INTELLIGENT DATA ANALYSIS

# SPAM CLASSIFIER

Valdimar Ágúst Eggertsson (800244),
02.10.19

- 1. Introduction
- 2. Naive Bayes Classifier
- 3. Support Vector Machine
- 4. Random Forest
- 5. Conclusions / comments

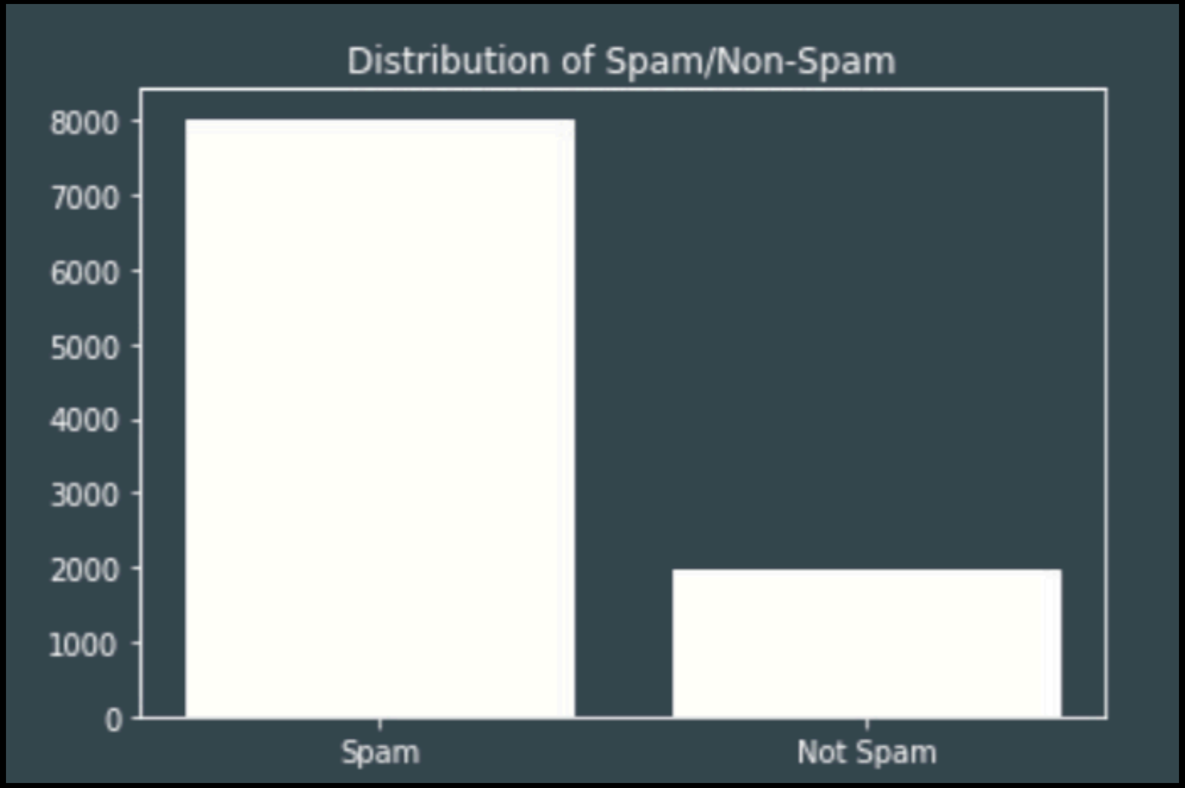
# INTRODUCTION

## Problem Analysis - Requirements

- The task is to train a classifier that can determine whether a bag-of-words representation of an email is spam or not.
- False positives are worse than false negatives (losing mail as spam is bad - story). No more than 0.2% FPR!
- Methods used:
  - Naive Bayes, the classic method.
  - Support Vector Machines (SVMs),
  - Random Forests (RF) 🌳 🌳 🌳
  - ~~Neural Network~~

## The data set

- The data has already been pre-processed to a bag of words representation, with a word count vector for each email.
- Stored in a sparse matrix, with 10.000 emails (data rows) and 57173 words (features).
- Class ratio is realistic.



## Key results

- Naive Bayes, the go-to solution for the task, did the job just fine.
- To make it into more of a machine learning exercise:
  - I spent some days playing around with SVMs and RFs trying to get a classifier that was as good.
  - And it didn't really work out!
    - Found some ok classifiers but none as good.
- Moral: Use simple tools when they work!

# INTRODUCTION

Overview of the Jupyter Notebook written to solve the task

## Machine Learning Exercise - Spam filter

Table of contents

# NAIVE BAYES

# NAIVE BAYES

- Naive assumption
  - Words in an email are assumed to be independent events.
    - Only the relative frequencies of words are used to estimate the probability of a bunch of them occurring together. No correlation.
    - The assumption is obviously false.
      - But doesn't matter, because words such as 'viagra' are so much more likely to appear in spam than in ham.
      - -> We don't need to worry about the conditional probabilities of words on the other words appearing in the mail.
- Logic of the method makes sense (not a black box).
  - I started the exercise by programming it from scratch
  - but ended up using sklearn, and reminded myself to not waste energy on inventing the wheel

## Finding the optimal smoothing parameter alpha

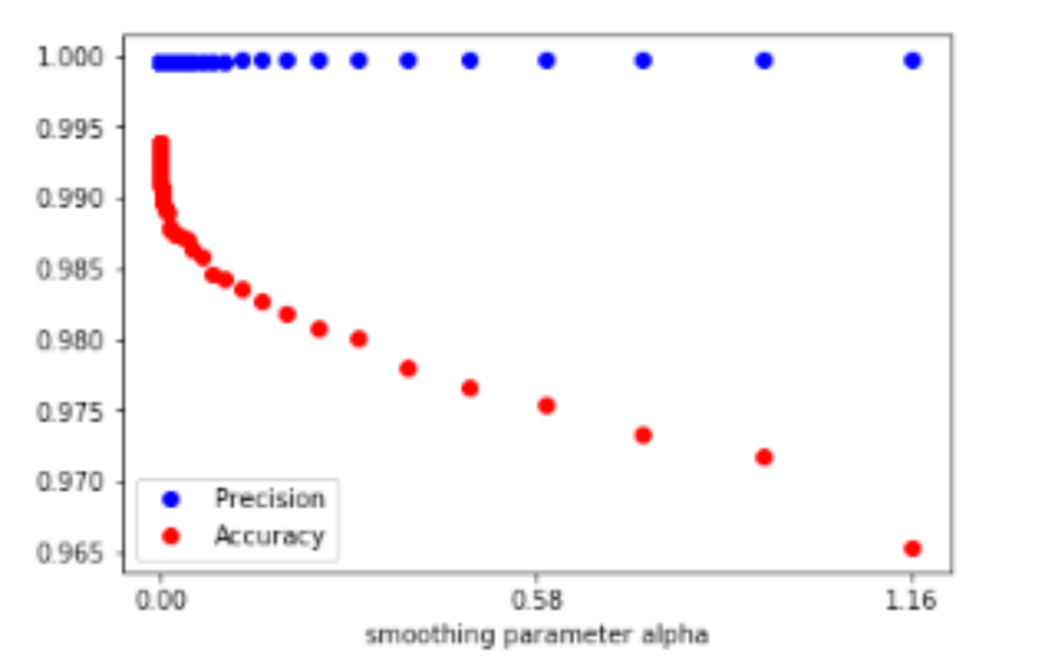
- Only one parameter to tune:
  - Smoothing parameter 'alpha'
  - With / Without TF-IDF
- Evaluation criteria:
  - Get minimum false positive rate and maximum accuracy.
  - If many alphas are as good, choose the one with the highest value.

## Finding the optimal smoothing parameter alpha

- I trained models for various values, using 5-fold (triple) stratified cross validation
- Alpha is by default 1.0.  Better results had lower alpha.
  - Might be a problem for emails with words not in the trained vocabulary - they'll have a small probability.
  - Overfitting? New types of spam not caught?
- Using TF-IDF pre-processing didn't make it better.
- Tuned it manually. Programming everything took time, but I learned from it.
- Lesson learned: Don't spend energy coding things from scratch, except if there's something to learn from it!

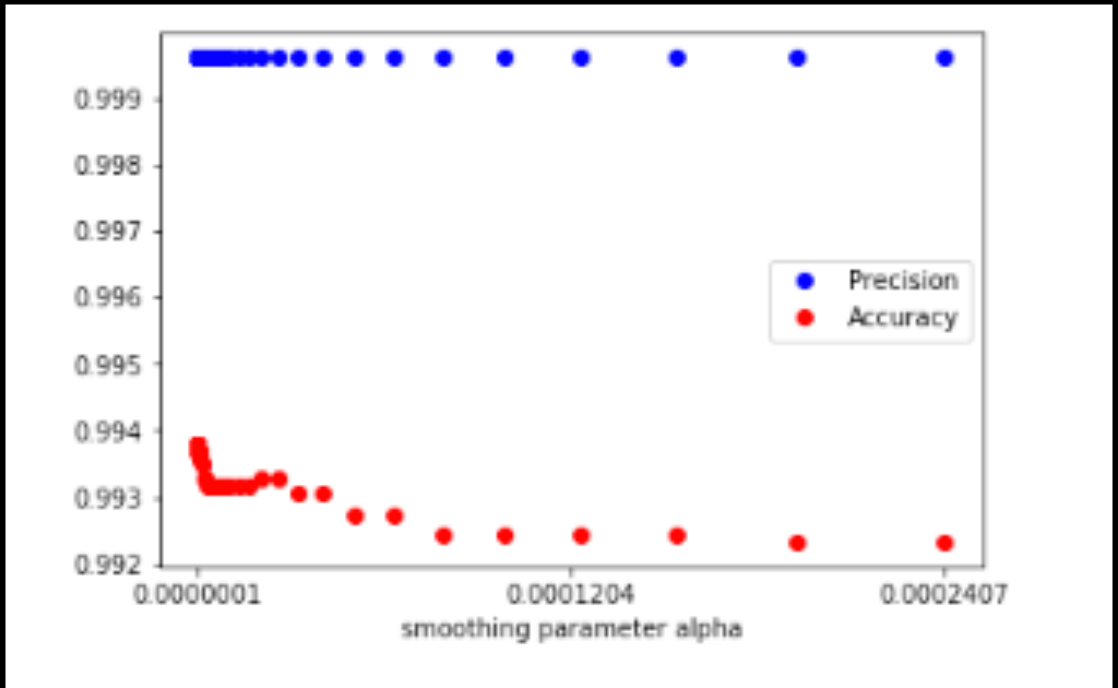
# NAIVE BAYES

0 < alpha < 1

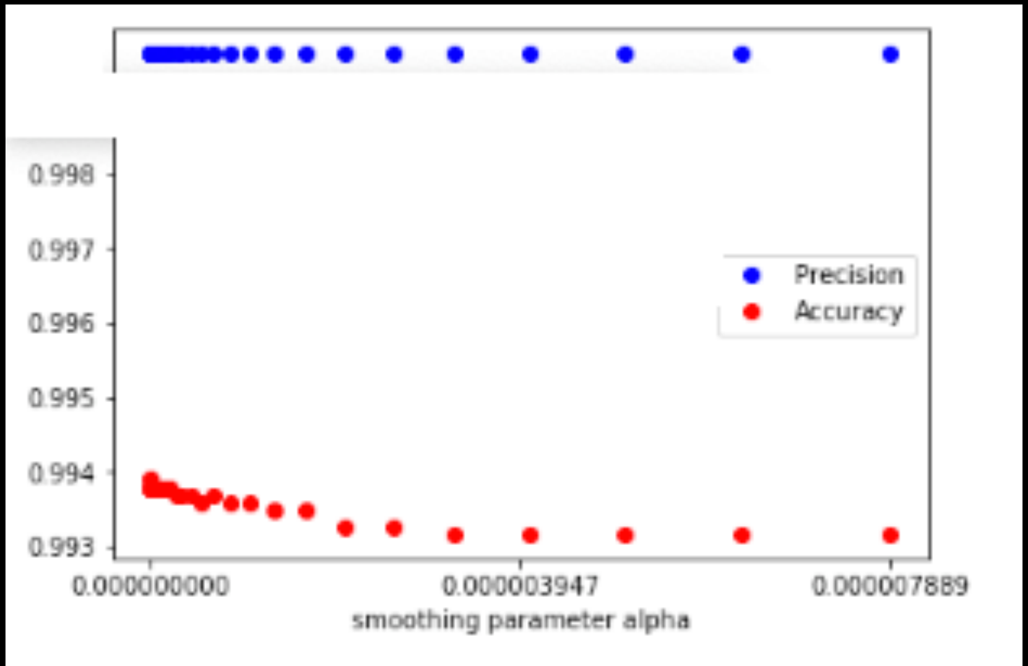



- Precision, accuracy and alpha
- Accuracy increases slightly as alpha approaches 0.
- We don't want a very small alpha.

alpha drops when > 10^-7




Good NB classifiers with low alpha:



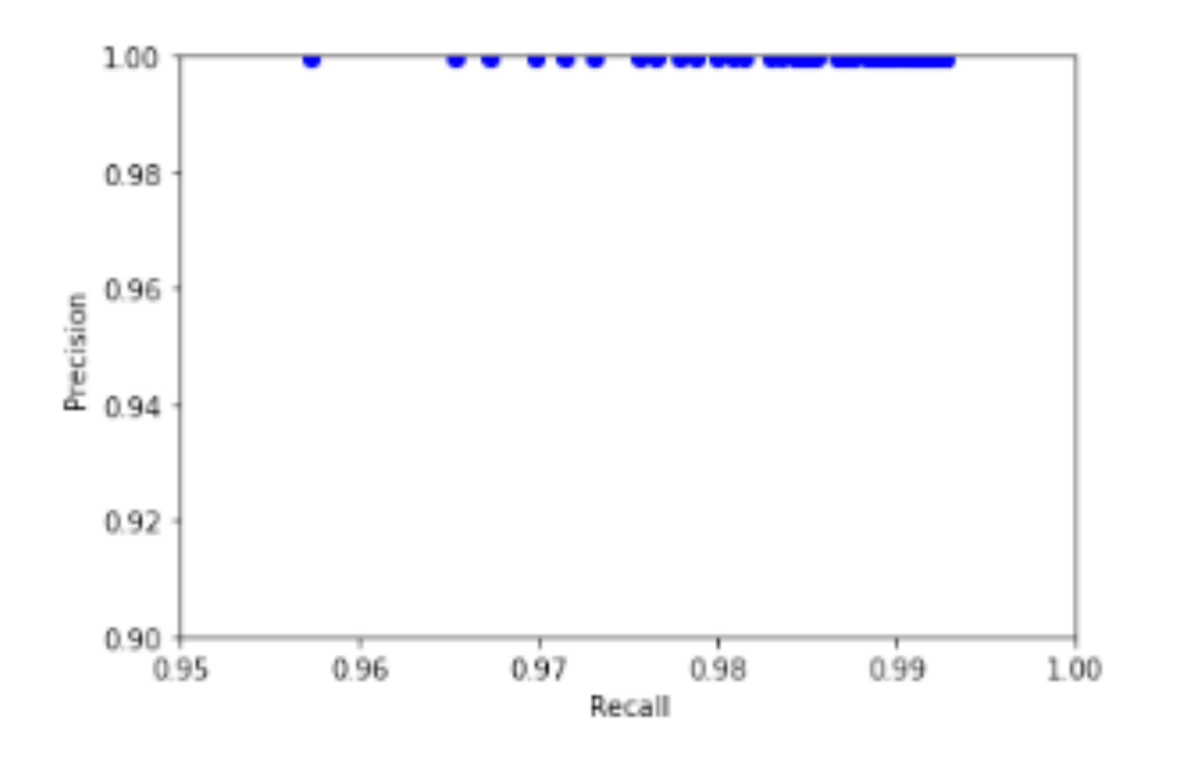
# NAIVE BAYES

- Grid search for parameters that fulfil precision requirements
  - Best  =  Minimum false positive rate and maximum accuracy.
    - If many alphas are as good, choose the biggest one.
  - Best result:   alpha = 2*10^-7
  - Average results in cross-validation:
  - 99.96% Precision, 99.5% accuracy, 0.15% FPR, 99.5% TPR
  - Results on test set:
  - 100% precision,  99% accuracy,  0% FPR,  98.5% TPR

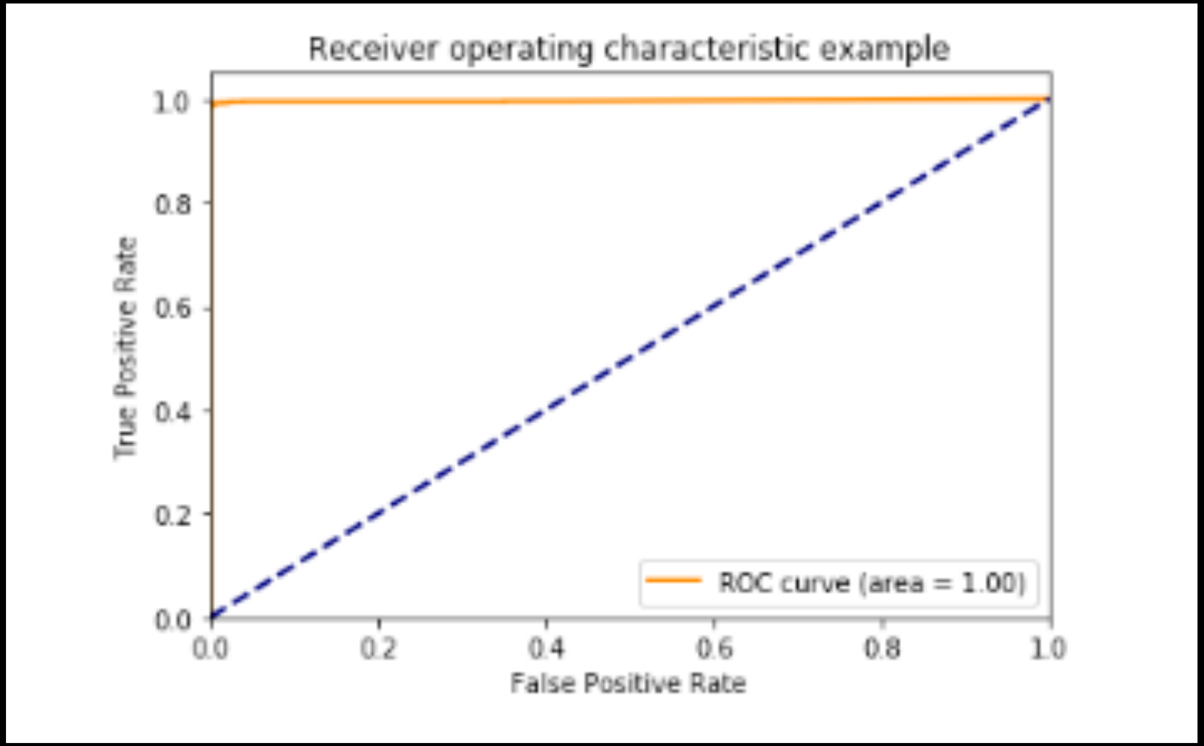
Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 25 | 1581 |

Precision / Recall for various alpha




ROC curve for one of the good classifiers



## Optimal NB classifier found with grid search

Later, after getting used to doing grid search in sklearn, I found a very similar optimal classifier with few lines of code and quick computation:

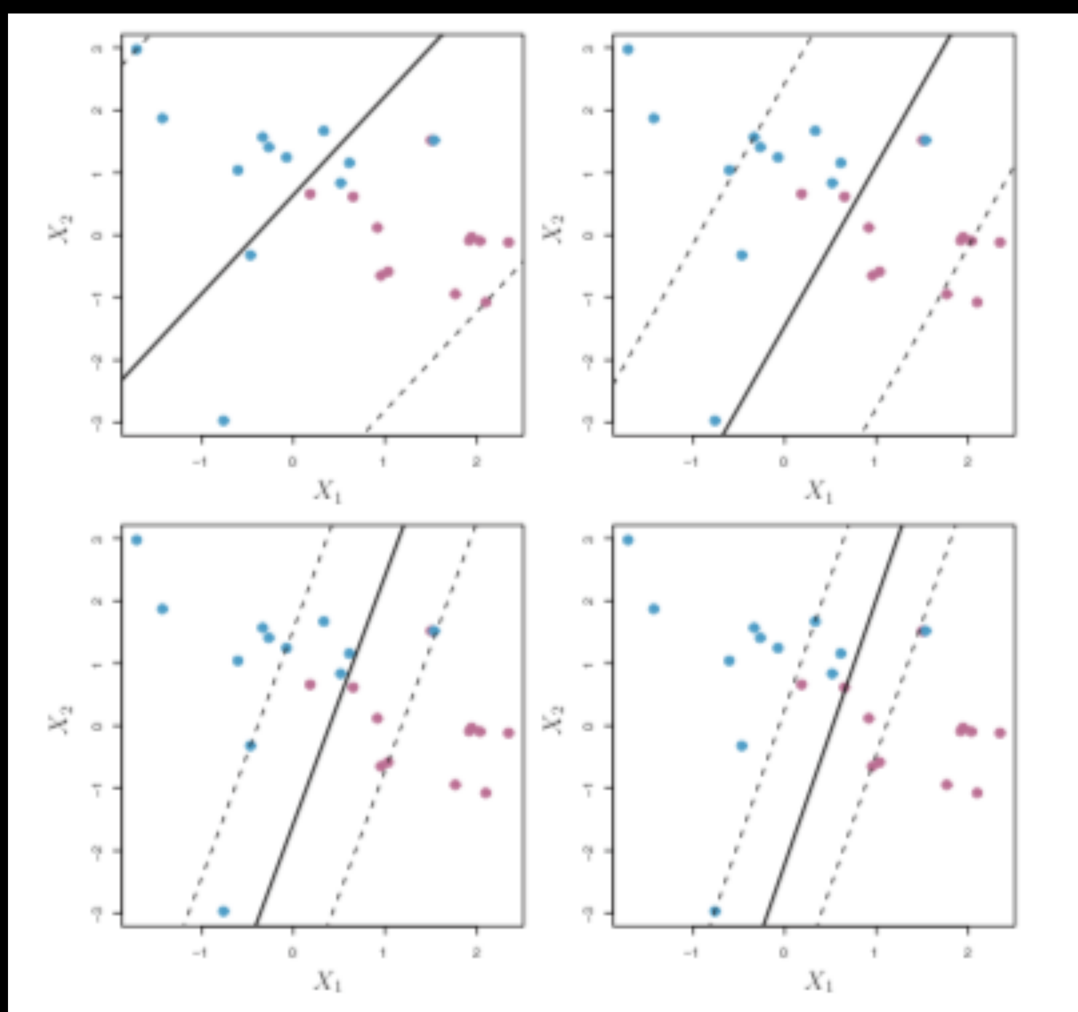
```
params_to_check = param_range(5e-8,7e-3, 4)

random_grid = {'alpha': params_to_check}

nb_random = RandomizedSearchCV(estimator = MultinomialNB(), param_distributions = random_grid,
                               scoring = scorer, n_iter = 10, cv = 5, random_state = 42, n_jobs = -1)

nb_random.fit(big_train_X, big_train_y)
```

```
show_best(nb_random)
```

```
[[ 394   25]
 [   0 1581]]
MultinomialNB(alpha=9.094947017729282e-07, class_prior=None, fit_prior=True)
```

# SUPPORT VECTOR MACHINES

Optimal hyperplanes, different C




Optimisation criteria (linear)

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,M}{\text{maximize}} \quad M \tag{9.12}$$

$$\text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.13}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i), \tag{9.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C, \tag{9.15}$$

The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$. $d$ is specified by keyword `degree`, $r$ by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$. $\gamma$ is specified by keyword `gamma`, must be greater than 0.
- sigmoid ($\tanh(\gamma \langle x, x' \rangle + r)$), where $r$ is specified by `coef0`.

- Hyper-parameters:
  - C ( sum of slack variables = margin violations) - “soft margin”
  - Kernel function (basis function / feature mapping, measures similarity )
  - Gamma (parameter in Kernels)
  - Class weight (how much C is allowed for each class)
- I programmed the grid search for the hyperparameters like I did for Naive Bayes.
  - Unlike Naive Bayes, the search was multidimensional and time consuming.

- Steps made in search of a good classifier:
  - 1. Try some random configurations of C, kernel function and class weights.
  - 2. For good results from 1, search for good hyper-parameters close to them.
  - 3. Search a wide range of values of C and class weights and different Kernel functions.
  - 4. Search even more values of C, class weights, along with Gamma, for the Kernel function found in 1.

# SUPPORT VECTOR MACHINES

Experiment 1

The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$. $d$ is specified by keyword `degree`, $r$ by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$. $\gamma$ is specified by keyword `gamma`, must be greater than 0.
- sigmoid ($\tanh(\gamma \langle x, x' \rangle + r)$), where $r$ is specified by `coef0`.

- Tried different values for the slack sum **C** and for the ‘class_weight’ parameter to tune the False Positive Rate.
- Trained SVMs with C = (0.5, 1.5, 5,10) and class_weight for non-spam as (3,5,7,10,100) with the four different kernels
  - Training took long (up to 1 minute per training), compared to NB which was fast.
- Best results found:
  - **Sigmoid** kernel, class_weight = 5 and C = 0.5.

# SUPPORT VECTOR MACHINES

- Experiment 2
- Searched the space around the hyper-parameters whose models had good results ( <0.2% FPR and >98% accuracy).
  - Only the **Sigmoid** kernel had good results,  with C = **0.5** and weights **5** and **7.**
- Searched close by it and found an OK model:
  - SVC(C=0.536, kernel='sigmoid', class_weight={1: 1, -1: 4.571},

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 56 | 1550 |

  - No real mail classified as spam, which is good.
  - 100% precision, 97.2% accuracy.
  - Double the number of misclassification compared to the best Naive Bayes classifier.

# SUPPORT VECTOR MACHINES

- Experiment 3
- Tried out many values for 'C' and 'class_weight' for the different Kernels.
  - Put strong constraints on accuracy for all 5 cross validation splits (>98%), along with the min false alarm rate (<0.2%).
  - Trained models using 30 different values of C and 22 class_weights for all 4 types of Kernels.
  - Took 1 hour. Found no good classifiers. Maybe the constraint on accuracy was too strong?

# SUPPORT VECTOR MACHINES

- Experiment 4: Big exhaustive search
- Tried even more values, varying 'C' and 'gamma',
    - with 'class_weight' either the best value found so far or equally balanced.
    - Only trained on the Sigmoid kernel.
    - Put weaker constraints than last time on accuracy for all 5 cross validation splits (>96%), along with the min false alarm rate (<0.2%).
    - After running the program for 14 hours while finding only 1 good candidate (similar to the NB model) out of thousands of models, I **gave up**.

- Experiment 4: Big exhaustive search
- Comments:
  - Should have put more thought into choosing the range of parameters to
  - Didn't know before what gammas were good.
    - Probably didn't need to train on gamma = 4*10^-6 when 2*10^-6 didn't give any good results.
  - The "slack budget" C shouldn't have been allowed to be very high.
    - Because we didn't want any false positives.
  - Next time: Use **Randomized Grid Search** from SKLearn!

# SUPPORT VECTOR MACHINES

- Best result that was found using SVMs:
- SVC(C=0.536, kernel='sigmoid', class_weight={1: 1, -1: 4.571}
    - 100% precision, 97% accuracy, 0% FPR, 96% recall.
-

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 56 | 1550 |

# RANDOM FORESTS

- Ensemble method
- Forests of randomised decision trees
  - Bagging (bootstrap aggregating)
    - Averaging a set of predictions reduces variance
    - Majority vote
  - The algorithm is not even allowed to consider most of the available features
    - De-correlates the trees
  - Like asking 10 different people for advice who ask you different yes/no questions

# Random Forests 🌳 🌳 🌳

- This time didn't waste energy hard-coding,  only used built-in methods from scikit-learn.
- Baseline model: Default RandomForestClassifier()
- Baseline results:
  - 99.7% accuracy  and  99.4% precision.     1.4% false positive rate!
- Problem:   There are so many hyper-parameters to try out that it would take ages to do an exhaustive search for a great configuration.

```
Out[1095]: {'bootstrap': True,
            'class_weight': None,
            'criterion': 'gini',
            'max_depth': None,
            'max_features': 'auto',
            'max_leaf_nodes': None,
            'min_impurity_decrease': 0.0,
            'min_impurity_split': None,
            'min_samples_leaf': 1,
            'min_samples_split': 2,
            'min_weight_fraction_leaf': 0.0,
            'n_estimators': 'warn',
            'n_jobs': None,
            'oob_score': False,
            'random_state': None,
            'verbose': 0,
            'warm_start': False}
```

# Random Forests 🌳 🌳 🌳

- Strategy: Randomly search for optimal configurations of various parameters using RandomizedSearchCV().
- Parameters to vary:
  - class_weight, max_depth, max_features, min_sample_split, min_sample_leaf, bootstrap
- First try:
  - Used the default scoring function in RandomizedSearchCV, for evaluating what is best.
  - After 10 iterations the best estimator found had 1.5% FPR and 99.9% TPR.

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 388 | 6 |
| Spam | 2 | 1604 |

# Random Forests 🌳 🌳 🌳

- I pursued 5 strategies:
  - 1. Find a classifier with 100% precision and search for a better one close to it (with similar hyper-parameter configuration)
  - 2. Find a classifier with 100% recall and search for a better one close to it.
  - 3. Find a classifier with very high accuracy and
    - a) look for a 0% FPR classifier close to it
    - b) post-process its output such that an email is classified as spam only if the classifier's predicted probability is high
  - 4. Random search in a big hyper-parameter space for classifiers with 0% FPR and maximum accuracy
  - 5. Exhaustive search in the vicinity of all the hyper-parameters of the best classifiers found in steps 1-4 for a 0% FPR and maximum recall classifier.

## Random Forests 🌳 🌳 🌳

- Experiment 1: Look for a classifier with no false positives.
  - Ran it this time with precision as scoring (evaluation metric) for the search and found a classifier with no false positives.

Predicted

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 410 | 1196 |

  - Next step: Search hyper-parameters close to it for classifiers with better recall.

# Random Forests 🌳 🌳 🌳

- Experiment 1: Search hyper-parameters close by for classifiers with better recall.
- Customised a scoring function that gives 0 score when there are more than 0.2% FPR.
  - Score function maximises,    Loss function minimises
- Found one classifier with 100% precision, 87% accuracy, 0% FPR and 84% TPR/recall.
-

```python
def my_score_function(y_true, y_predict):
    """
    Use:
        score = my_score_function(y_true,y_predict)
    Before:
        y_true are true labels, y_predict predicted labels
    After:
        score is an evaluation of how good the prediction was
        (only good if the fpr was adequate and recall is good)
    """
    con = confusion_matrix(y_true,y_predict)
    precision, accuracy, fpr, recall = conf_interpretation(con)
    if fpr <= 1/500:
        score =5* recall
    else:
        score = 0

    return score
```

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 249 | 1357 |

- Experiment 2: Search for a classifier with no false negatives.
  - Found one when running 10 iterations using 'roc_auc' scoring.
  - 100% TPR/recall, 98.7% accuracy, but 6.6% FPR!

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 368 | 26 |
| Spam | 0 | 1606 |

  - Idea: Search around it for more precise classifiers.

# Random Forests 🌳 🌳 🌳

- Experiment 2: Found a classifier with no false negatives.
- Idea: Search around it for the classifier that maximises recall, with <=0.2% FPR.
- Results:

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 57 | 1606 |

- Further idea to do: Exhaustive search around there?

# Random Forests 🌳 🌳 🌳

- Experiment 3: Search for a high accuracy classifier.
- Ran it this time for 600 iterations, with 'roc_auc' scoring, to find an accurate classifier.
  - With 1.7% FPR, the result didn't satisfy our minimum requirements.
  - Accuracy of 99.5% is great, but it doesn't take into account the severity of a false alarm.

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 387 | 7 |
| Spam | 2 | 1604 |

  - Next steps:
  - a) Search around it to find one with <0.2% FPR.
  - b) Post-process the outcome by only classifying as Spam if probability is > threshold

## Random Forests 🌳 🌳 🌳

- Experiment 3a: Search around the high accuracy RF to find one with <= 0.2% FPR.
- Found one, at the cost of decreased accuracy.
- Average of 99.9% precision, 98% accuracy, 0.2% FPR and 97.5% TPR/recall.

Predicted:

| | Ham | Spam |
|---|---|---|
| Ham | 393 | 1 |
| Spam | 40 | 1566 |

# Experiment 3b - post-processing the high accuracy classifier

- I tried to tune the FPR manually by post-processing the class predictions:
  - Have classification remain as Spam only if the predicted probability from the model is higher than a certain threshold.

```
def adjusted_classes(spam_probabilities,t):
    return [1 if y >= t else 0 for y in spam_probabilities]

# Example, prediction as Spam only if probability is over 0.7:
predicted_spam_p = clf.predict_proba(X_test)[:,1]
adjusted_classes(predicted_spam_p, 0.7)
```

  - Found that by increasing the threshold from 0.50 to 0.52, the single false positive disappeared when testing on the test set.
  - Threshold 0.50

Predicted

| | Ham | Spam |
|---|---|---|
| Ham | 393 | 1 |
| Spam | 40 | 1566 |

Threshold 0.52

Predicted

| | Ham | Spam |
|---|---|---|
| Ham | 394 | 0 |
| Spam | 40 | 1566 |

## Random Forests 🌳 🌳 🌳

- Experiment 4: Random search over a big grid with custom scoring to find classifiers with 0% FPR and high accuracy.
  - Score criteria:   <= 0.2% FPR false positives, maximum accuracy.
  - Tried 3 times:
    - A quick search found a classifier with 0 FP but 93% recall, which is not adequate.
    - Longer random searches of 300 and 600 iterations. Found classifiers with 0 FP but 90% and 87% recall.
- With TF-IDF:
  - Realised I needed to try something else. Pre-processing with TF-IDF improved it to 95% recall

- **Experiment 5**: Exhaustive search for good parameters

- Ran exhaustive search overnight for combinations of parameters from the best classifiers.

| Class weight | max_depth | min_samples_split | max_features | min_samples_leaf | n_estimator |
|---|---|---|---|---|---|
| **1,3,5,7,15** | 50,55,60 70,80 | 2,5,10 | 100,100, 'sqrt' | 1,5,10 | 20,40,80,120 |

- Results:

Default scoring

Predicted

| | Ham | Spam |
|---|---|---|
| Ham | 389 | 5 |
| Spam | 2 | 1604 |

Custom scoring

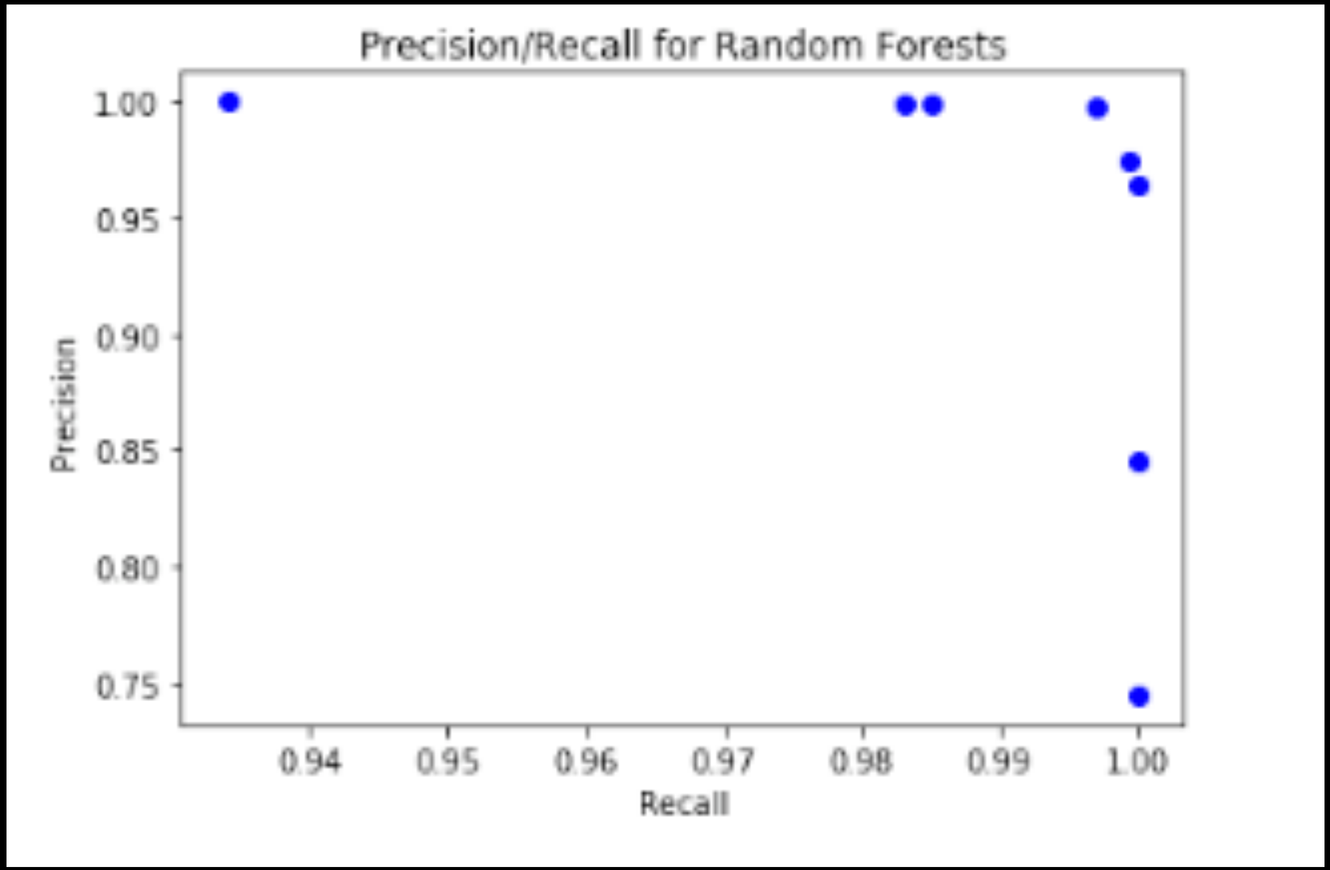
???

Predicted

| | Ham | Spam |
|---|---|---|
| Ham | 389 | 5 |
| Spam | 4 | 1602 |

# Random Forests 🌳 🌳 🌳

Average Precision/Recall for the good classifiers, from cross validation



## Best results from the different methods

| *Method* | Accuracy | Precision | Recall | FPR |
|---|---|---|---|---|
| NB | 99 % | 100 % | 98 % | 0 % |
| RF | 98 % | 100 % | 97 % | 0 % |
| SVM | 97 % | 100 % | 96 % | 0 % |

## Thoughts when preparing presentation

- Improvements?
  - Preprocess with TF-IDF
    - Did it with Naive Bayes - didn't help.
    - Tried it with Random Forests (yesterday…) and it helped
  - Define a better scoring function?
  - Run exhaustive search on a big grid for a few days?

## LESSONS LEARNED

- Just use a simple method if it works.
- Don’t waste energy programming things from scratch
- When in doubt what hyper-parameter values are good, use RandomizedGridSearch() and an appropriate scoring function to find candidates.
- Pre-processing has different effects on different methods.
- “Brute force” training all kinds of hyper-parameters can take days.
- Machine learning can be a lot of trial & error!

- *Screenshots were taken from  ScikitLearn docs and  Introduction to Statistical Learning by Hastie.*