Assignment 2

Martin Tomaschek

July 2020

1 The assignment

Write a spam filter using discrimitative and generative classifiers. Use the Spambase dataset which already represents spam/ham messages through a bag-of-words representations through a dictionary of 48 highly discriminative words and 6 characters. The first 54 features correspond to word/symbols frequencies; ignore features 55-57; feature 58 is the class label (1 spam/0 ham).

1. Perform SVM classification using linear, polynomial of degree 2, and RBF kernels over the TF/IDF representation

   Can you transform the kernels to make use of angular information only (i.e., no length)? Are they still positive definite kernels?

2. Classify the same data also through a Naive Bayes classifier for continuous inputs, modelling each feature with a Gaussian distribution, resulting in the following model:

$$p(y = k) = \alpha_k$$

$$p(x|y = k) = \prod_{i=1}^{D} \left[(2\pi\sigma_{ki}^2)^{-1/2} \exp\left\{ -\frac{1}{2\sigma_{ki}^2}(x_i - \mu_{ki})^2 \right\} \right]$$

   where α_k is the frequency of class k, and μ_{ki}, σ_{ki}^2 are the means and variances of feature i given that the data is in class k.

   Provide the code, the models on the training set, and the respective performances in 10 way cross validation.

Explain the differences between the two models.

P.S. you can use a library implementation for SVM, but do implement the Naive Bayes on your own

2 Theoretical background

2.1 Discriminative and generative models

A **discriminative model** models the decision boundary between the classes. In other words it tries to estimate the parameters of $P(Y|X)$ directly from the learning data, where X is the observation data and Y is the label.

Meanwhile **generative models** model the actual distribution of the classes as the joint distribution P(X,Y). They estimate $P(X|Y)$ and $P(X)$ from the learning data and use Bayes theorem to calculate $P(Y|X)$.

Support vector machines are an example of a dicriminative model based classifier, while Naive Bayes classifier is based on a generative model.

2.2 Support vector machines (SVM)

Support vector machines are a supervised learning algorithm, based on the idea finding the maximum margin (largest distance to examples) hyperplane (linear separator) separating the classes of a classification problem.

By finding the maximum margin separator, SVMs try to minimize the expected generalization loss. The next chapter explains how the maximum margin separator can be found.

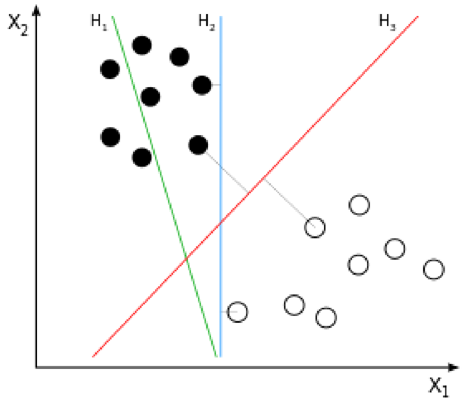


Figure 1: H1 does not separate the classes. H2 does, but only with a small margin. H3 separates them with the maximal margin. By intuition, H3 is more likely to classify new data correctly.

2.2.1 Hard margin

If the data is linearly separable, we can find two parallel hyperplanes that separate the classes and are as far apart as possible. The maximum margin separating hyperplane is thus midway between them.

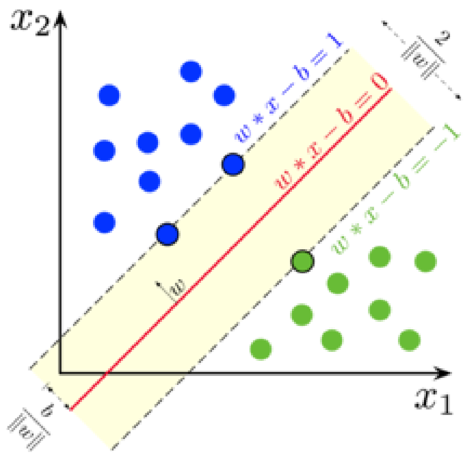


Figure 2: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

A hyperplane can be written as a set of points satisfying

$$w \cdot x - b = 0$$

where w is a normal vector of the hyperplane and $\frac{b}{|w|}$ is the offset from the origin along w.

Lets consider two hyperplanes:

$$H_1 : w \cdot x - b = 1$$

$$H_2 : w \cdot x - b = -1$$

Now we define constraints:

$$w \cdot x_i - b \leq -1 \text{ if } y_i = -1$$

$$w \cdot x_i - b \geq 1 \text{ if } y_i = 1$$

for all training points x_i with label y_i, which means there shall be no points in between the hyperplanes, and all points are on the correct side the hyperplanes. We can merge the constraints to one form

$$g_i(w) = y_i(w \cdot x_i - b) - 1 \geq 0, i = 1...n$$

To maximize the distance between the hyperplanes, which is

$$\frac{2}{|w|}$$

which equivalent to saying we must minimize $|w|$ while satisfying constraints.But instead of minimizing $|w|$ we could just as much minimize the function

$$f(w) = \frac{1}{2}|W|^2$$

subject to the same constraint, the constant is just for later convenience. This now has the advantage of being a convex quadratic optimization problem subject to linear constraints, which has a unique global optimum. This problem can be solved using a quadratic programming package.

We can alternatively try to use another formulation - the dual problem. We will apply the KKT conditions to problem. Introducing a Lagrange multiplier λ_i for each constraint (of which we have one for each training point) we get

$$\mathcal{L}(w, b, \Lambda) = f(w) - \sum_i \lambda_i g_i(w, b)$$

$$\mathcal{L}(w, b, \Lambda) = \frac{1}{2}|w|^2 - \sum_i \lambda_i [y_i(w \cdot x_i + b) - 1]$$

where Λ is a vector of said Lagrange multipliers. To obtain the solution right now we would need to minimize $\mathcal{L}$ with respect to w, b but maximize with respect to Λ, subject to $\lambda_i \geq 0$. This form is known as the primal Lagrangian. To move on we fix Λ and we apply the stationarity KKT condition (we want to select a point where the gradient of the Lagrangian vanishes):

$$\Delta_w \mathcal{L}(w, b, \Lambda) = w - \sum_i \lambda_i y_i x_i = 0$$

$$\frac{\delta \mathcal{L}()w, b, \lambda}{\lambda b} = \sum_i \lambda_i = 0$$

Substituting the above expressions into the Lagrangian we get

$$\mathcal{L}(\Lambda) = \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i x_j^T$$

with the constraints ($\lambda_i \geq 0$) (dual feasibility KKT condition) and $\sum_i \lambda_i y_i = 0$ in addition to the primal feasibility condition (the constraints of the primal problem) we already had.

Thus we have arrived at the Wolfe dual Lagrangian problem:

$$\arg\max \Lambda sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i x_j^T$$

subject to already stated constraints. Because the KKT conditions in addition to Slater's condition hold so the duality gap is 0.

In this formulation of the problem the data enter in form of a dot product of pairs of points, which is an important property that will come in play later.

After the optimal solution has been found, the Lagrangian multipliers λ_i are zero except for the examples on the margin. These are called the support vectors and there are usually much fewer of them then the examples in the dataset - other examples from the training set can be forgotten, reducing trained model size.

Now w and b can be computed:

$$w = \sum_i \lambda_i y_i x_i$$
$$b = \frac{1}{S} \sum_{i \in S} y_i - w x_i$$

where S is the number of / are the support vectors.

The hypothesis function (the class of a example) then is:

$$h(x) = \operatorname{sign}\left(w x_i (x - b)\right.$$

or

$$h(x) = \operatorname{sign}\left(\sum_j \alpha_j (x \cdot x_j) - b\right)$$

using the dual formulation.

2.2.2 Soft margin

Often data is not entirely linearly separable, for example a few outliers are present. In such case we can not find a hard margin and we would like to allow some missclassification in exchange for some kind of penalty.

We can do so by softening the constraint by allowing points to fall within the margin or completely on the wrong side of it. The distance from the margin is than used as a measure of missclassification. We call this distance as ζ_i, also know as the slack variable.

The minimization problem then becomes

$$\operatorname*{arg\,min}_{w,b,\zeta} \frac{1}{2} w^2 + C \sum_i \zeta_i$$

subject to $(y_i(w \cdot x_i - b) \geq 1 - \zeta_i)$ and $\zeta_i >= 0$

The coefficient C then governs the trade-off between missclassification error on the training set and width of the margin. A good value for C depends and the data, it is usually tuned using crossvalidation.

Strangely we arrive at the same dual formulation as before

$$\arg\max \Lambda sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i x_j^T$$

but now constrained by $0 \leq \lambda_i \leq C$ and $\sum_i \lambda_i y_i = 0$.

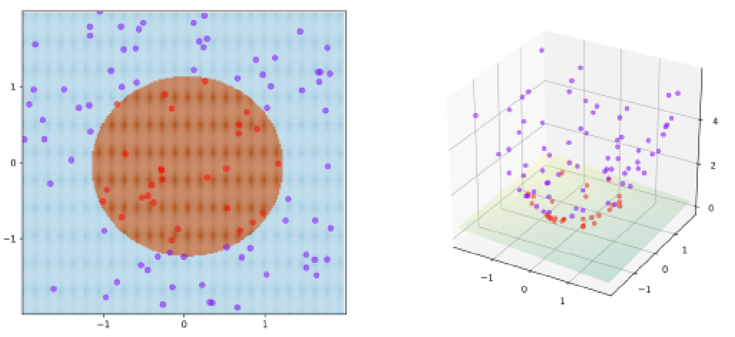


Figure 3: A training example of SVM with kernel given by $(\varphi((a, b)) = (a, b, a^2 + b^2))$.

2.2.3 Kernel trick

While soft-margin SVM might be able to find workable separator in case of a little noise or outliers present in the data, it doesn't help when the data is not linearly separable at all.

To deal with this issue SVMs employ so called kernel trick. The kernel trick enables the computation of a dot product of points in other, higher dimensional space (possibly infinite), using only inputs from the original space (without explicitly computing the mapping). Data that is not linearly separable in the original space, is more likely linearly separable in the higher dimensional space (Cover's theorem).

To use a kernel, one just needs to replace the dot product of the feature vectors with a kernel function of those vectors in dual formulation of the SVM optimization problem. Kernel function must behave as a dot product in some space - $K(x, y) = \varphi(x)^T \varphi(y))$, existence of such a mapping is guaranteed by Mercer's theorem in case of positive definite symmetric kernels - whose kernel matrix is symmetric semi-definite. A a kernel can be though of as a similarity measure.

Among the most commonly used kernels are:

- linear kernel: $K(x, y) = x^T y$
- polynomial kernel: $K(x, y) = (c + x^T y)^d$
- Gaussian kernel/Radial basis function kernel: $K(x, y) = \exp\{-\frac{||x-y||^2}{2\sigma^2}\}$ or equivalently $\exp(\gamma||x - y||^2)$

Conceptually, the polynomial kernels considers not only the similarity between vectors under the same dimension, but also across dimensions. When used in machine learning algorithms, this allows to account for feature interaction.

The Gaussian kernel measures the similarity of points based on Euclidean distance, with the similarity exponentially decreasing form 1 for identical points to almost 0 for distant points.

2.2.4 Normalization

Let's define normalized kernel as

$$K'(x, y) = \begin{cases} 0, & \text{if } K(x, x) = 0 \vee K(y, y) = 0 \\ \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}}, & \text{otherwise} \end{cases}$$

where K is a positive definite symmetric kernel. We can prove k' is still a positive definite symmetric kernel by using Mercer's condition

$$\sum_{i,j=1}^{m} \frac{c_i, c_j K(x_i, x_j)}{\sqrt{K(x_i, x_i)K(x_j, x_j)}}$$

expanding the kernels K as dot product of feature maps $K(x, y) = \langle \Phi(x), \Phi(y) \rangle$, note that $K(x, x) = \Phi(x)^2$

$$\sum_{i,j=1}^{m} \frac{c_i c_j \langle \Phi(x_i), \Phi(x_j) \rangle}{\sqrt{\Phi(x_i)^2 \Phi(x_j)^2}} = \sum_{i,j=1}^{m} \frac{c_i c_j \langle \Phi(x_i), \Phi(x_j) \rangle}{\|\Phi(x_i)\| \|\Phi(x_j)\|}$$

Because the sum iterates over the same range we can rewrite it this way

$$\left\| \sum_{i=1}^{m} \frac{c_i \Phi(x_i)}{\|\Phi(x_i)\|} \right\|^2 \geq 0$$

so now we can see that Mercer's condition hold's and the kernel is positive definite symmetric.

Thus we have performed kernel space normalization where all point's have been mapped to the surface of a unit sphere in the space in which the dot product is computed by the kernel. This normalization is sometimes called cosine normalization because when we replace the dot product with it's geometrical interpretation $\langle x, y \rangle = \|x\| \|y\| \cos \Theta$ we get

$$K'(x, y) = \frac{K(x, y)}{\sqrt{K(x, x)K(y, y)}} = \frac{\langle \Phi(x), \Phi(y)}{\|\Phi(x)\| \|\Phi(y)\| \rangle} = \frac{\|\Phi(x)\| \|\Phi(y)\| \cos \Theta}{\|\Phi(x)\| \|\Phi(y)\|} = \cos \Theta$$

Thus we can see that normalized kernels depend only on the angle between the two vectors mapped to kernel's space. Interestingly, normalization has no effect on RBF kernel - it is already the result of normalization of the kernel $K(x, y) = \exp \frac{\langle x,y \rangle}{2\sigma^2}$.

We can also apply normalization in the input space by dividing each input vector by its norm. This is be equal normalizing the kernel for some kernels, notably polynomial kernels without the constant r (i.e. mononomial, including the linear case) $K(x, y) = \langle x, y \rangle^d$ because:

$$K'(x,y) = \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}} = \frac{\langle x,y\rangle^d}{\sqrt{\langle x,x\rangle^{2d}\langle y,y\rangle^{2d}}} = \frac{\langle x,y\rangle^d}{\|x\|^d\|y\|^d} = K\left(\frac{x}{\|x\|},\frac{y}{\|y\|}\right)$$

unfortunately this does not seem to be true for the general polynomial kernel and RBF kernel, so the mappings in the kernel space are not normalized. Nonetheless this prepossessing technique seems to be quite widely used, more so than kernel normalization, probably because they are equivalent in the most common cases (or kernel normalization is unnecessary as is the case of RBF kernel).

2.3 Gaussian Naive Bayes

Naive Bayes classifier family uses the Bayes theorem to calculate the posterior probability $p(C_k \mid \mathbf{x})$ (probability that the class label is C_k given some evidence $\mathbf{x}$

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

The numerator is equivalent to the joint probability

$$p(C_k, x_1, \dots, x_n)$$

this can be expanded using the chain rule[1] for repeated application of the definition of conditional probability.

$$\begin{aligned}
&p(C_k, x_1, \dots, x_n) = \\
&= p(x_1, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\ p(x_2, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\ p(x_2 \mid x_3, \dots, x_n, C_k)\ p(x_3, \dots, x_n, C_k) \\
&= \dots \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\ p(x_2 \mid x_3, \dots, x_n, C_k) \dots p(x_{n-1} \mid x_n, C_k)\ p(x_n \mid C_k)\ p(C_k)
\end{aligned}$$

This method is called naive, because it assumes conditional independence of the variables given a class label. While such assumption rarely holds, despite this, in practice naive Bayes classifiers perform well. Based on this assumption

$$p(x_i \mid x_{i+1}, \dots, x_n, C_k) = p(x_i \mid C_k)$$

Now we can rewrite the numerator as

$$p(C_k, x_1, \dots, x_n) = p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

[1]permits the calculation of any member of the joint distribution of a set of random variables using only conditional probabilities: $\mathrm{P}\left(\bigcap_{k=1}^n X_k\right) = \prod_{k=1}^n \mathrm{P}\left(X_k \mid \bigcap_{j=1}^{k-1} X_j\right)$

The denominator is (expansion through law of total probability):

$$p(\mathbf{x}) = \sum_k p(C_k)\ p(\mathbf{x} \mid C_k)$$

putting everything together

$$\begin{aligned} p(C_k \mid \mathbf{x}) &= \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})} \\ &= \frac{p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)}{\sum_k p(C_k)\ p(\mathbf{x} \mid C_k)} \end{aligned}$$

Now we can compute the posterior probability for each class, and to classify an we just pick the maximum, and if we do not care about the probability, we do not even have to compute the denominator, because it is the same for every class. Thus the naive Bayes classifier is:

$$y = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

2.3.1 Gaussian naive Bayes

The naive Bayes classifier that we have introduced in previous chapter is still incomplete - it needs to know the class priors, which are usually taken to be the relative frequencies of the classes in the training set, or sometimes set to equal share per class if the classifier being a priori biased is a problem, and it needs to know the conditional probabilities $p(x_i \mid C_k$.

When dealing with continuous data the assumption is often that the values each variable takes for each class are distributed according to a Gaussian distribution. So we will estimate $p(x = v \mid C_k)$ using probability density function of normal distribution:

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

To estimate the σ_k^2 and μ we split the training data by class and compute the sample means and variances.

2.3.2 Multinomial naive Bayes

This variation of naive Bayes be well suited for text classification tasks, so I wanted to give it a try. It models the conditional probabilities $p(x_i \mid C_k$ using multinomial distribution, which depicts the frequencies of occurrences of each of n events out of m tries (generalization of the binomial distribution to n events) . The feature vectors are then viewed as histograms, counting such events. Than the likelihood that the example $x = (x_1...n)$ was generated by the multinomial with event probabilities $(p_{k1}....p_{kn}$ for respective events for the class C_k, is given by its probability mass function (discrete probability density):

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

applying the Bayes theorem to compute the posterior probabilities

$$\begin{aligned} p(C_k \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid C_k) p(C_k)}{p(x)} \\ &= \frac{\frac{(\sum_i x_i)!}{\prod_i x_i!} p(C_k) \prod_i {p_{ki}}^{x_i}}{p(x)} \end{aligned}$$

Now just as before we do not need any parts of the expression that do not depend on C_k, for determining the maximum probability out of the classes because these parts are the same for each class.

The estimates and of p_{ki} can be computed by separating by class and computing per feature means, then computing the probability p_{ki} from that mean, deriving from the formula for the expected value of event i for multinomial distribution, as:

$$p_{ki} = \frac{\bar{X_{ki}}}{\sum_j X_j}$$

or in plain words as the relative frequencies of x_i in an average of the histograms.

The problem is that if a particular combination of feature and class is always 0, that is the feature and class never appear together in the training data the particular estimate for p_{ki} will be also 0, which would make the whole product 0 despite the contribution of other functions. To combat this, a small value is added to each feature when computing the relative frequencies, this is called is called Laplace smoothing when the value added is one, and Lidstone smoothing in the general case.

Originally intended for uses with the bag of words model, this classifier can potentially perform well when used with the TF-IDF representation.

2.4 TF, TF-IDF

Term frequency (TF) is a numerical statistic that is intended to reflect how important a word is to a document. It is the ratio of the number of occurrences of a term to all terms present in a document.

But some words occur in many more documents, being less specific. Thus the term frequency – inverse document frequency (TF-IDF) statistic weights term frequencies by logarithmically scaled inverse fraction of the documents that contain the word to all the documents.

$$\begin{aligned} TF - IDF(t, d, D) = TF(t, d) \cdot IDF(t, D) = \\ \frac{count(x == t \; for \; x \in d)}{size(d)} \cdot \log \frac{count(t \in x \; for \; x \in D)}{size(D)} \end{aligned}$$

where t is the term, d the document and D the whole document set.

3 Experimental results

3.1 Gaussian Naive Bayes Classifier implementation

This classifier is really easy and straightforward to implement, the code is in the file `naive_bayes.py`. The only hiccup along the way, was the zero division error encountered on evaluating the probability density function when the variance of a feature for given label is zero. To circumvent this issue a small value (let's call it ϵ) is added to the variance. Luckily we have a known correct implementation (`GaussianNB` class from `sklearn.naive_bayes`), we can compare to.

Scikit GaussianNB	0.8458	0.8598	0.8543	0.8413	0.6902
Our GaussianNB	0.8458	0.8598	0.8576	0.8359	0.7033

Table 1: 5-fold cross validation accuracies of 2 Gaussian naive Bayes classifier implementations on the spambase data set.

The classification performance of the two implementations, as seen in table1 is nearly identical, which suggests that our implementation is correct. I believe, that the small differences could be explained by the method for picking ϵ. I used just a simple constant, meanwhile scikit uses a more sophisticated approach[2].

Implementation	Data structure	Fitting time	Scoring time
Scikit GaussianNB	NumPy array	0.0092	0.0024
Our Gaussian NB	NumPy array	0.7517	0.5160
Our Gaussian NB	Python list	0.1831	0.1732

Table 2: Comparison of the speed of 2 Gaussian naive Bayes classifier implementations on the spambase data set. The values given are the means of a 5-fold cross validation run. Time in seconds.

Table 2 shows that our implementation is very slow in comparison to scikit's, especially if the processed data is contained in NumPy arrays, which only provide slow element access because of the more complex indexing logic, creation of new Python objects representing the element of the array and other overhead.

3.2 SVM kernels

Next, I have tried using the 3 kernels on simple problems in 2 dimensional space in order to get my head around the way they work and how they are affected by their hyper-parameters (figure 4).

The linear kernel behaves intuitively, but of course doesn't work well on datasets, that are not linearly separable. In the second case, in which one class completely encircles the other, the algorithm chooses to classify all the points as the same class, this probably means that the w vector was chosen to be almost

[2]Portion of the largest variance of all features is added to variances for calculation stability.

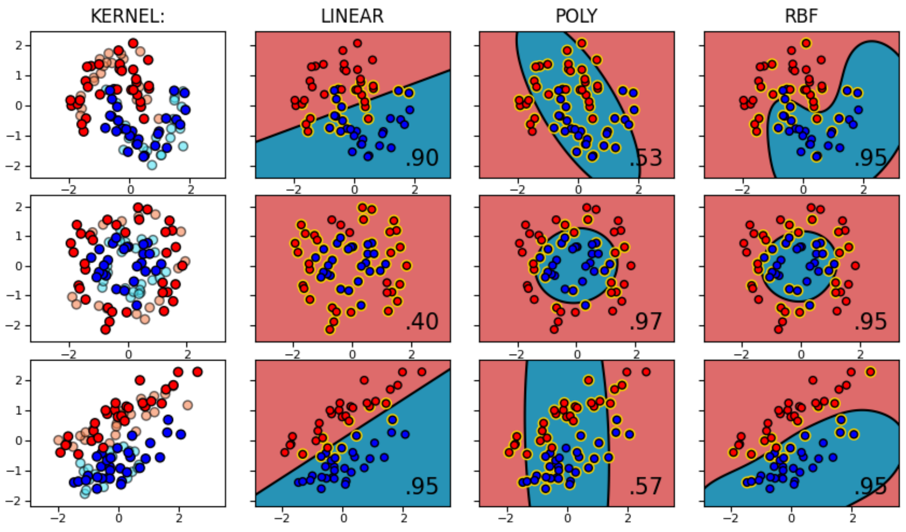


Figure 4: Plots of the decision boundaries produced by SVMs using differnt kernels (linear, polynomial of second degree and RBF, with SCIKIT's default settings for other parameters) on simple generated datasets. Datasets are plotted in the first column, the darker colored points form the training set, the lighter colored form the test set. Support vectors (or rather the pre-images of) are circled yellow. The number in the corner is the classification accuracy calculated on the test set.

zero lenght vector, providing an extremely large margin, but also extremely useless, as the other class in its entirety lies on the wrong side of the dividing hyperplane (if a high regularization via C is applied, the solution would change to one that effectively maximizes the proportion of the samples on the correct side). Such a huge margin also makes almost all the points to be support vectors. A similar situation is more likely occur in case of imbalanced datasets (scikit's SVM allows user to supply sample/class weighs to compensate).

The polynomial kernel delivered excellent performance on the second dataset, but unexpectedly struggled on the other two. I believed that polynomial kernels of higher degree allow a superset of hypotheses allowed by a kernel of lesser degree (linear kernel can be considered a polynomial kernel of degree 1), so the higher degree kernel, using the suitable parameters, should be able to mach the performance of a lower degree kernel. Changing the C parameter didn't help, nor did changing the degree. Skimming through multiple tutorials and lectures on SVMs, I've noticed that the actual formulation of the polynomial kernel varies from source to source - form simple $K(x,x') = \langle x, x'\rangle^D$ to more general $K(x,x') = (\gamma\langle x, x'\rangle + r)^D$ [3], adding two more parameters. The effect of these are overlooked in tutorials and scikit, which implements both, only has a short entry in the documentation about these. The γ is sometimes referred to as 'slope' and I didn't investigate this parameter too much, but it seems to me

[3]The course slides use $K(x,x') = (\langle x, x'\rangle + 1)^D$

as sort of inverse of regularization - I suspect that higher values produce more 'extreme' decision boundaries, which in input space I would describe as having higher bumps for lack of my understanding and descriptive skills. However the default value choosen by a heuristic in scikit implementation seems to work reasonably well. The problem was in the default value of $r = 0$, called the offset. Wikipedia page on polynomial kernel describes the effect of this parameter as "a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial". Looking at the provided expansion of the kernel for $d = 2$ (without γ parameter)

$$K(x, x') = \left(\sum_{i=1}^{n} x_i x'_i + r\right)^2 =$$

$$\sum_{i=1}^{n} \left({x_i}^2\right)\left({x'_i}^2\right) + \sum_{i=2}^{n}\sum_{j=1}^{i-1}\left(\sqrt{2}x_i x_j\right)\left(\sqrt{2}x'_i x'_j\right) + \sum_{i=1}^{n}\left(\sqrt{2r}x_i\right)\left(\sqrt{2r}x'_i\right) + r^2$$

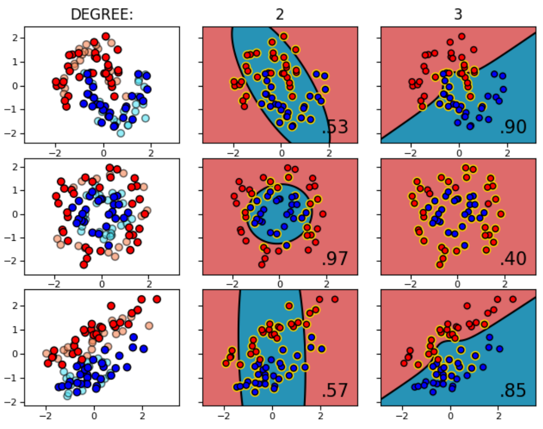
which corresponds to this feature map

$$\begin{aligned}\varphi(x) = \langle x_n^2, \ldots, x_1^2, \sqrt{2}x_n x_{n-1}, \ldots, \sqrt{2}x_n x_1, \\ \sqrt{2}x_{n-1}x_{n-2}, \ldots, \sqrt{2}x_{n-1}x_1, \ldots, \sqrt{2}x_2 x_1, \sqrt{2c}x_n, \ldots, \sqrt{2c}x_1, c\rangle\end{aligned}$$

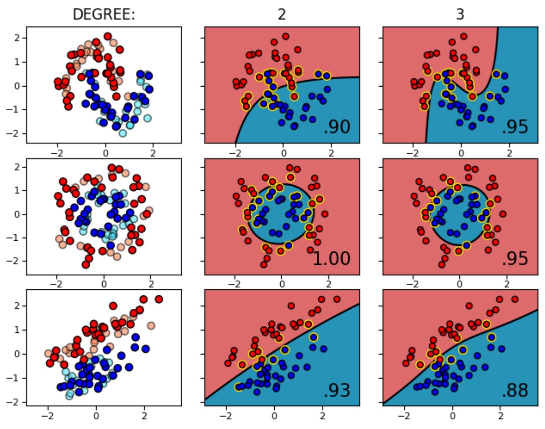
one can see that setting $r = 0$ effectively masks the first order features and only the 'interaction' features are used. It might be inaccurate, but I think polynomials of one variable might be a good analogy - if the polynomial has just one term it has one of these general shapes - a straight line, parabola-like, cubic-curve like. Adding the lover degree terms allows for more complex shapes, with more 'bumps' bound by the degree of the polynomial. Also the behavior of odd and even degree kernels are sort of in line with this - even degree work well on the circles dataset which has a symmetric distribution of the classes around the origin, but not so on the other datasets, which have more of anti-symmetric distribution of the classes around the origin, and odd degree kernels behave in exactly the opposite way. With increasing dimensionality of the problem, this should however pose a lesser problem as the proportion of asymmetric (nor symmetric, nor antisymmetric) class distributions of all the class distributions of given dimensionality, increases (because the number of axes in which this property can be broken increases). The other thing. perhaps even more significant, is that it is easier to fit the separating hyperplane through the origin in higher dimensions, we just need to make one of them sacrificial and make the hyperplane equal 0 in that dimension.

Anyway changing r to 1 solves the issue and so does adding a dummy feature of all ones to the dataset, which should have the similar effect (figure 5), giving the same mapping upon expanding the kernel.

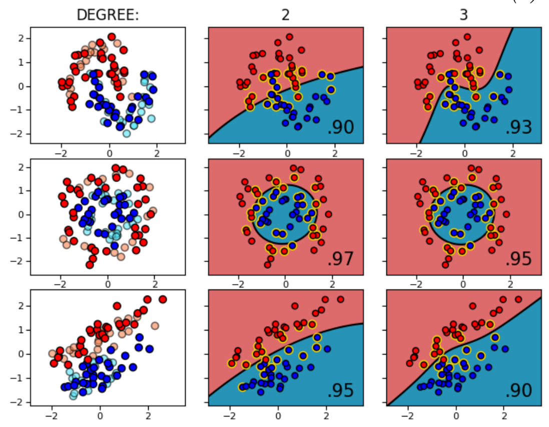
The RBF kernel is very flexible kernel. It is stationary (translation invariant), and adapts very well to the dimensionality of the problem so it had no problems adapting to these toy datasets.



(a) $r = 0$



(b) $r = 1$



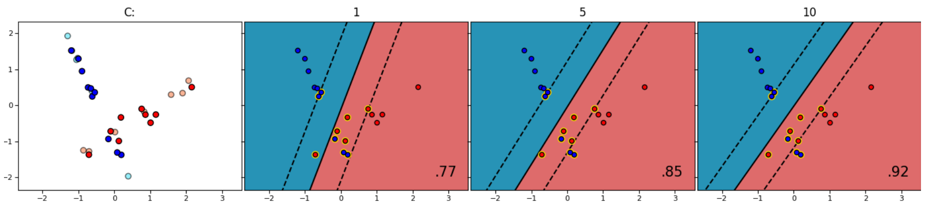
(c) $r = 0$, datasets extended with an extra feature of all ones

Figure 5: A visualisation of decision boundaries on toy datasets using polynomial kernel SVMs of odd end even degree, with different settings of the r parameter

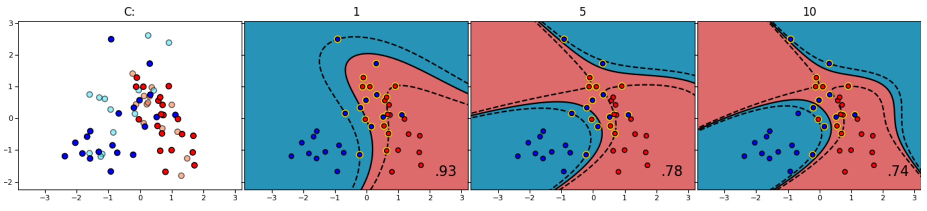
Figure 6 aims at demonstrating the effect the change of various parameters causes in the resulting model. Firstly the C parameter balances the two terms of the objective function of the SVM learning algorithm - higher values penalize points violating the margin more, causing the selection of model that better fits the training data, but with a thinner margin. Thus low values lead to under fitting and high values to overfitting.

The degree of a polynomial kernel impacts the maximum complexity of the decision surface(5b). While higher degree allows for a more complex hypothesis it is more prone to overfitting, while using lower degree than necessary for the problem, will severely underfit.

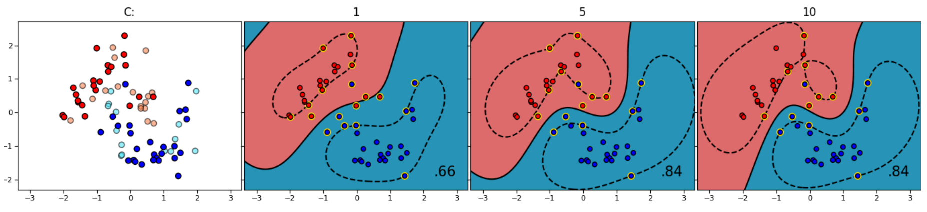
The RBF kernel's gamma parameter defines how much influence a single training example has. The larger gamma is, the closer other examples must be



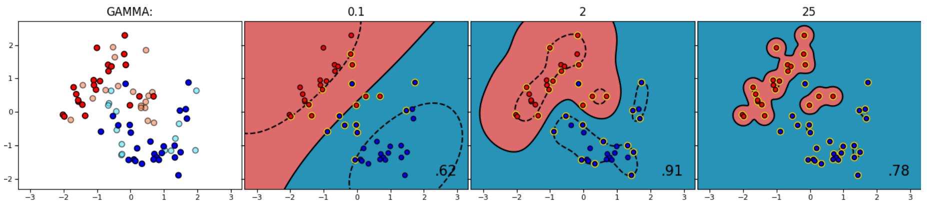
(a) Changing C, linear kernel SVM



(b) Changing C,polynomial kernel SVM



(c) Changing C, RBF kernel SVM



(d) Changing gamma, RBF kernel SVM

Figure 6: Effects of some parameters on SVMs. Dotted lines mark the margin - here the decision function equals 1 and -1 respectively. The size of the margin that the SVM strives to maximize, is hard to gauge graphically in any but the linear case, because it lives in the space generated by the kernel, not the feature space.

to be affected. Thus gamma is in a way related to the scale of the problem. A value too small causes a point to have a influence over a large portion of the dataset, resulting in underfitting and high values cause that only the immediate surroundings of the point are affected resulting in severe overfitting.

3.3 Hyper-parameter tuning results for SVMs

It is necessary to avoid using the same data to estimate the parameters and to evaluate the performance of the selected best combination, or we would commit a peeking error, which skews the results towards higher numbers, because we would directly optimize the performance on the test set, which should be inaccessible at the time of learning. So I chosen a nested crossvalidation approach in which the dataset is first split to test and training sets in the outer 10-fold crossvalidation, and then the parameter grid search on the training set is used to select the best parameters for the SVM (and its kernel), equaling a 5-fold crossvalidation inner loop.

		Mean score						Mean fitting time						Mean scoring time					
		TF			TF IDF			TF			TF IDF			TF			TF IDF		
	C		norm	scale		norm	scale		norm	scale		norm	scale		norm	scale		norm	scale
rbf	1.00E-01	72.37	60.60	60.60	76.23	60.60	60.60	1.09	1.07	1.07	1.07	1.04	1.07	0.19	0.20	0.20	0.20	0.19	0.20
	1.00E+00	88.84	60.60	60.60	90.24	60.60	60.60	0.68	1.00	1.04	0.55	1.00	1.01	0.12	0.20	0.19	0.11	0.20	0.20
	1.00E+01	92.32	88.91	70.41	92.34	91.59	70.41	0.43	0.73	1.02	0.39	0.65	1.00	0.07	0.13	0.20	0.07	0.13	0.20
	1.00E+02	93.36	92.22	86.84	92.95	92.75	86.84	0.35	0.42	0.66	0.33	0.39	0.67	0.06	0.07	0.13	0.05	0.06	0.13
	1.00E+03	93.70	93.48	90.72	93.60	93.07	90.72	0.53	0.32	0.47	0.52	0.28	0.47	0.04	0.05	0.08	0.04	0.04	0.08
	1.00E+04	93.53	93.53	92.75	93.45	93.33	92.75	1.40	0.40	0.44	1.63	0.45	0.44	0.04	0.04	0.05	0.03	0.04	0.06
	1.00E+05	92.73	93.79	92.97	92.37	93.84	92.97	9.16	1.02	0.76	7.27	1.39	0.77	0.04	0.04	0.05	0.03	0.04	0.05
	1.00E+06	90.75	94.37	93.07	89.49	94.11	93.07	12.94	6.14	1.89	10.74	8.04	1.90	0.04	0.03	0.04	0.03	0.03	0.03
	1.00E+07	87.49	93.38	93.16	87.34	92.46	93.16	13.11	6.89	2.40	10.85	8.52	2.41	0.03	0.03	0.03	0.03	0.03	0.03
	1.00E+08	85.02	91.81	92.95	83.67	91.74	92.95	10.31	5.85	2.50	8.87	7.14	2.49	0.03	0.02	0.03	0.02	0.03	0.03
	1.00E+09	82.34	91.26	92.13	82.58	90.41	92.13	9.78	5.68	2.47	8.17	6.68	2.47	0.02	0.03	0.04	0.02	0.02	0.02
linear	1.00E-02	91.16	86.69	60.75	91.69	90.80	60.75	0.47	0.87	1.01	0.44	0.78	1.02	0.07	0.12	0.15	0.05	0.11	0.14
	1.00E-01	92.27	91.88	84.93	92.75	92.32	84.93	0.38	0.46	0.77	0.36	0.40	0.77	0.04	0.07	0.10	0.05	0.05	0.10
	1.00E+00	92.92	93.19	89.93	92.71	92.83	89.93	0.50	0.30	0.48	0.61	0.28	0.47	0.04	0.04	0.07	0.03	0.03	0.07
	1.00E+01	92.78	93.48	92.08	92.73	93.29	92.08	1.94	0.30	0.37	3.13	0.33	0.38	0.03	0.03	0.04	0.04	0.03	0.04
	1.00E+02	91.01	93.41	92.71	92.37	93.31	92.71	8.01	0.71	0.55	9.65	0.88	0.54	0.03	0.03	0.03	0.04	0.02	0.03
	1.00E+03	88.84	93.41	92.80	79.44	93.14	92.80	34.43	3.46	2.12	35.15	4.78	1.99	0.05	0.03	0.03	0.06	0.03	0.03
	1.00E+04	66.88	93.00	92.75	62.10	93.16	92.73	29.47	12.84	6.39	29.31	17.51	6.42	0.04	0.02	0.02	0.03	0.03	0.03
poly	1.00E-02	63.19	86.67	72.90	61.35	89.32	72.90	0.98	0.82	0.94	1.03	0.87	0.96	0.15	0.12	0.14	0.16	0.12	0.14
	1.00E-01	82.42	92.27	82.87	62.90	92.80	82.87	0.78	0.45	0.65	0.98	0.44	0.64	0.11	0.07	0.10	0.15	0.07	0.09
	1.00E+00	87.63	93.99	87.92	80.02	93.94	87.92	0.50	0.28	0.45	0.82	0.30	0.44	0.07	0.05	0.07	0.12	0.04	0.06
	1.00E+01	91.26	93.77	91.67	87.08	93.96	91.67	0.40	0.29	0.44	0.51	0.33	0.44	0.05	0.03	0.04	0.08	0.03	0.05
	1.00E+02	92.49	92.87	92.17	91.38	92.25	92.17	0.54	0.54	0.87	0.39	0.55	0.87	0.04	0.03	0.04	0.05	0.03	0.04
	1.00E+03	92.32	91.74	91.06	92.27	91.38	91.04	3.17	1.75	4.82	0.48	1.40	4.83	0.03	0.03	0.03	0.04	0.02	0.03
	1.00E+04	91.21	90.77	90.24	92.00	91.45	90.24	6.17	5.75	7.17	1.07	4.94	7.14	0.02	0.02	0.03	0.03	0.02	0.02

Table 3: Heatmaps of mean accuracy, fitting time and scoring time of one instance of the gridsearch (inner 5-fold crossvalidation). The color map is scaled each kernel's results individually, The polynomial kernel is of degree 2 and gamma for RBF kernel was choosen as 0.001. Norm means that l2 normalization has been applied to samples, while scale means that scaling by maximum value has been used

Table 3 shows that SVM exhibit underfitting when C is too low, regressing to always predicting the largest class prior, and overfitting where the performance on the training set is maximized at the expanse of generalization error. Normalizing or scaling of course rescales the problem, and the shift in optimal ranges for C is in line with SVMs not being an scale invariant technique.

Furthermore we can see a relationship between fitting times and the prediction performance of the SVMs. In the underfitting scenario the convergence of the underlying optimization algorithm is quite fast but gets even faster with increasing C up to point than begins increasing quite rapidly, but just as is bagins to rise the prediction accuracy seems to hit it's maximum. The growth seems to stop at some point and the times to go down again as we enter the severe overfitting region, but these times at the high end of the spectrum are somewhat unreliable because there are cases in which the optimizer hit the cap of maximum iterations which I've set to 1000000 iterations, which means that just these iterations were completed faster. I've used this cap because otherwise the computation would be too long, and there seem to be be little difference to the results - after so many iterations the steps done by the optimizer should be small even if not in the tolerance yet. Remembering that C is the upper bound on the Laplace multipliers I would says that as the problem gets essentially less constrained with rising C the search space gets larger and that is why we see an increase in training times. Scaling or normalizing will get all the future values in a range that is typically much smaller range, maybe this scales the search space of the optimization problem as well.

The scoring times are included in the table as an indicator of the number of support vectors, which I was unable to get directly due to technical limitations of the `GridsearchCV` class which doesn't store all estimator variants tested only the best one. The dropping scoring time suggest fewer support vectors, which is line with the expectation that the margin width decreases with growing C so that fewer point violate it and also that the number of missclassified points is minimized (which are also support vectors).

		gamma									
		scale	1.00E-08	1.00E-07	1.00E-06	1.00E-05	1.00E-04	1.00E-03	1.00E-02	1.00E-01	1.00E+00
	1.00E-01	90.29	60.60	60.60	60.60	60.60	60.60	72.37	88.62	88.84	64.06
	1.00E+00	93.29	60.60	60.60	60.60	60.60	73.02	88.84	92.75	93.60	84.83
	1.00E+01	93.67	60.60	60.60	60.60	73.04	88.45	92.32	93.67	93.50	85.43
	1.00E+02	93.36	60.60	60.60	73.00	88.33	91.86	93.36	93.74	92.56	85.14
	1.00E+03	92.29	60.60	72.97	88.24	91.79	93.19	93.70	92.92	91.69	85.12
C	1.00E+04	90.77	73.26	88.31	91.74	92.83	93.45	93.53	91.93	91.06	85.02
	1.00E+05	90.70	88.29	91.40	92.56	93.21	93.72	92.73	90.87	90.68	84.78
	1.00E+06	90.75	90.29	91.57	91.69	93.12	93.14	90.75	90.12	90.12	83.74
	1.00E+07	86.23	88.16	90.48	91.50	92.90	91.93	87.49	88.31	88.55	82.37
	1.00E+08	77.05	88.02	91.45	91.26	93.14	91.16	85.02	83.91	82.29	81.50
	1.00E+09	75.31	88.67	90.12	91.16	93.12	90.87	82.34	82.83	81.01	81.50

Table 4: Heatmap of C and gamma parameter combinations for RBF kernel SVM on the TF representation of the SPAMBASE dataset. Mean accuracies from one of the inner gridsearches. The scale column represents gamma picked by heuristic $1/(n_features * X.var())$ *where the variance is computed over the flattened array*

Table 4 demonstrates that a good balance of gamma and C is necessary as the best values are located on the diagonal of the table. We can also see that only a range of gammas are viable - small gammas cause the influence radius to be very big , and so is the margin if the decision boundary is not 'pushed' from the other side by a support vector of the other class, so the SVM needs higher regularization via C so that it is forced to classify some points correctly instead of just keeping the margin by predicting just one class. On the other side of the spectrum big values of gamma essentially break up the problem to small areas around the support vectors. Because of the exponential the influence of a point never quite reaches 0 at this long distance of 0 exponential resembles in a way a constant function. And because all distances in the problem are very large with regard to high gammas, many distant points can overpower a few close one because all of them have a comparably small influence.

C	coef0	
	0	1
1.00E-02	63.19	85.00
1.00E-01	82.42	90.70
1.00E+00	87.63	93.38
1.00E+01	91.26	93.70
1.00E+02	92.49	93.26
1.00E+03	92.32	92.56
1.00E+04	91.21	91.14

Table 5: Table of second degree polynomial kernel SVM with coef0 set to 0 (default) and 1, mean accuracies from one of the gridsearch runs. Scores over the TF representation

Table 5 shows that setting coef0 to 1 for the polynomial kernel seems a like good thing to do - it seems that it makes the kernel work well with broader range of Cs and even yields a slightly higher score. The training times were only slightly higher in some cases for $coef0 = 1$.

3.4 Test results

Classifer	**TF**			**TF-IDF**		
	----	**normalized**	**scaled**	**----**	**normalized**	**scaled**
svm-rbf	93.59 ± 0.43	94.33 ± 0.32	93.35 ± 0.28	93.76 ± 0.43	94.28 ± 0.27	93.35 ± 0.28
svm-linear	92.55 ± 0.37	93.20 ± 0.46	92.50 ± 0.33	92.33 ± 0.36	92.94 ± 0.48	92.13 ± 0.47
svm-poly-d2-c00	92.41 ± 0.52	93.65 ± 0.39	92.46 ± 0.32	92.76 ± 0.37	93.57 ± 0.29	92.46 ± 0.32
svm-poly-d2-c01	93.74 ± 0.33	94.02 ± 0.33	93.33 ± 0.32	93.65 ± 0.45	93.96 ± 0.41	93.33 ± 0.32
svm-poly-d3-c00	91.35 ± 0.36	93.96 ± 0.36	91.20 ± 0.35	90.85 ± 0.42	93.89 ± 0.32	91.20 ± 0.35
svm-poly-d3-c01	93.55 ± 0.33	94.59 ± 0.36	93.20 ± 0.30	93.46 ± 0.46	94.26 ± 0.41	93.20 ± 0.30
gnb-sk	81.24 ± 0.41	82.68 ± 0.48	81.18 ± 0.41	81.24 ± 0.41	84.29 ± 0.57	81.18 ± 0.41
gnb-my	81.18 ± 0.42	82.63 ± 0.49	81.16 ± 0.42	81.16 ± 0.42	84.20 ± 0.60	81.16 ± 0.42
multinomialnb	87.15 ± 0.53	90.52 ± 0.51	88.70 ± 0.20	88.11 ± 0.54	89.68 ± 0.52	88.70 ± 0.20

Table 6: Table of mean accuracy in 10-fold crossvalidation, SVM hyperparameters trained using nested 5-fold crossvalidation over a grid of parameters. The standard deviation of the mean is given next to the mean accuracy. d is the degree of polynomial kernel and c0 is the coef0/r parameter. Normalized means l2 normalization has been applied to feature vectors, and scaled means that the values have been scaled by the maximum of respective feature in the training data. Gnb-sk is the Gaussian naive Bayes implementation from Scikit-learn while the other is mine.

From the results we can see that the data is linearly separable as the linear kernel achieved almost the same performance as polynomial or RBF kernels. Setting the coef0 to 1 boosts the performance of polynomial kernels a little. Degree three kernel, with coef0 set to 0 performed slightly lower without normalization, the gridsearch always seemed to choose $C = 10000$ as the best value which was the maximum of the searched range in this test, so probably even higher values were necessary to bring these cases in line with results other SVM based classifiers achieved.

The TF-IDF[4]representation offered no gains over TF for the SVMs. I believe that the our use-case doesn't match the intended one - to weight the words by their importance to dataset, because all features in the dataset were selected for being highly discriminative. I've looked at the scaling factors in the debugger and they are in the 0.3 to 4.5 range, not a very broad range, which in my opinion supports the theory that all the features were pretty close in their significance. Also IDF scaling doesn't affect Gaussian naive Bayes method because it doesn't change feature's distribution's shape. Multinomial Naive Bayes however is affected because it models the feature vector as a whole not every feature by it self as Gaussian, so the weighting of the features is significant.

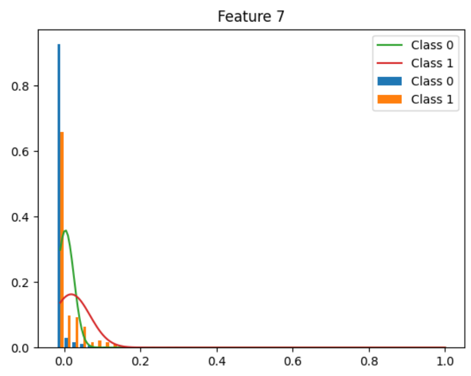
But when we apply the l2 normalization to the the representations every classifier achieves higher accuracy, with the Naive Bayes methods benefiting the most. When thinking about what the normalization does to the dataset from the sort of semantic view of the dataset, I would say that while the TF representation answered the question of frequent a term in a document is the normalized variant is more akin to which of this set of terms occurred most frequently. We removed the information about the length of the text, which was hidden in the denominators of term frequencies (the longer the text the shorter the vector statistically was), but introduced a new relation to the rest of the features. I think this works because we are more interested in the patterns appearing in the documents and want to be able to recognize them and compare two documents without their length playing major role.

Scaling the data had almost no effect on accuracy as expected, but can improve training times of SVMs. Of course scaling undoes any scaling done by the TF-IDF transformation (that I even computed this column was unnecessary and I could have made the testing shorter by 1/6, oh well...).

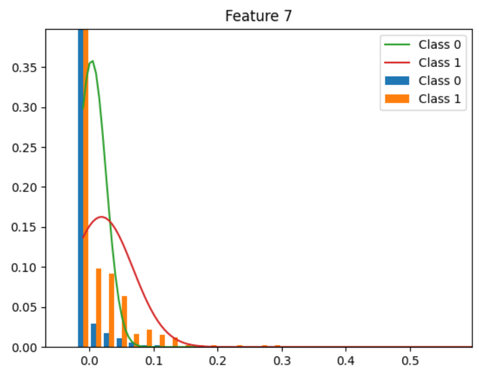
[4]There is a small problem with how I calculated the TF-IDF representation - I have transformed the whole dataset, thus I believe I have committed a peeking error, because the occurrences in the test sets should be locked away from the learning process. But since I realized this too late, I did not redo the tests, but I implemented a small experiment on the side (and while doing so I improved the speed of my TF to TF-IDF transformation considerably) and the results suggest that the results would not change much, probably because the testing datasets, even in the inner crossvalidation, contain more than 70% of the dataset, which might be already good estimate for the document frequencies. Anyway I think we can treat the numbers as best-case/upper bound. Moreover if the problem was like this: we have a partially labeled set of documents and we have to label the rest, than, in such a case, we could indeed compute the TF-IDF over the whole set (this resembles a bit the original usecase of TF-IDF - document retrieval)

3.4.1 Results discussion

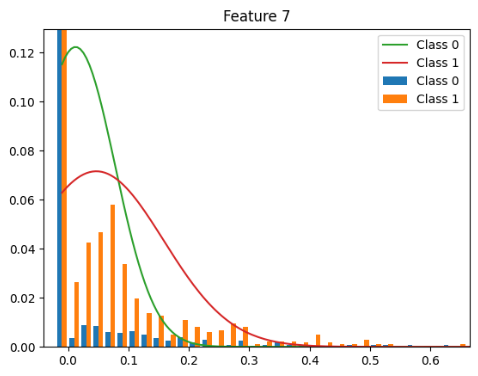
Why does Gaussian Naive Bayes perform so poorly? I believe the problem comes down to to that the features do not follow normal distribution - there is a huge proportion of zeros for each feature in both classes. Other values are represented only very rarely and not necessarily near 0 or near each other.



(a) Feature 7 (indexed from 0) histogram, scaled by maximum value. Lines are the learned normal distributions.



(b) Feature 7 histograms magnified.



(c) Normalized feature 7 histograms magnified

Figure 7: Histograms depicting the difference between the Gaussian model and relative frequencies in the dataset

We can see (and we can see this in other features), that the Gaussian is not a good fit:

- The probabilities of feature being 0 should be the maximums but they are not.
- The probabilities close to 0 are flipped.

- The Gaussian has too low of a tail.

Now I might be able to explain the better performance of multinomial naive Bayes - it is actually much better at modeling the problem at and near 0. When we take a look at the multinomial distribution in one dimension we should see what is essentially binomial distribution (which for large number of trials approximates Gaussian distribution) but with a twist of that when we put feature $x_i = 0$ the actual contribution of this feature to the value of multinomial distribution probability mass function jumps to 1 (both multiplicative terms than depend on it, $x_i!$ and $p_i^{x_i}$, are 1) so the end value is determined by the other features.

I believe that if we modified Gaussian naive Bayes to model the the zero and nonzero feature values separately, as Bernoulli and Gaussian distributions respectively, the performance might be similar (maybe even better) as multinomial naive Bayes, but I have not tried.

Another thing I tried is binarizing the data. Thus all non zero data is grouped and has the required weight to move the mean away from the beginning. Of course when working with binary features Bernoulli naive classifier works better.

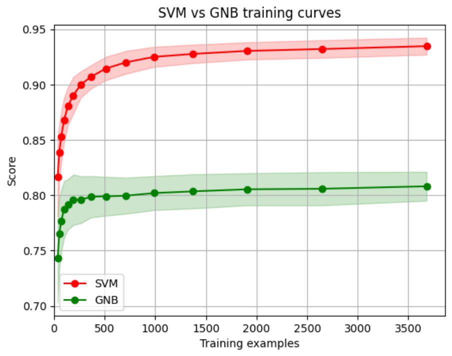
	GaussianNB	MultinominaNB	BernoulliNB
	81.24%	87.15%	
normalized	82.68%	90.52%	
binarized	86.00%	89.24%	88.57%

Table 7: Table of mean accuracy in 10-fold crossvalidation

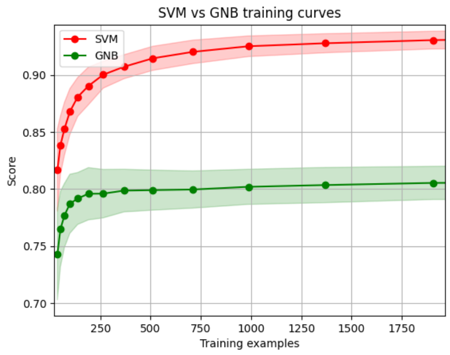
Another thing that has caught me by surprise is that normalization affected the histograms in the opposite way I predicted. Instead of concentrating all the samples around one or two spikes, the distributions actually got wider with more smaller spikes.

The figure **??** demonstrates the performance of SVM and Naive Bayses using training datasets of various sizes. While multinomial naive Bayes can compete at low amount of training data, it's learning rate rapidly slows done at just around 250 samples and seems to almost stop at 700 samples. Meanwhile the SVMs might not be fully saturated even at the last test point, but the learning rate is slow very slow at this point. SVMs also showcase more stable performance, with lower variance.

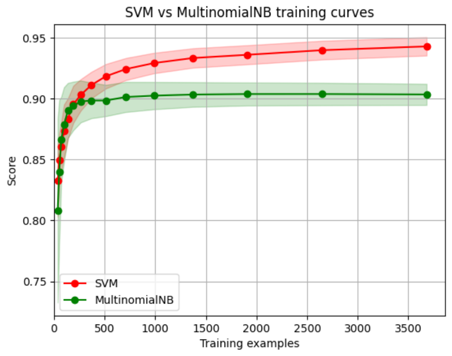
The last interesting bit is that the Gaussian when inspecting the accuracies per class, SVM delivers solid performance on both classes, Naive Bayes were very bad at detecting legitimate email and often missclassified it as spam - false positives is the more severe type of mistake in this case. At first I thought it is caused by the class priors but it is not the case, overwriting the priors did not change the results much (in fact spam is only roughy 40%, opposite of what I expected). My second guess is that that there are more features that are good indicators of an email being spam, but no so many for the other class.



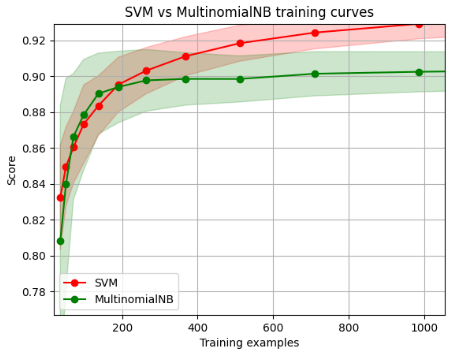
(a) *Learning curves on the TF representation, best SVM in the test vs GNB.*



(b) *Previous figure magnified.*



(c) *Normalized TF representation, best SVM in the test vs multinomaialNB*



(d) *Previous figure magnified.*

Figure 8: Learning curves, the line represents mean accuracy over random 100 splits, with 0.2 of the dataset reserved as the test each time. The shading represents a distance of one standard deviation.

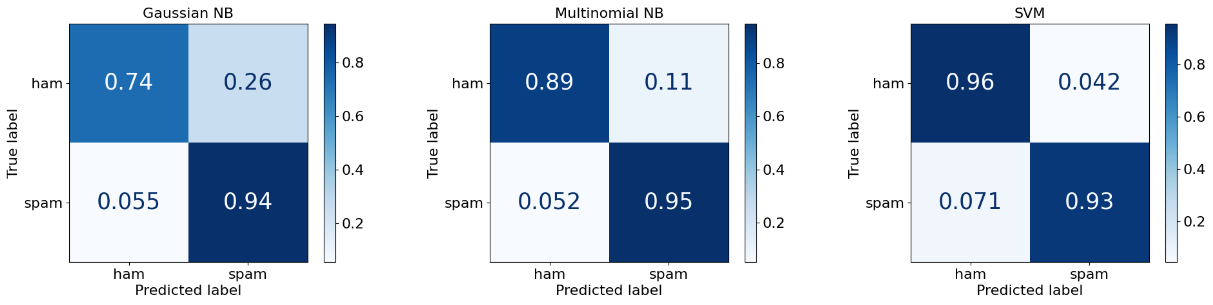


Figure 9: Confusion matrices with 0.3 of the dataset as test set, normalized TF representation. Best performing SVM from the test.

3.5 Conclusion

In this assignment I've studied Support vector machines and Naive Bayes classifies. I applied them to the SPAMBASE dataset and discussed the results. While this project is nothing groundbreaking it was a wonderful first step into machine learning.

References

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2010.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[3] A. Kowalczyk, *Support Vector Machines Succintly.* Syncfusion, 2017.

[4] C. J. Burges, *A Tutorial on Support Vector Machines for Pattern.* Kluwer Academic Publishers, 1998.

[5] H. Zhang, "The Optimality of Naive Bayes,"

[6] J. Teevan, L. Shih, J. D. M. Rennie, and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," 2003.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[8] J. Ah-Pine, "Normalized Kernels as Similarity Indices,"

[9] G. Paul. (2003). Better bayesian filtering, [Online]. Available: `http://www.paulgraham.com/better.html`.

[10] Y. Abu-Mostafa. (2012). Caltech cs 156 [video recordings], [Online]. Available: `https://youtu.be/eHsErlPJWUU`.

Images that were not created by me, mainly in the first half, are from Wikipedia and fall under creative commons license.

And of course the course materials by prof. Andrea Torsello and prof Marcello Pelillo at Ca'Foscari, plus ton of online sources that didn't manage to keep track of - thank you to all.

[9] is old, but a fun read about spam filtering.