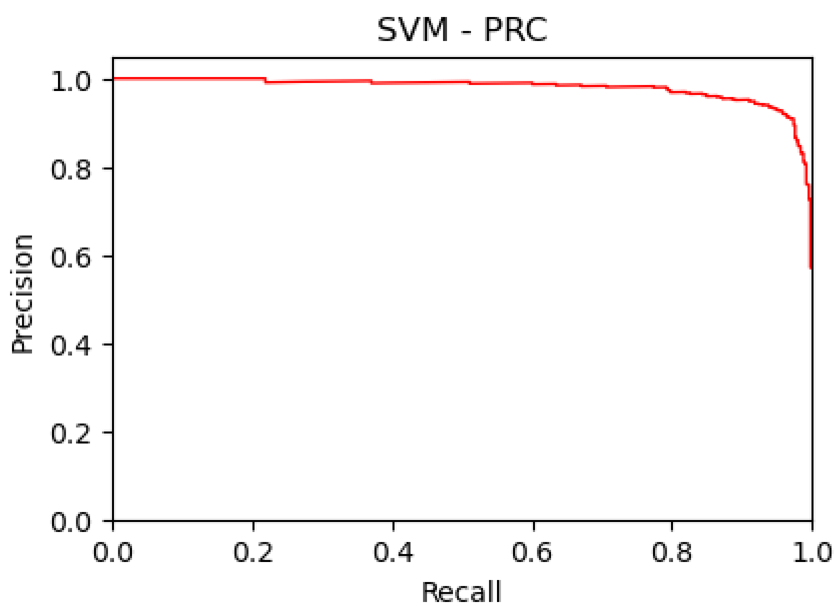




```
AUC: 98.10% , PRC: 97.99%
```

In [13]:
```
train_df = data_train["answer"]
train_y = data_train["label"]
test_df = data_test["answer"]
test_y = data_test["label"]
result = create_advanced_model(train_df, train_y, test_df, test_y, \
             stop_words=None, min_df = 4, print_result = True)
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.95 | 0.93 | 0.94 | 500 |
| 1 | 0.93 | 0.95 | 0.94 | 500 |
| accuracy | | | 0.94 | 1000 |
| macro avg | 0.94 | 0.94 | 0.94 | 1000 |
| weighted avg | 0.94 | 0.94 | 0.94 | 1000 |




```
AUC: 98.09% , PRC: 97.94%
```

- As be can seen, the F1 macro score in my model is higher 3% compared to the model in Question 2

## Q4 (Bonus): Model Interpretation

Take the best-performing model you achieve in Task 2, can you identify the most important words that can differentiate human answers from ChatGPT generated answers?

For both SVM and Naive Bayes models, describe your idea, implement your idea, and show the top 20 most descrimiating words.

- Yes, we can identify the identify the most important words that can differentiate human answers from ChatGPT generated answers, for example:

- For SVM model, we find the coefficient of each feature with the taget label. A positive coefficient means that the feature has a positive effect on the target variable, while a negative coefficient means that the feature has a negative effect. So, the most important words have the highest coefficients, as they have the greatest impact on the prediction.
- For Naive Bayes model, we find the log probability of the words through each label, and compare the difference of each word in 2 labels. Finally, the most important words have the highest differences, as they have the greatest impact on the prediction.
  - A small probability will have a large negative log probability, indicating that they have minimal impact on the model. To determine the highlight score of a feature, we need to consider the difference between the two classes.
  - We can also try the highest log probability of words in both label 1 and label 0. We can see that there is not much the top words between 2 labels.
- I get some ideas from this link: https://stackoverflow.com/questions/11116697/how-to-get-most-informative-features-for-scikit-learn-classifiers

In [14]:
```
docs = data_train["answer"]
y = data_train["label"]
gs_clf = search_para(docs, y, model_type = 'svm')
model, tfidf_vect, auc_score, prc_score = create_model(train_docs, train_y, tes
              model_type='svm', stop_words=gs_clf.best_params_["tfidf__stop_wor
                      min_df = gs_clf.best_params_["tfidf__min_df"], print_resu
svm_coef = model.coef_[0]
svm_names = np.array(tfidf_vect.get_feature_names_out())

topwords_svm = svm_names[np.argsort(svm_coef)[-20:]]
```

In [15]:
```
docs = data_train["answer"]
y = data_train["label"]
gs_clf = search_para(docs, y, model_type = 'nb')
model, tfidf_vect, auc_score, prc_score = create_model(train_docs, train_y, tes
              model_type='nb', stop_words=gs_clf.best_params_["tfidf__stop_word
                      min_df = gs_clf.best_params_["tfidf__min_df"], print_resu
nb_coef = model.feature_log_prob_
diff_score = nb_coef[0] - nb_coef[1]
nb_names = np.array(tfidf_vect.get_feature_names_out())
topwords_nb = nb_names[np.argsort(diff_score)[-20:]]
topwords_nb_log_1 = nb_names[np.argsort(nb_coef[1])[-20:]]
topwords_nb_log_0 = nb_names[np.argsort(nb_coef[0])[-20:]]
```

In [16]:
```
print("The top 20 most descrimiating words for SVM model: ")
print((topwords_svm),"\n")
print("The top 20 most descrimiating words for Naive Bayes model with the diffe
print(topwords_nb,"\n")

print("-----OTHER OBSERVATION FOR NB-----\n")
print("The top 20 most descrimiating words for Naive Bayes model with the log p
print(topwords_nb_log_1)
print("The top 20 most descrimiating words for Naive Bayes model with the log p
print(topwords_nb_log_0)
print("Almost the most descrimiating words in 2 classes are stopword. It can no
```

```
The top 20 most descrimiating words for SVM model:
['other' 'questions' 'sure' 'means' 'different' 'such' 'located' 'because'
 'earth' 'called' 'help' 'to' 'including' 'helps' 'can' 'or' 'might'
 'important' 'may' 'and']

The top 20 most descrimiating words for Naive Bayes model with the difference:
['url_1' 'll' 'edit' 'answered' 'going' 'thus' 'now' 'regards' 'really'
 'query' 'pretty' 'hello' 'got' 'basically' 've' 'dr' 'my' 'etc' 'ca'
 'url_0']

-----OTHER OBSERVATION FOR NB-----

The top 20 most descrimiating words for Naive Bayes model with the log probabi
lity for label 1:
['not' 'on' 'they' 'may' 'as' 'your' 'be' 'for' 'are' 'can' 'you' 'or'
 'in' 'that' 'it' 'is' 'of' 'and' 'to' 'the']
The top 20 most descrimiating words for Naive Bayes model with the log probabi
lity for label 0:
['not' 'this' 'but' 'be' 'as' 'have' 'if' 'for' 'they' 'are' 'your' 'that'
 'in' 'it' 'is' 'and' 'of' 'you' 'to' 'the']
Almost the most descrimiating words in 2 classes are stopword. It can not the
pivot features to help for model.
```

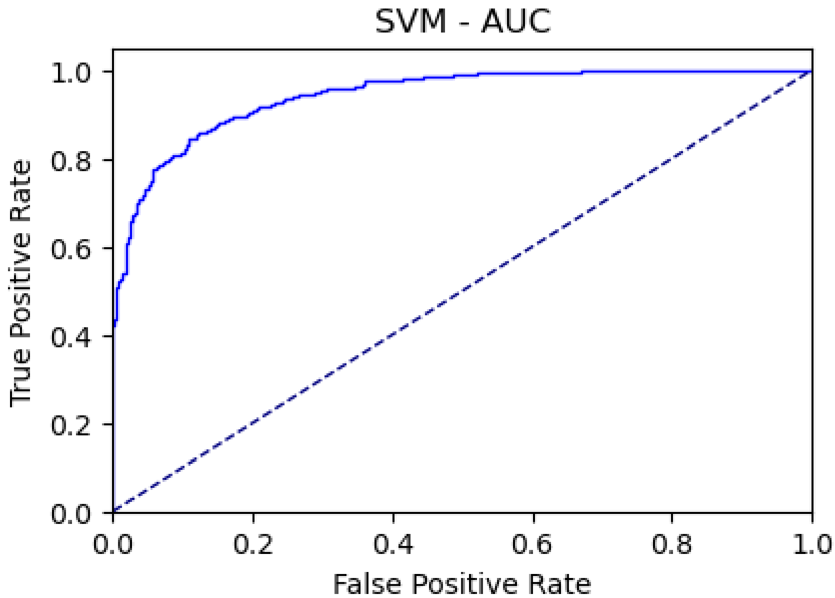
# Test

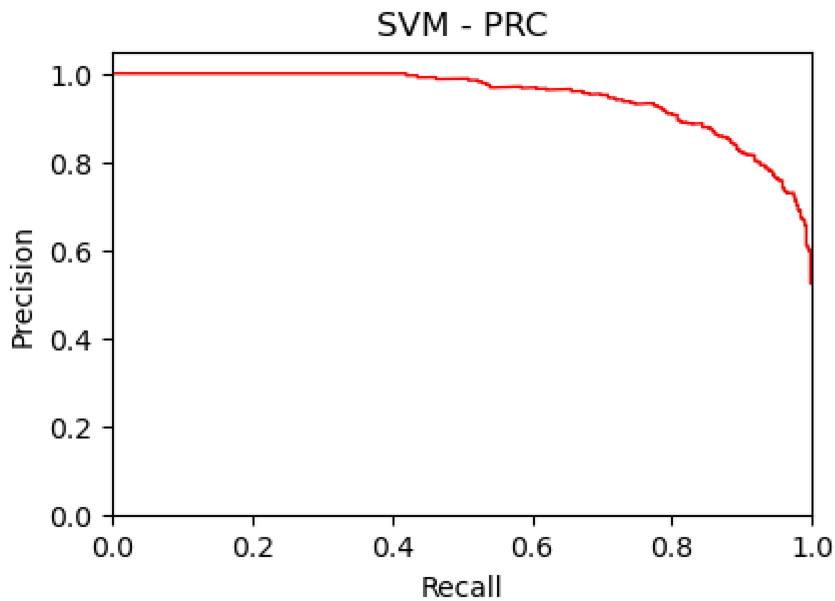
In [19]:

```python
if __name__ == "__main__":

    # add test code
    print("---QUESTION 1:---\n")
    train_docs = data_train["answer"]
    train_y = data_train["label"]
    test_docs = data_test["answer"]
    test_y = data_test["label"]
    print("a) For SVM model:\n")
    result = create_model(train_docs, train_y, test_docs, test_y, \
                  model_type='svm', stop_words=True, min_df = 1, print_result =
    print("\nb) For NB model:\n")
    result_1 = create_model(train_docs, train_y, test_docs, test_y, \
                  model_type='nb', stop_words=True, min_df = 1, print_result =

    print("\n---QUESTION 2:---\n")
    docs = data_train["answer"]
    y = data_train["label"]
    gs_clf = search_para(docs, y, model_type = 'svm')
    print("a) The best parameters for SVM model:\n")
    for param_name in gs_clf.best_params_:
        print("{0}:\t{1}".format(param_name,\
                                 gs_clf.best_params_[param_name]))

    print("best f1 score: {:.3f}".format(gs_clf.best_score_))

    model, tfidf_vect, auc_score, prc_score = create_model(train_docs, train_y,
                  model_type='svm', stop_words=gs_clf.best_params_["tfidf__stop
                          min_df = gs_clf.best_params_["tfidf__min_df"], print_

    print("\nb) The best parameters for NB model:\n")
    gs_clf = search_para(docs, y, model_type = 'nb')
    for param_name in gs_clf.best_params_:
```

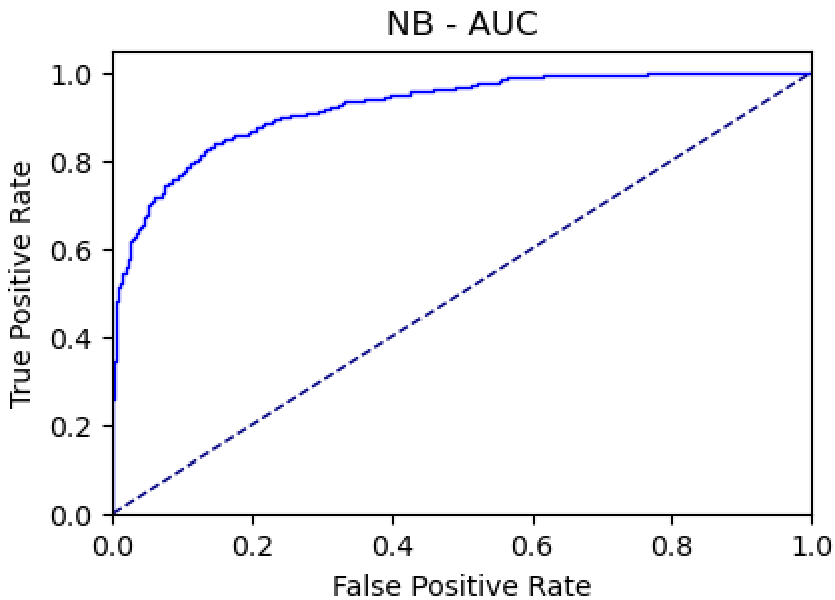
```
            print("{0}:\t{1}".format(param_name,\
                                           gs_clf.best_params_[param_name]))

    print("best f1 score: {:.3f}".format(gs_clf.best_score_))

    model1, tfidf_vect1, auc_score, prc_score = create_model(train_docs, train_
                  model_type='nb', stop_words=gs_clf.best_params_["tfidf__stop_
                          min_df = gs_clf.best_params_["tfidf__min_df"], print_

    print("\n---QUESTION 3:---\n")
    train_df = data_train["answer"]
    train_y = data_train["label"]
    test_df = data_test["answer"]
    test_y = data_test["label"]
    result = create_advanced_model(train_df, train_y, test_df, test_y, \
                 stop_words=None, min_df = 3, print_result = True)

    print("\n---QUESTION 4:---\n")

    svm_coef = model.coef_[0]
    svm_names = np.array(tfidf_vect.get_feature_names_out())
    topwords_svm = svm_names[np.argsort(svm_coef)[-20:]]

    nb_prob = model1.feature_log_prob_
    diff_score = nb_prob[0] - nb_prob[1]
    nb_names = np.array(tfidf_vect1.get_feature_names_out())
    topwords_nb = nb_names[np.argsort(diff_score)[-20:]]
    topwords_nb_log_1 = nb_names[np.argsort(nb_prob[1])[-20:]]
    topwords_nb_log_0 = nb_names[np.argsort(nb_prob[0])[-20:]]
    print("The top 20 most descrimiating words for SVM model: ")
    print((topwords_svm),"\n")
    print("The top 20 most descrimiating words for Naive Bayes model with the c
    print(topwords_nb,"\n")

    print("-----OTHER OBSERVATION FOR NB-----\n")
    print("The top 20 most descrimiating words for Naive Bayes model with the l
    print(topwords_nb_log_1)
    print("The top 20 most descrimiating words for Naive Bayes model with the l
    print(topwords_nb_log_0)
    print("Almost the most descrimiating words in 2 classes are stopword. It ca
```

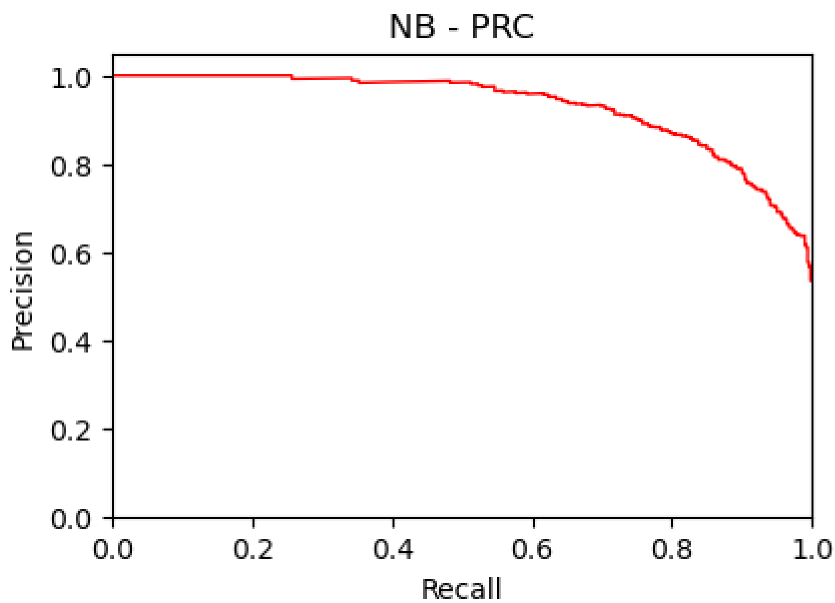
```
---QUESTION 1:---

a) For SVM model:

              precision    recall  f1-score   support

           0       0.86      0.86      0.86       500
           1       0.86      0.86      0.86       500

    accuracy                           0.86      1000
   macro avg       0.86      0.86      0.86      1000
weighted avg       0.86      0.86      0.86      1000
```



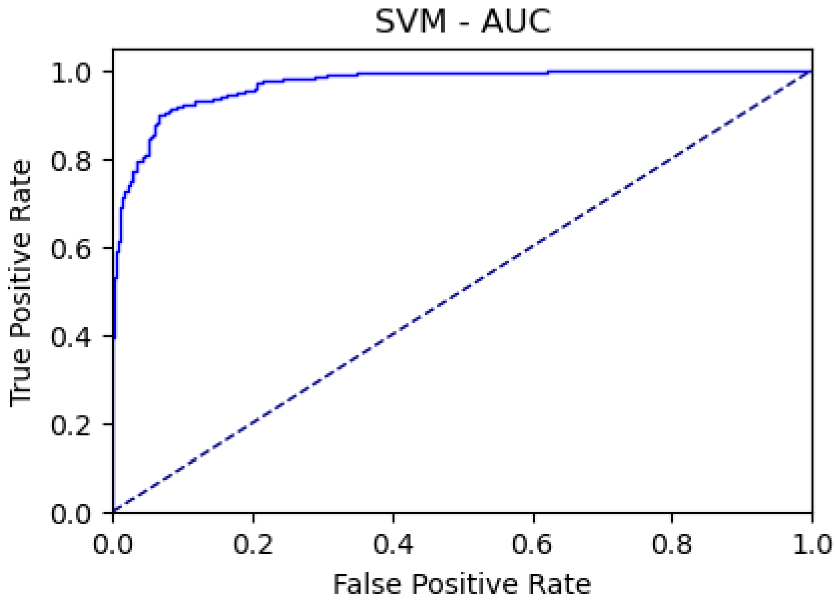



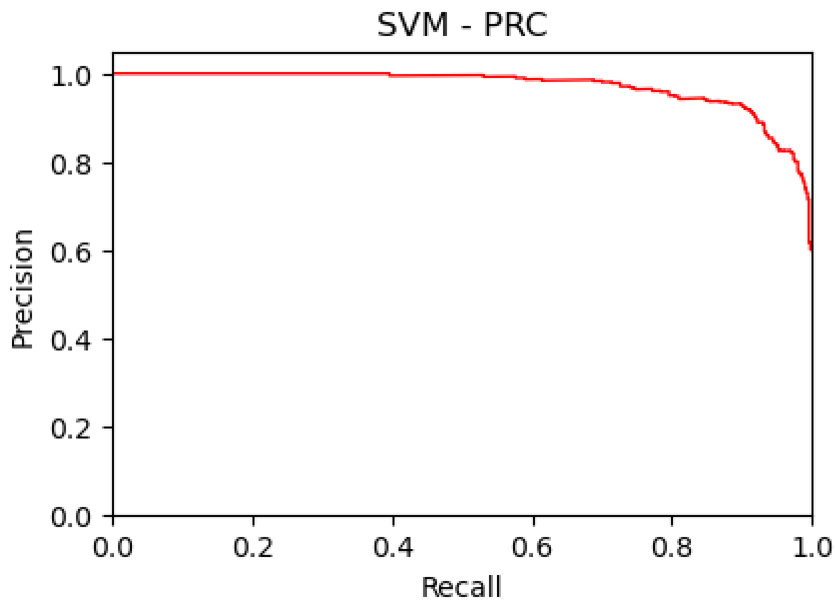

```
AUC: 94.36% , PRC: 94.78%

b) For NB model:

              precision    recall  f1-score   support

           0       0.91      0.67      0.77       500
           1       0.74      0.93      0.82       500

    accuracy                           0.80      1000
   macro avg       0.82      0.80      0.80      1000
weighted avg       0.82      0.80      0.80      1000
```



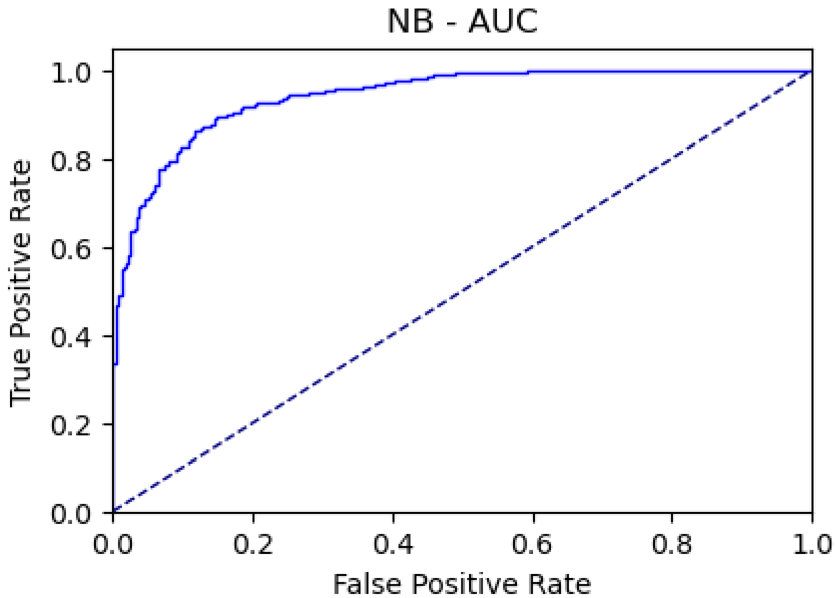



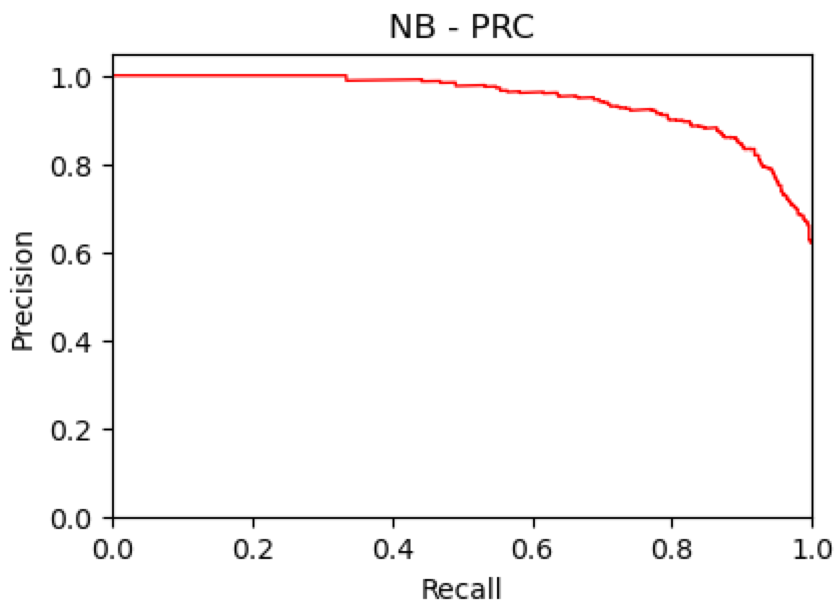

```
AUC: 92.36% , PRC: 93.08%

---QUESTION 2:---

a) The best parameters for SVM model:

tfidf__min_df:  2
tfidf__stop_words:      None
best f1 score: 0.902
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       500
           1       0.90      0.92      0.91       500

    accuracy                           0.91      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```



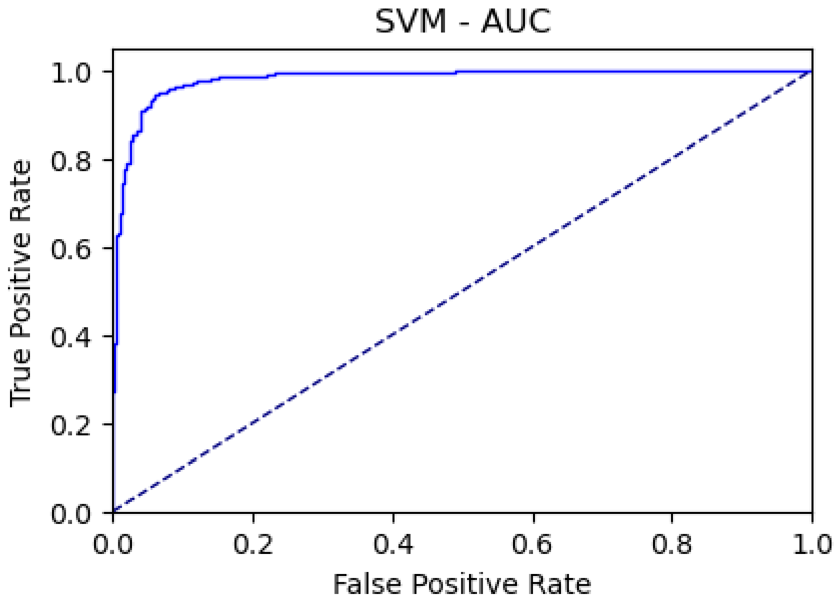



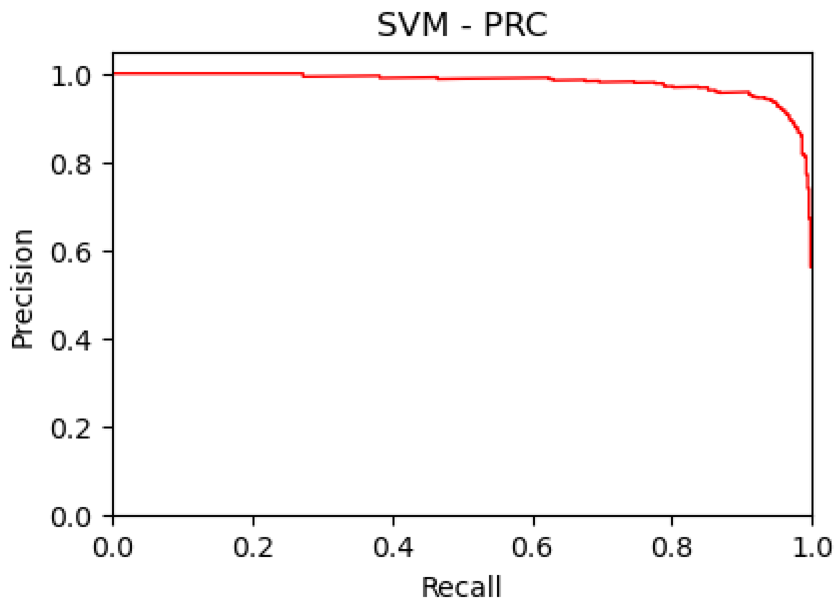

```
AUC: 97.03% , PRC: 97.15%

b) The best parameters for NB model:

tfidf__min_df:  5
tfidf__stop_words:      None
best f1 score: 0.851
              precision    recall  f1-score   support

           0       0.89      0.83      0.86       500
           1       0.84      0.90      0.87       500

    accuracy                           0.86      1000
   macro avg       0.87      0.86      0.86      1000
weighted avg       0.87      0.86      0.86      1000
```




```
AUC: 94.39% , PRC: 94.59%

---QUESTION 3:---

              precision    recall  f1-score   support

           0       0.94      0.94      0.94       500
           1       0.94      0.94      0.94       500

    accuracy                           0.94      1000
   macro avg       0.94      0.94      0.94      1000
weighted avg       0.94      0.94      0.94      1000
```


SVM - AUC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


SVM - PRC
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

```
AUC: 98.10% , PRC: 97.99%

---QUESTION 4:---

The top 20 most descrimiating words for SVM model:
['other' 'questions' 'sure' 'means' 'different' 'such' 'located' 'because'
 'earth' 'called' 'help' 'to' 'including' 'helps' 'can' 'or' 'might'
 'important' 'may' 'and']

The top 20 most descrimiating words for Naive Bayes model with the difference:
['url_1' 'll' 'edit' 'answered' 'going' 'thus' 'now' 'regards' 'really'
 'query' 'pretty' 'hello' 'got' 'basically' 've' 'dr' 'my' 'etc' 'ca'
 'url_0']

-----OTHER OBSERVATION FOR NB-----

The top 20 most descrimiating words for Naive Bayes model with the log probabi
lity for label 1:
['not' 'on' 'they' 'may' 'as' 'your' 'be' 'for' 'are' 'can' 'you' 'or'
 'in' 'that' 'it' 'is' 'of' 'and' 'to' 'the']
The top 20 most descrimiating words for Naive Bayes model with the log probabi
lity for label 0:
['not' 'this' 'but' 'be' 'as' 'have' 'if' 'for' 'they' 'are' 'your' 'that'
 'in' 'it' 'is' 'and' 'of' 'you' 'to' 'the']
Almost the most descrimiating words in 2 classes are stopword. It can not the
pivot features to help for model.
```

In [ ]: