Chapter 10

Support Vector Machines, part 2

Support Vector Machine: soft margins and the kernel trick. Regularization of SVM

10.1 Soft Margins

In the previous chapter, we derived this *dual problem* from the problem of minimizing the norm of the hyperplane vector subject to the constraint that the margin would be at least 1:

$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \vec{x}_n^T \vec{x}_m$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0 \qquad \alpha_n \geq 0$$

However, this is only possible if the data sets are linearly separable. Otherwise, the constraints are incompatible and the problem has no solution, as illustrated on Figure 10.1. An intuitive way of understanding this is to note that, if vectors $\vec{x}$ are surrounded by neighbours of another class, then the corresponding α values can rise to infinity to maximize the target function. This means the function no longer has a maximum value.

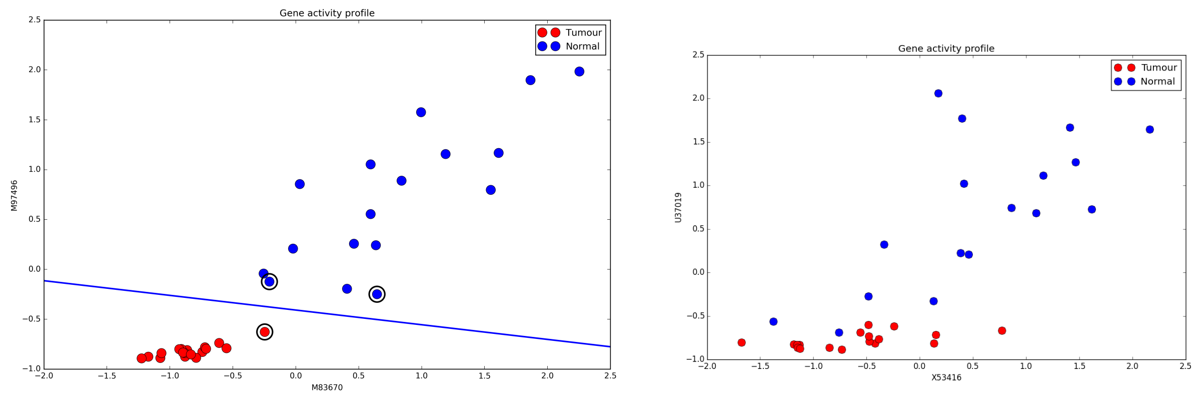

Figure 10.1: The left panel shows the decision frontier of a linear SVM. The right panel shows an example of data that is not linearly separable.

To solve this problem, we can add a slack variable ξ_n for each vector, a positive value representing the distance between the vector and the inside of the margin, or zero if the vector is not inside the margin:

$$y_n(\vec{w}^T x_n + b) \geq 1 - \xi_n, \forall n \in N \qquad \xi \geq 0$$

If $1 > \xi_n > 0$, then the vector $\vec{x}$ is inside the margin; if $\xi_n > 1$, then the vector $\vec{x}$ is on the wrong side of the decision hyperplane. This allows vectors to penetrate the margins. However, we want to minimize the violation of the margin constraint, so now we want to minimize $||\vec{w}||^2$ but penalizing violations to the margin:

$$\arg\min C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\vec{w}||^2$$

For the new lagrangian, we need additional lagrangian multipliers for the ξ variables, given the constraint that they must be at zero or greater:

$$\mathcal{L}(\vec{w}, b, \vec{\alpha}, \vec{\mu}, \vec{\xi}) = \frac{1}{2}||\vec{w}||^2 + C\sum_{n=1}^{N} \xi_n - C\sum_{n=1}^{N} \alpha_n \left(y_n(\vec{w}^T\vec{x} + b) - 1 + \xi_n\right) - \sum_{n=1}^{N} \mu_n \xi_n$$

Setting the derivatives to 0 we obtain the same dual problem, but the derivative of the lagrangian as a function of the ξ slack variables forces the α parameters to be less than C:

$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \vec{x}_n^T \vec{x}_m$$

$$\sum_{n=1}^{N} \alpha_n y_n = 0$$

$$\frac{\delta\mathcal{L}}{\delta\xi_n} = 0 \Leftrightarrow C - \alpha_n - \mu_n = 0 \Leftrightarrow 0 \leq \alpha_n \leq C, \qquad \sum_{n=1}^{N} \alpha_n y_n = 0$$

So, to fit the SVM with soft margins, we just need to add the C parameter to the upper bounds of the α values:

```
H = H_matrix(Xs,Ys)
A = Ys[:,0] # sum of alphas is zero
cons = {'type':'eq',
        'fun':lambda alphas: np.dot(A,alphas),
        'jac':lambda alphas: A}
bounds = [(0,C)]*Xs.shape[0] #alpha>=0
x0 = np.random.rand(Xs.shape[0])
sol =  minimize(loss, x0, jac=jac, constraints=cons, method='SLSQP', bounds = bounds)
```

With this change, it is now possible to compute a SVM classifier for a data set in which the classes have a slight overlap. Figure 10.2 shows the result. The thin green lines represent the margins. All support vectors – the vectors for which the α values are greater than zero – are marked with a circle. Those with red circles are inside the margins, corresponding to ξ values greater than zero and α values maximized to C

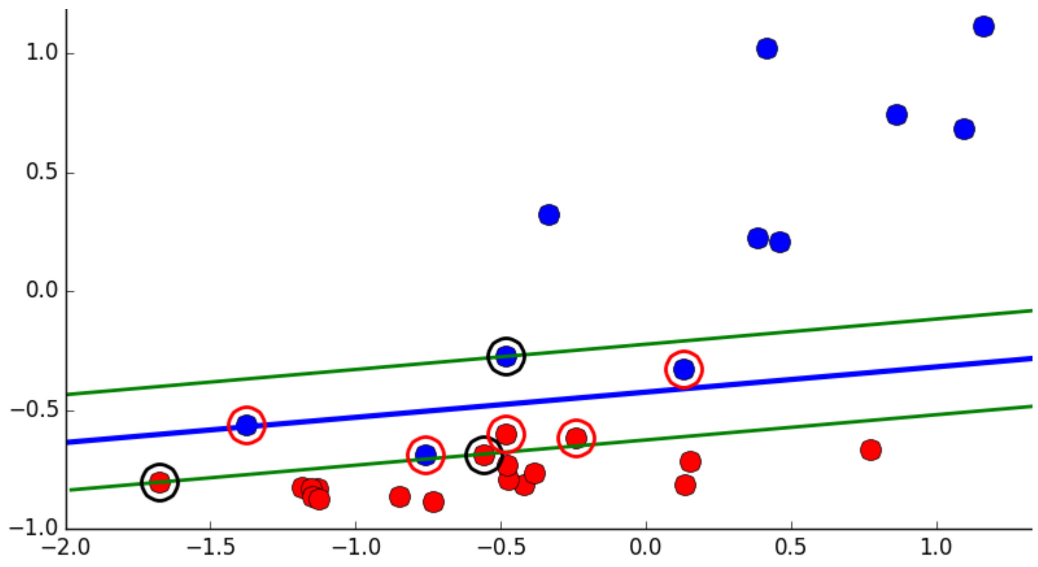

Figure 10.2: Soft-margin separation with SVM. Vectors indicated with circles are support vectors. Those inside red circles are support vectors inside the margins, for which $\alpha = C$.

Note that, in this case, to compute the w_0 parameter we can only use the support vectors that do not penetrate the margins (*i.e.* those for which $alpha < C$). For those that lie inside the margins ($\alpha = C$) the equation $y_n(\vec{w}^T\vec{x}_n + w_0) = 1$ is not valid. So, for computing w_0 we average $y_n - \vec{w}^T\vec{x}_n$ using only those support vectors for which $0 < \alpha_n < 0$.

10.2 Non-linear separation and the Kernel Trick

Soft margins can solve slight overlaps, but sets that are not linearly separable we'll generally need a different approach.

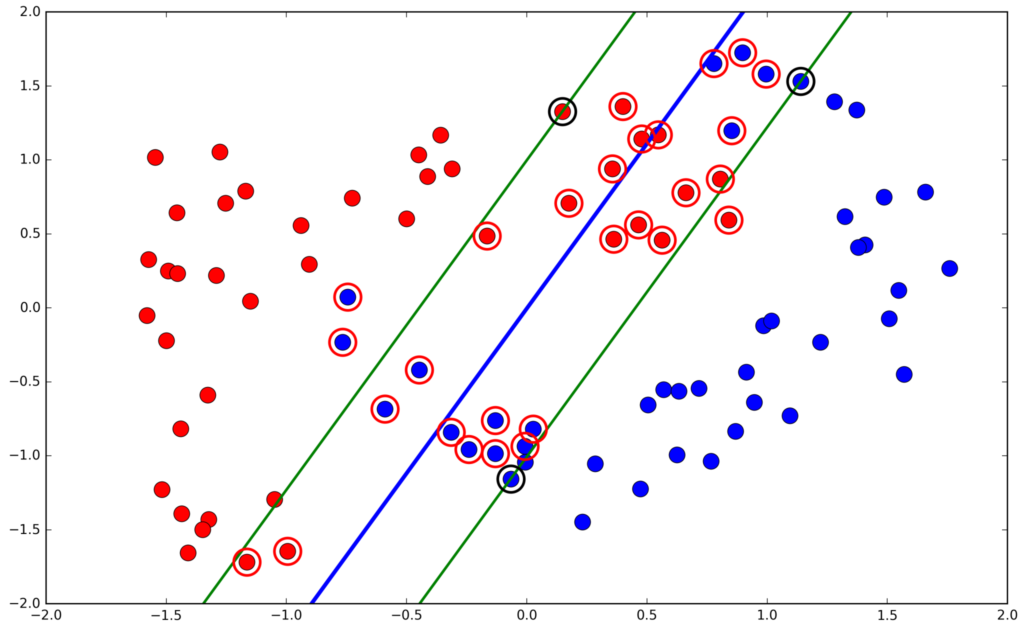

Figure 10.3: For data that is not linearly separable, soft-margins alone are generally not a good solution.

As we saw several times before, the way to classify sets that are not linearly separable with linear classifiers is to use a representation of the data in higher dimensions. With SVM, this is easy to do with *kernel functions*. First, we note that all the inner products of the pairs of training vectors can be precomputed, as we did in the last chapter:

$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n\alpha_m y_n y_m \boxed{\vec{x}_n^T\vec{x}_m}$$

If we expand the training vectors to higher dimensional representations, we could compute the inner products of these expanded vectors. However, there is an even better way to do this. Since all we need are the inner products and not the vectors themselves, we can use *kernel functions*. A *kernel function* is a function that gives us the inner product of the transformed vectors as a function of the original vectors:

$$K(\vec{x}_1, \vec{x}_2) = \phi(\vec{x}_1)^T\phi(\vec{x}_2)$$

For example, this function ϕ transforms a two-dimensional vector into a six-dimensional vector:

$$\phi(\vec{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2]^T$$

But, in this case, $\phi(\vec{x_1})^T\phi(\vec{x_1}) = (\vec{x_1}^T\vec{x_2}+1)^2$, and so we have a *kernel function* in this case, which is $(\vec{x_1}^T\vec{x_1}+1)^2$. More generally, for degree n polynomial expansions of this sort, the *kernel function* is:

$$K_{\phi^n}(\vec{x_1}, \vec{x_2}) = (\vec{x_1}^T\vec{x_2}+c)^n$$

So we can solve the original problem, but using a *kernel function* that implicitly expands the data to a higher-dimensional space:

$$\arg\max_{\vec{\alpha}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} \alpha_n\alpha_m y_n y_m K(\vec{x_n}, \vec{x_m})$$

To classify a new vector $\vec{x_t}$, once the SVM is trained, we cannot compute $\vec{w}$ because$\vec{w}$ will define a hyperplane on the higher-dimensional space where the *kernel* implicitly projects the original data. So, instead, we compute the class of $\vec{x_t}$ using the support vectors:

$$\vec{w}^T\phi(\vec{x_t}) + w_0 = \sum_{n=1}^{N} \alpha_n y_n K(\vec{x_n}, \vec{x_t})$$

For example, we can use a polynomial kernel of degree d, $K_{\phi^d}(\vec{x_1}, \vec{x_2}) = (\vec{x_1}^T\vec{x_2}+1)^d$:

```python
def H_poly(X,Y,n):
    H = np.zeros((X.shape[0],X.shape[0]))
    for row in range(X.shape[0]):
        for col in range(X.shape[0]):
            k = (np.dot(X[row,:],X[col,:])+1)**n
            H[row,col] = k*Y[row]*Y[col]
    return H
```

And to classify a new point we use the support vectors:

$$\vec{w}^T\phi(\vec{x_t}) + w_0 = \sum_{n=1}^{N} \alpha_n y_n K_{\phi^d}(\vec{x_1}, \vec{x_2})$$

```python
def poly_k_class(X,alphas,Y,xt,d):
    s = 0
    for ix in range(len(alphas)):
        s = s + (np.dot(X[ix,:],xt)+1)**d*Y[ix]*alphas[ix]
    return s
```

However, in practice we will not implement this ourselves. This code is just to illustrate the computation. It is best to use an optimized implementation of a SVM, such as the one provided in the `sklearn` library. To use the SVM classifier from `sklearn`, we just need to specify the kernel type and corresponding parameters to the constructor of the `SVC` class (support vector classifier):

```python
from sklearn import svm
#load and standardize
sv = svm.SVC(C=C,kernel = 'poly', degree = poly, coef0 = 1)
sv.fit(Xs,Ys[:,0])
```

Figure 11.7 shows the result of fitting the SVM with a third degree polynomial kernel. Using a lower value of C places a lower upper limit on the penalty for margin violations. This leads to several support vectors being placed inside the margin (thus maximizing the α values to C). A higher value of C results in greater penalties for margin violations and, in this case, with $C = 1000$ no support vectors are placed inside the margins.

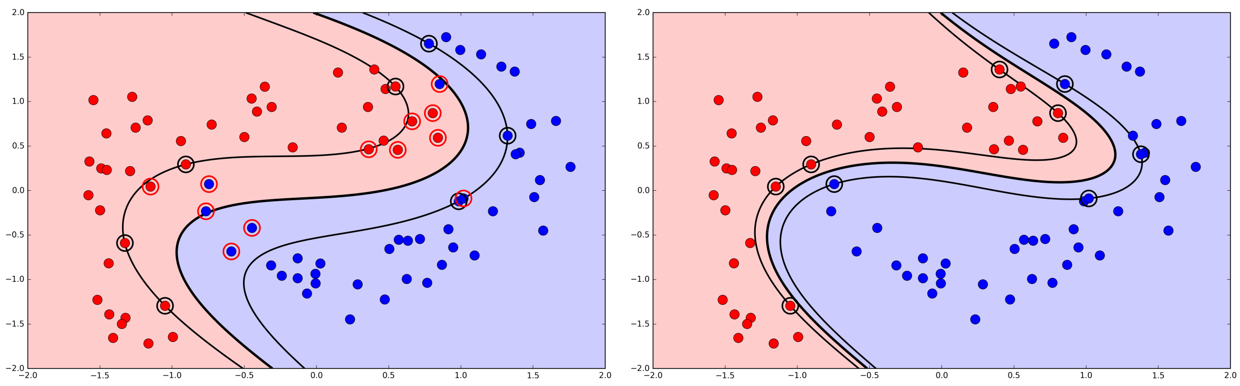

Figure 10.4: SVM trained with a third degree polynomial kernel. The SVM on the left panel was trained with $C = 1$, and the right with $C = 1000$.

So we can see C as a regularization parameter, with lower C values corresponding to higher regularization, simplifying the decision surface at the cost of allowing more errors. Figure 10.5 shows another example, this time using a Gaussian kernel, also known as Gaussian Radial Basis Function kernel, or RBF:

$$K(\vec{x_1}, \vec{x_2}) = e^{\frac{-||\vec{x_1}-\vec{x_2}||^2}{2\sigma^2}}$$

In Scikit Learn, $1/2\sigma^2$ is combined into the γ parameter in $K(\vec{x_1}, \vec{x_2}) = e^{-\gamma||\vec{x_1}-\vec{x_2}||^2}$.

```python
from sklearn import svm
#load and standardize
sv = svm.SVC(C=C,kernel = 'rbf', gamma=gamma)
sv.fit(Xs,Ys[:,0])
```

Figure 10.5 shows different combinations of C and γ values. The top panels show the results of using $\gamma = 0.01$ and $\gamma = 2$ with $C = 1$. A higher γ value makes the RBF kernel weigh nearby points more strongly, making the decision frontier conform more tightly to the two classes; a lower γ value broadens the radius of the RBF kernel smoothing the frontier. The bottom panels, with $C = 1000$ show the same effect but with less regularization.

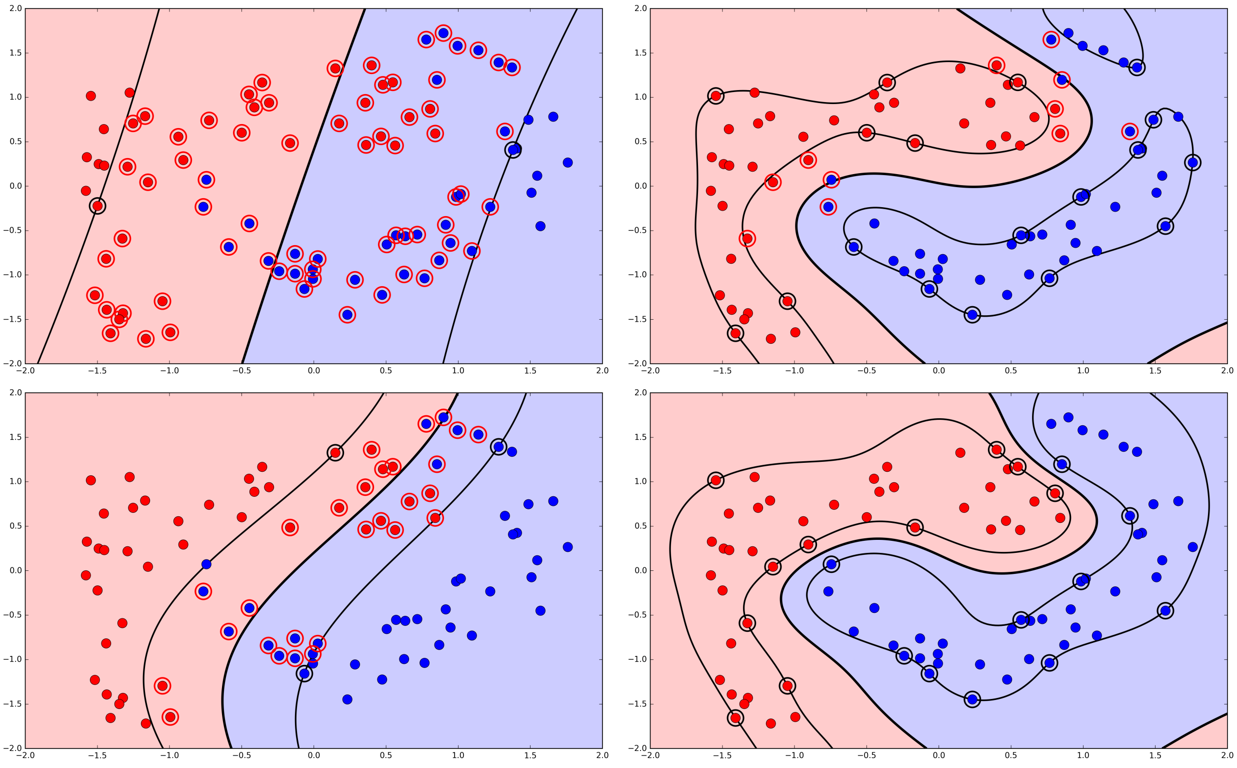

Figure 10.5: SVM trained with a RBF kernel. In the top panels, the SVM was trained with $C = 1$, and $C = 1000$ in the bottom panels. The panels on the left show SVM trained with $\gamma = 0.01$, those on the right with $\gamma = 2$.

10.3 Further Reading

1. Alpaydin [2], Sections 13.1 - 13.8
2. Marsland [17], Chapter 5
3. Bishop [4], Section 7.1

Bibliography

[1] Uri Alon, Naama Barkai, Daniel A Notterman, Kurt Gish, Suzanne Ybarra, Daniel Mack, and Arnold J Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. *Proceedings of the National Academy of Sciences*, 96(12):6745–6750, 1999.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[3] David F Andrews. Plots of high-dimensional data. *Biometrics*, pages 125–136, 1972.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, New York, 1st ed. edition, oct 2006.

[5] Deng Cai, Xiaofei He, Zhiwei Li, Wei-Ying Ma, and Ji-Rong Wen. Hierarchical clustering of www image search results using visual. Association for Computing Machinery, Inc., October 2004.

[6] Guanghua Chi, Yu Liu, and Haishandbscan Wu. Ghost cities analysis based on positioning data in china. *arXiv preprint arXiv:1510.08505*, 2015.

[7] Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990.

[8] Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning. Stanford CA Morgan Kaufmann*, pages 231–238, 2000.

[9] Hakan Erdogan, Ruhi Sarikaya, Stanley F Chen, Yuqing Gao, and Michael Picheny. Using semantic analysis to improve speech recognition performance. *Computer Speech & Language*, 19(3):321–343, 2005.

[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[11] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[12] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[13] Patrick Hoffman, Georges Grinstein, Kenneth Marx, Ivo Grosse, and Eugene Stanley. Dna visual and analytic data mining. In *Visualization'97., Proceedings*, pages 437–441. IEEE, 1997.

[14] Chang-Hwan Lee, Fernando Gutierrez, and Dejing Dou. Calculating feature weights in naive bayes with kullback-leibler measure. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 1146–1151. IEEE, 2011.

[15] Stuart Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

[16] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[17] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman & Hall/CRC, 1st edition, 2009.

[18] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[19] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[20] Roberto Valenti, Nicu Sebe, Theo Gevers, and Ira Cohen. Machine learning techniques for face analysis. In Matthieu Cord and Pádraig Cunningham, editors, *Machine Learning Techniques for Multimedia*, Cognitive Technologies, pages 159–187. Springer Berlin Heidelberg, 2008.

[21] Giorgio Valentini and Thomas G Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *The Journal of Machine Learning Research*, 5:725–775, 2004.

[22] Jake VanderPlas. Frequentism and bayesianism: a python-driven primer. *arXiv preprint arXiv:1411.5018*, 2014.