ABSTRACT

WIMBERLEY, CHARLES ERIC. PeakPass: A Machine Learning Approach for ChIP-Seq Blacklisting. (Under the direction of Dr. Steffen Heber.)

High-throughput DNA sequencing experiments, particularly ChIP-Seq experiments, suffer from 'problematic' genomic regions that produce repeatedly occurring anomalous results. Although in many cases the reason for this phenomenon is unclear, several studies have suggested that low sequence complexity, a high degree of polymorphism, or the presence of assembly mistakes in the reference genome sequence might be associated with the erroneous results. In an effort to filter noise and reduce false positives in their ChIP-Seq experiments, several groups have created genomic blacklists that can be used to eliminate peaks and sequencing reads that occur in suspicious regions. These blacklists have proven extremely useful, but their creation is time consuming and requires expert knowledge. Due to the rapidly increasing number of sequenced species and genome assembly updates, there is a need to automate the generation of blacklists.

We suggest a classification-based approach to identify blacklist regions. Our approach identifies 13 informative genomic features, including distance to nearest assembly gap, distance to nearest gene, average alignability, and number of unique k-mers. Using these features, we evaluate the performance of multiple classification algorithms. Since blacklist regions only form a small fraction of the genome sequence, our training data exhibits a significant class imbalance. To accommodate for this challenge, we focus on area under the receiver operating characteristic curve and area under the precision/recall curve as performance measures. We also explore different sampling schemes to boost classifier performance, and suggest a novel approach that oversamples the minority class. Instead of reusing or artificially modifying existing data items, our approach selects overlapping, sliding windows of blacklist regions and uses them for training.

We tune classifier parameters using the R library Caret and 10-fold cross validation. The tuned classifiers were then compared by their average performance on multiple test sets derived from the human genome assembly hg19 and expert generated blacklists. Due to its superior performance in our evaluation, we use random forest for our blacklist prediction algorithm PeakPass.

The PeakPass algorithm allows users to generate and tune blacklists for new organisms from a small list of suspect regions. PeakPass takes a whole genome assembly as input, and uses this list of suspect regions or a previously created blacklist to label training samples. The algorithm computes a variety of performance metrics and reports the predicted blacklist in bedGraph format, which is convenient for display in a genome browser. This makes PeakPass a useful tool for researchers that want to generate a new blacklist for recently sequenced genomes, or update an existing blacklist after the release of a new genome assembly update.

PeakPass: A Machine Learning Approach for ChIP-Seq Blacklisting

by
Charles Eric Wimberley

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2018

APPROVED BY:

Dr. Robert Franks Biotechnology Minor Representative	Dr. Nagiza Samatova

Dr. Steffen Heber
Chair of Advisory Committee

BIOGRAPHY

Eric Wimberley grew up in Blacksburg, Virginia. He received a bachelor's degree from Radford University in Computer Science. While he was there he conducted and undergraduate research project working with recombinant DNA and transcription factor-based logic gates. He worked for Harmonia Holdings Group, LLC as a programmer while studying at Radford University for his undergraduate degree, and began work there full time after graduation. He has worked on projects for Harmonia including a compiler, static analysis tool, and optimization problems with genetic algorithms. More recently he has worked on projects for Harmonia including natural language processing, data mining, and ontological reasoning.

He began a master's degree at NC State in Computer Science in 2014. His computer science classes include Artificial Intelligence, Advanced Topics in Machine Learning, and Computational Methods for Molecular Biology. In 2016 he started working on a Biotechnology minor and a thesis under Dr. Heber. He took Biotechnology classes including Epigenetics and Gene Manipulation in Zebrafish. Thesis research topics include machine learning on biological data and high throughput sequencing experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Advances in DNA sequencing technology have made it possible for researchers to determine genomic locations of DNA-protein interactions using high-throughput sequencing and immunoprecipitation of target proteins (ChIP-Seq). Each ChIP-Seq experiment produces hundreds or thousands of peaks that correspond to potential protein binding sites. Large numbers of these experiments are required to map all potential protein binding sites, and binding patterns can be different across different tissues, developmental stages, or environments. The cost of sequencing has fallen significantly in the past decade[1], and the rate of sequencing experiments is likely to increase. While this will make a large amount of data available to researchers, it will also lead to many false positives within the data.

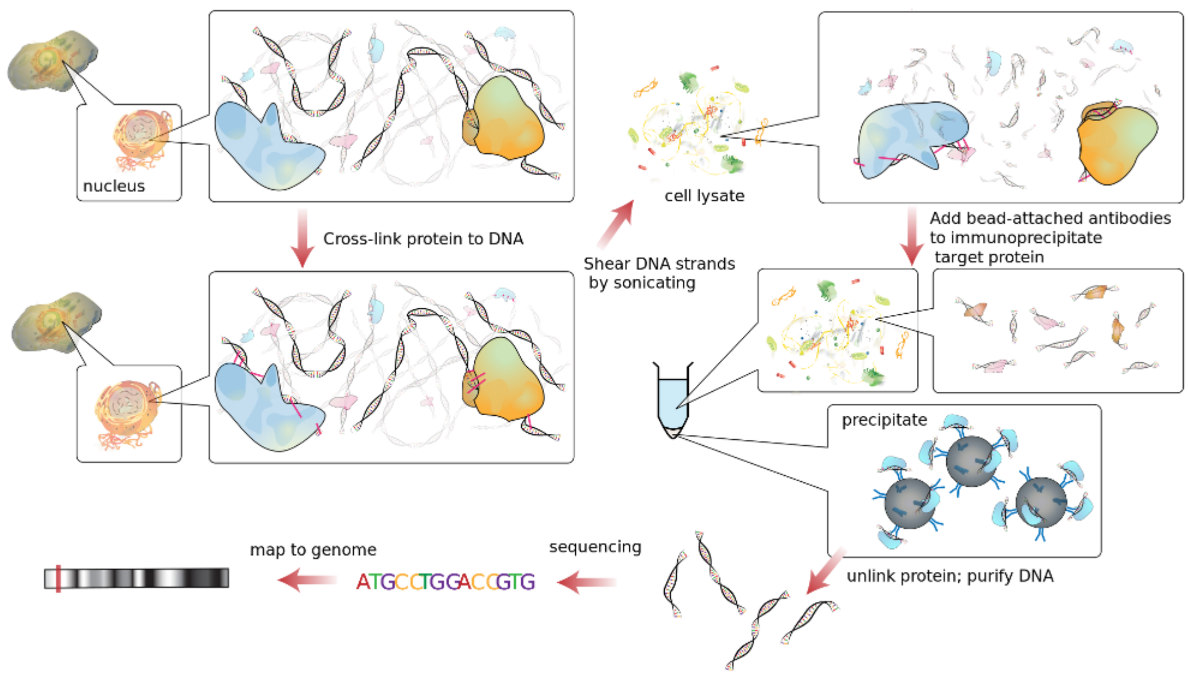

Figure 1.1: **High-Level Illustration of ChIP-Seq Process -** This figure shows a high-level illustration of the ChIP-Seq experiment. First a cellular sample is collected. Proteins are cross-linked to cellular DNA, creating a snapshot of the epigenetic state of the cell. Fragmenting and immunoprecipitation filter out the portions of the genome that aren't bound to a protein of interest. Finally, the isolated fragments are sequenced and mapped to a reference genome to produce a list of binding sites (Jkwchui - Modified Under Creative Commons Attribution-Share Alike 3.0 Unported[2]).

Kharchenko et al. were some of the earliest researchers to notice what they call "large anomalous regions" of the genome, which produced false positive ChIP-Seq results[3]. A large number of reads from their sequencing experiments mapped to these regions, and bioinformatics pipelines marked some of these regions as protein binding locations. However, upon further inspection they were found to have no binding affinity for the protein of interest. It was speculated that the cause of the anomalies was related to sequencing bias or unspecific antibody binding, but a complete explanation was not presented in their paper. The 1000 Genomes project[4], which has created genome assemblies for more than 1000 individuals, calls the portion of the genome that can be easily studied by current sequencing technologies and sequence alignment methods the "accessible genome". Other regions either produce inconsistent results, or sequences cannot be uniquely mapped to them. The authors of the 1000 Genomes paper[4] estimate that only 85% the human reference genome is accessible. However, they note that this number depends on sequencing depth and sequencing technology. Paired-end reads and longer read lengths may increase the accessible portion of the genome.

There are many methods of reducing false positives in ChIP-Seq experiments. Blacklists are one popular method to mitigate these regions that produce inconsistent results, and have been shown to increase the overall quality of pipeline output[5]. A blacklist is a list of regions (defined by a chromosome name, start, and end position) in the chromosome that commonly produce false positive results in ChIP-Seq and other sequencing experiments. Blacklists are designed to catch problematic regions that show up in many experiments regardless of the protocol and controls used. Carroll, et al. report that these problematic regions may be caused by factors such as chromosomal structure and sequence complexity that current controls do not adequately account for[5].

Peaks and reads in ChIP-Seq data that intersect with the elements of a blacklist are considered suspect, and should either be discarded or confirmed by an alternative method depending on the importance of the peak to the researcher. Individual peaks can be confirmed using ChIP-qPCR, which can use different sets of genome-location specific primers to increase specificity over ChIP-Seq[6].

The Kundaje blacklist[7] for the human genome uses sequence features and ChIP-Seq experiments to determine if a region is problematic. Several types of genomic features are associated with blacklists, such as centromere, telomere, and satellite repeats. Other regions are in the blacklist due to extremely low or extremely high mappability and signal, or the presence of a repeat motif. Although Kundaje's method has been used across multiple organisms and genome versions[8], it still relies on a domain expert to generate and update genomic blacklists. The goal of this thesis is to reduce the effort required to during this process by automating blacklist generation.

Kundaje's protocol starts with an automated tool that identifies suspicious 2000 bp regions. An expert then looks through the list and selects suitable blacklist candidates from these regions. The size of the selected regions may be changed if they include regions that appear normal or exclude adjacent abnormal regions. Figure 1.2 shows an example of a blacklisted region in the UCSC Genome Browser[9], along with some tracks that Kundaje used to select blacklisted regions. Relative signal strength tracks are generated using a tool called align2rawsignal[10], and the tracks are available for download along with documentation for the tool. Mappability tracks are generated with a tool called Umap[11].

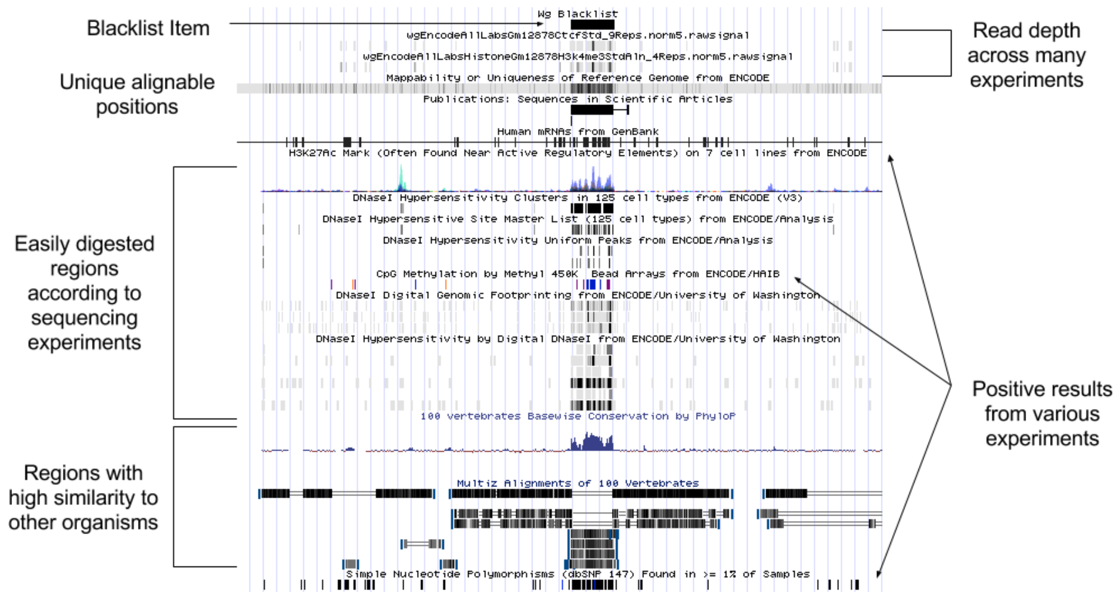

Figure 1.2: **Blacklist Visualization -** The Kundaje blacklist was loaded into the UCSC[9] genome browser for hg19 - The top track labeled "Wg Blacklist" highlights the blacklisted region. The three tracks below the blacklist track are analyzed when deciding whether to blacklist a region. They combine data from multiple experiments to determine relative signal strength. The border of the blacklisted region is clear in the other tracks displayed, which may indicate that the blacklist region contains artifacts in these tracks.

The next few chapters provide background information necessary to understand our blacklists algorithms. Chapter 2 describes ChIP-Seq, an experimental method for identifying protein binding sites using immunoprecipitation and DNA sequencing. This chapter also introduces ChIP-Seq quality metrics. Chapter 3 describes available ChIP-Seq blacklists for various organisms and genome assemblies. Blacklist region types, numbers of blacklist items in different genome assemblies, and length of blacklist regions are analyzed. Chapter 4 provides an overview of various machine learning algorithms that were evaluated for blacklist creation. Chapter 5 provides background information on machine learning methods used for imbalanced datatsets, as blacklist regions are a very small portion of the genome.

The second half of the thesis describes the PeakPass algorithm. Chapter 6 explains the methods and algorithms used to generate blacklists automatically. The chapter also describes how we evaluate the predicted blacklist items and the predicted blacklists' impact on ChIP-Seq performance. Chapter 7 is a breakdown of the quantitative results from the protocols and algorithms described in Chapter 6. Finally, Chapter 8 is a critical discussion of our design decisions, results, and conclusions. Future topics of interest and possible improvements are also suggested.

2. CHIP-SEQ

ChIP-Seq uses immunoprecipitation of DNA-bound proteins and high throughput DNA sequencing to annotate binding sites of DNA associated proteins such as transcription factors and histones. These proteins often control gene expression, and therefore ChIP-Seq can help researchers to determine genes that are part of a regulatory network. Differences in protein binding patterns between tissues or individual organisms can explain the way that epigenetics influences phenotypes.

2.1 Biological Protocol

Although there are some minor differences between ChIP-Seq protocols for different organisms and conditions, the high-level steps remain the same. The following is a generic version of the protocol from Kaufmann's Arabidopsis ChIP-Seq paper[12]:

1. Prepare a cell sample from both treatment and control conditions.
2. Cross-link proteins to DNA using a formaldehyde treatment. This step ”freezes” the state of molecular interactions between proteins and DNA so that proteins remain bound to the DNA during the experiment.
3. Extract and fragment DNA. This step breaks the DNA up into smaller pieces. The resolution of a Chip-Seq experiment is related to the size of the fragments. Running a gel to verify that fragments cluster around a length of a few hundred bp is advised.
4. Filter DNA fragments by size (often using magnetic beads), and remove other cellular components.
5. Immunoprecipitate protein-bound DNA fragments using an antibody specific to the protein of interest.
6. Elute DNA from the immune complex and reverse cross-linking by heating the sample[13].
7. Prepare DNA for sequencing (e.g. adapter ligation).
8. Deep sequencing of the final DNA sample.

2.2 Bioinformatics Pipeline

Sequencing data must be analyzed to determine the location of the binding protein binding sites. There are a variety of analysis pipelines for different experiment types and conditions[14]. The protocol in this section is based on a workflow described by Bailey et al.[15]. The input to a ChIP-Seq pipeline is a set of sequences, typically in FASTQ format. Each entry in a FASTQ file contains the following lines: a name or id, the sequence, whether the strand is forward or reverse (+/-), and an encoded quality score for each nucleotide.

These sequences are then mapped to the genome, and regions with large numbers of structured reads are reported as peaks. The minimum output information for each peak includes the chromosome, a start, and a stop offset. However, many pipelines include information such as the number of reads in the peak, the p-value, and the relative peak center point. Both the mapped reads and annotated peaks can be visualized with a genome browser. The screenshot in Figure 2.1 below illustrates the difference between the level of background signal and a likely peak.

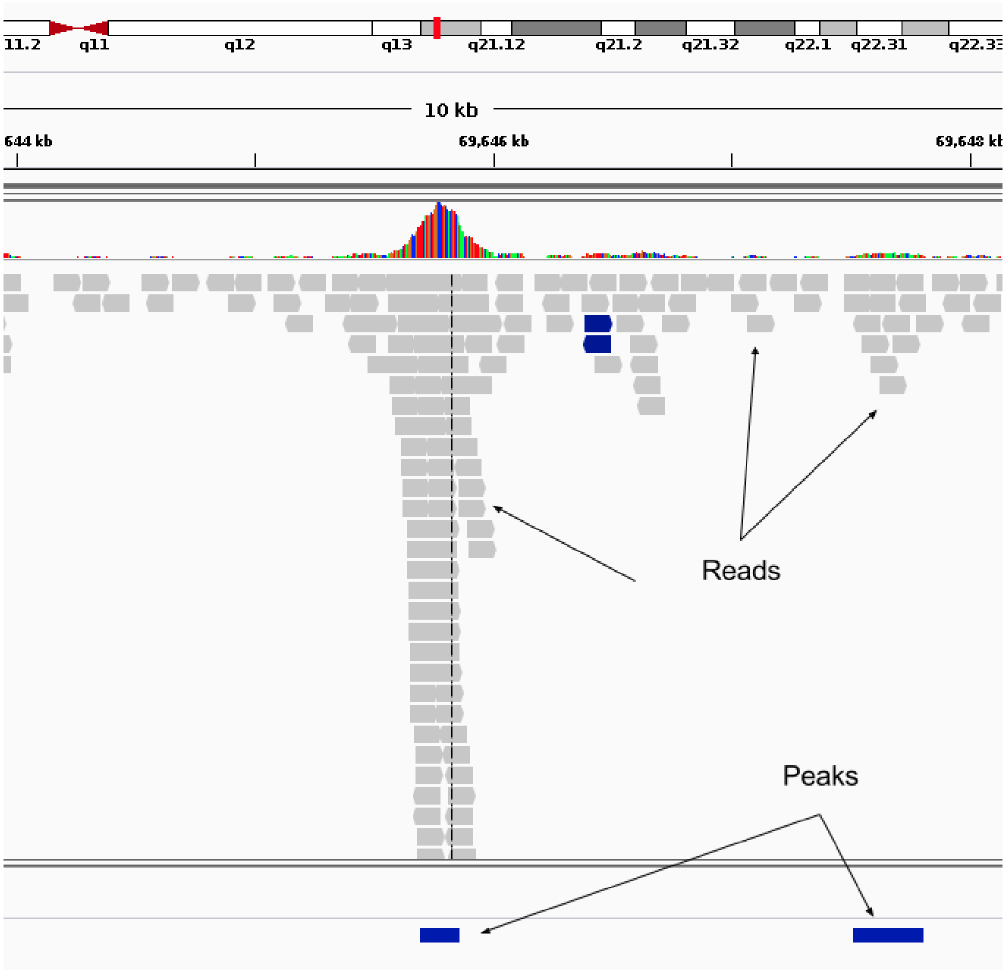

Figure 2.1: **ChIP-Seq Reads Visualization -** A visualization of raw ChIP-Seq reads from a ChIP-Seq dataset targeting the TOE1 protein in humans[16] at a called peak using IGV[17]. The occurrence of many reads significantly increases the chance that the peak is real, and not simply a coincidence.

1. Processing of sequenced reads and quality control. This step often includes trimming of adapters and low quality sequences.
2. Map the filtered reads to the genome. Mapping algorithms and experimental parameters vary in terms of how mismatches are determined, how many mismatches are allowed, and how reads with multiple mapping loci are handled.
3. A peak-caller is used to determine if a cluster of reads is likely to be a peak, or protein binding site. This is determined by the number of reads mapped to that region as well as the shape of the mapping, and the number of biological replicates it appears in.
4. The resulting peaks can be analyzed to find binding motifs, common GO terms, or biological themes. Nearby genes may be regulated by the protein if it is a transcription factor.

The screenshot below is a zoomed out view of Figure 2.1 with genes included. The positions of multiple peaks relative to genes can be seen from this view. There may be some functional relationship between the genes and protein binding sites.

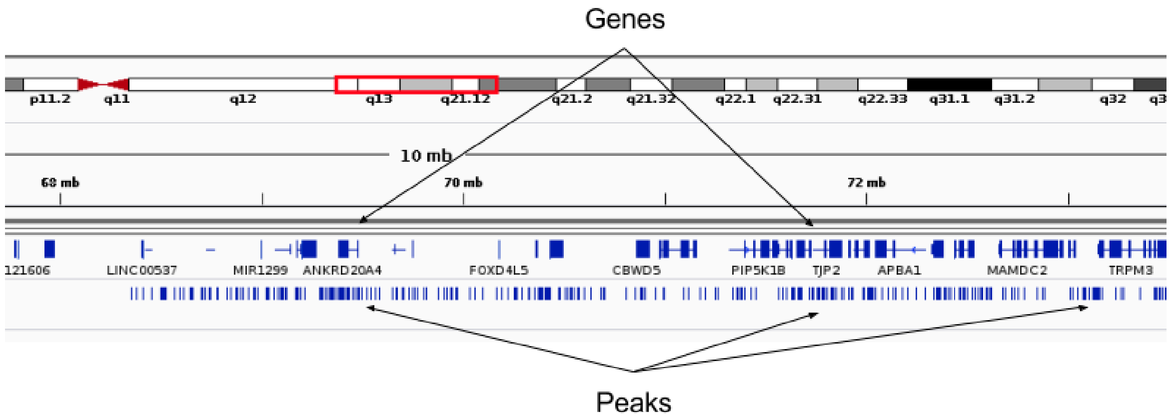

Figure 2.2: **ChIP-Seq Peaks Visualization -** Peaks from the TOE1 dataset[16] are overlaid on annotated genes. In the case of proteins such as transcription factors, proximity between a peak and a gene indicate a regulatory relationship.

Data may be uploaded to databases of different sequencing experiments such as ENCODE[18], so that other investigators can combine multiple experiments and correlate different DNA binding proteins. Enhanced pipelines and new quality assurance methods can be applied to these experiments where raw data is available.

2.3 Challenges in ChIP-Seq

There are several sources of error and noise in ChIP-Seq experiments.

- Insoluble components in the input DNA[12]
- Non-specific antibody binding
- Sequencing artifacts and biases[19, 20]
- Differences between the reference genome and the sequenced sample (e.g. due to a region with high variance among individuals)
- Repetitive regions of the genome and regions with high numbers of multi-mapping reads[15]
- PCR artifacts[20]

Fortunately, there are many methods to control for noise and artifacts that can applied during ChIP-Seq experiments. A partial list of techniques is given below:

- Blacklist regions of the genome with frequent false positives
- Verification using an alternative technique (such as ChIP-qPCR or ChIP-chip[20])[1]
- Multiple biological replicates[21], only select peaks that appear in two replicates
- Compare ChIP signal to input DNA (DNA sampled after fragmentation) as a negative control[21]
- Compare ChIP signal to a mock replicate created with only the secondary antibody[21]
- Correlate forward and reverse strands in the region (strand cross-correlation analysis)[21]
- Set a maximum threshold for the p-value (the probability that the region simply contains a large number of noise reads) of a peak (this number is output by many pipelines)
- Paired-end sequencing instead of single-end sequencing[15]

2.4 ChIP-Seq Quality Metrics

When conducting ChIP-Seq experiments it is important to perform quality control. There are several quality metrics to measure data quality, both for general sequencing experiments and specific to ChIP-Seq. We will focus on ChIP-Seq specific quality metrics in the following section.

2.4.1 Signal-to-Noise Ratio

Reads in the sequencing data that were not attached to the protein of interest during immunoprecipitation are noise in a ChIP-Seq experiment. The ratio of reads in peaks to noise reads is the signal-to-noise ratio, or SNR. While there is no direct way to determine which reads are noise and which are signal, one way to estimate an upper bound on the SNR is to compare the number of reads in an experimental replicate (probable signal) to the number of reads in the input control (noise)[22].

The ENCODE project uses Relative Strand Correlation (RSC) to estimate SNR[21]. The RSC is the ratio of the phantom-peak cross-correlation to the fragment-length cross-correlation after correcting for the minimum cross-correlation[21]. See Equation 2.5 below. A similar metric related to ChIP-Seq noise is Normalized Strand Coefficient (NSC)[21]. NSC is the ratio of correlation at the estimate fragment length to the minimum correlation over all strand-shifts[21]. NSC is defined in Equation 2.6 below.

$$c(x) = \text{Pearson correlation after shifting reverse-stranded reads by x base pairs} \tag{2.1}$$

$$c_{min} = \min_{shift_{min} \leq x \leq shift_{max}} c(x) \tag{2.2}$$

$$read_{len} = \text{the mean length of sequences in the FASTQ file} \tag{2.3}$$

Note that $shift_{min}$ and $shift_{max}$ are defined by the pipeline. The ENCODE pipeline uses -500 and 1500 respectively. In practice, the phantom peak may not be exactly at the read length. This leads some pipelines to estimate the fragment length as the read length when Equation 2.4[23] finds a higher correlation just after the $read_{len}$ and the phantom peak is higher than the true peak.

$$frag_{len} = \operatorname*{argmax}_{read_{len} < x \leq shift_{max}} c(x) \tag{2.4}$$

$$RSC = \frac{c(frag_{len}) - c_{min}}{c(read_{len}) - c_{min}} \tag{2.5}$$

$$NSC = \frac{c(frag_{len})}{c_{min}} \tag{2.6}$$

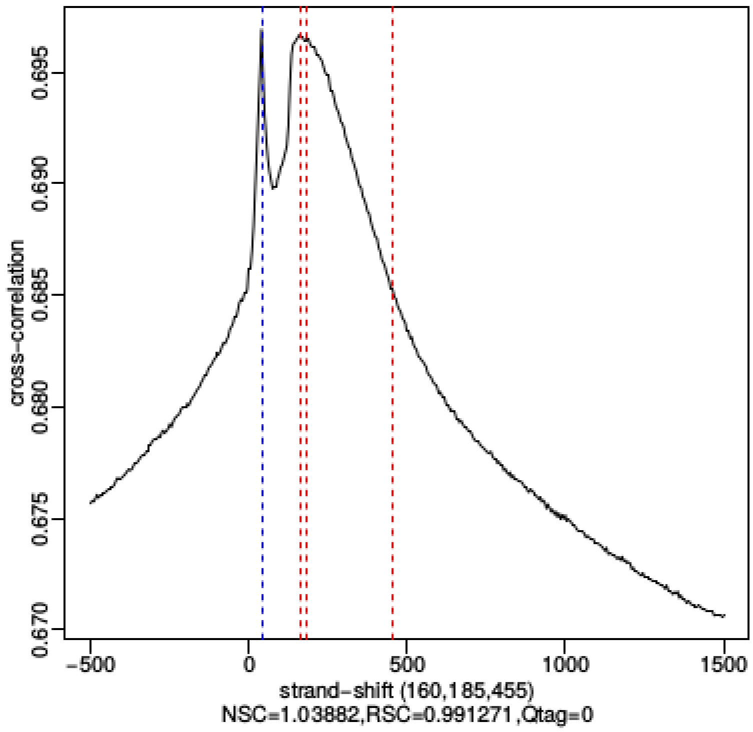

Figure 2.3: **Strand-Shifted Read Cross Correlation -** A cross-correlation graph between mapped reads on both strands. This graph was generated using the SPP peaks caller[3] using a ChIP-Seq experiment that targets the transcription factor AP1 in Arabidopsis thaliana[24]. There is an expected shift between the two strands related to fragment length in single-ended sequencing because only the 5' end is sequenced. The rest of the strand remains unsequenced, even though it was present in the sample. The desired peak occurs when the shift (x-axis) is equal to the average fragment length. A "phantom peak" occurs when the shift is equal to the length of the sequenced reads[21]. The RSC is indicated below the graph to be 0.99, and the NSC is indicated to be 1.04.

This method for estimating signal and noise uses an effect that occurs when mapping single-ended sequencing reads from ChIP-Seq sequence data. If fragments are sampled from the same position of the genome and sequenced, approximately half will be from the forward strand, and half from the reverse strand. Only the 5' end of each fragment is sequenced, resulting in two clusters of mapped reads, one at each end of the cluster of fragments. Figure 2.4 shows a single protein binding site with sequenced fragments.

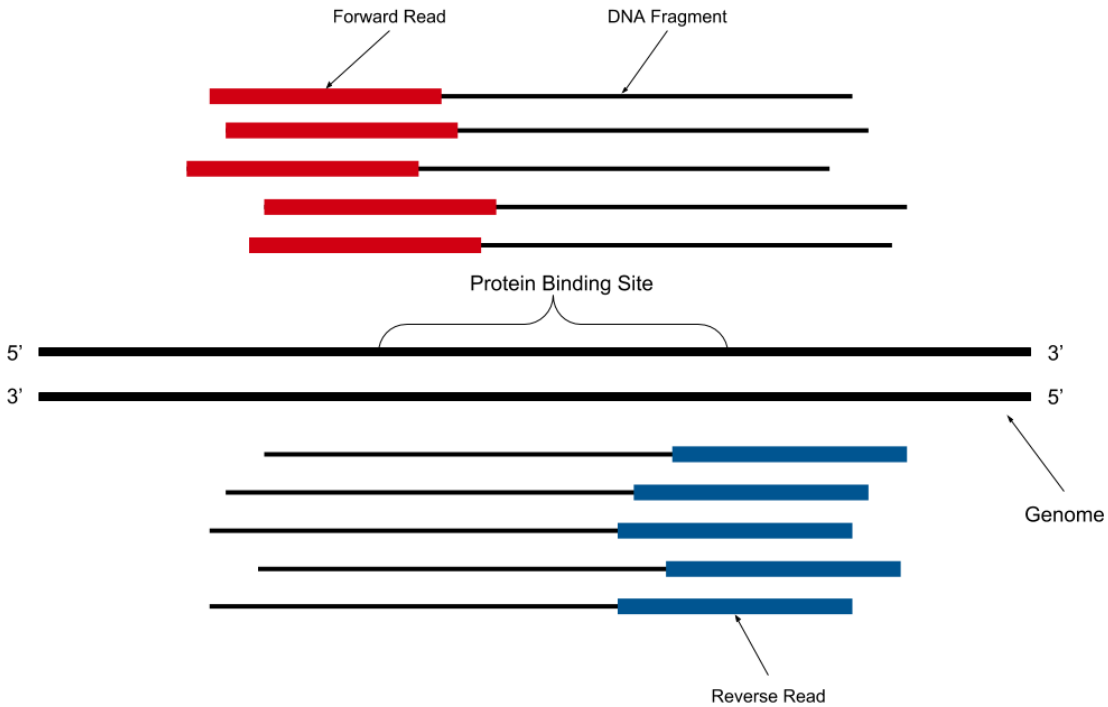

Figure 2.4: **Simple Read Correlation Model -** This figure based on the ENCODE pipeline model[21] shows how DNA fragments, sequenced reads, and a protein binding site are related. Fragments have a distribution of lengths and offsets from the center point of the binding site. When the fragment length is subtracted from the positions of the reverse reads, it results in a set of positions similar to the forward reads.

Noise appears in the cross-correlation curve because a small number of fragments from non-protein binding sites will make it through the immunoprecipitation step. This results in non-zero cross-correlation at shift lengths much different than the fragment length. Figure 2.5 shows a genomic region with only background mapped reads.

Figure 2.5: **Background Reads Cross-Correlation -** Fragments that are not associated with a protein binding site are scattered across this region of the genome, resulting in background cross-correlation.

Ramachandran et al. propose a mappability based explanation for the occurrence of a phantom peak[23] in the cross-correlation plot. If a sequence is unmappable due to not being unique, the reverse complement sequence is also unmappable (because both occur at a different location in the genome). This means that for every unmappable position (for a particular read length), there is an unmappable position at the other end of the read length. Both of these positions are guaranteed to have zero reads, which increases the number of correlating positions regardless of the input read data. At a shift distance other than the read length, this unmappable position will either be compared to a position with reads or without reads. Therefore, cross-correlation at distances other than the read length are decreased by all reads that are shifted to an unmappable position.

2.4.2 Irreproducible Discover Rate and Peak Confidence

Many peak calling algorithms compute a p-value for each peak. This p-value is the probability that the reads at that peak candidate piled up by chance, and not due to protein binding in that region[15]. An additional step to prevent false positives is calculating the False Discovery Rate (FDR). FDR estimates the number of false positive peaks by comparing experimental replicates[25].

The ENCODE project compares candidate peaks between replicates to determine how many peaks appear in only one of the replicates[21]. The rate of single-replicate peaks, also known as "Irreproducible Discovery Rate" (IDR)[2], is defined in Equation 2.8. The ENCODE pipeline uses the IDR for various peak p-value thresholds (Equation 2.7) to determine a suitable threshold for the final peak list.

$$peaks_{sig}(P, threshold) = \{p \in P | pValue(p) \geq threshold\} \tag{2.7}$$

$$IDR(p, q) = \frac{\|p \cap q\|}{\|p \cup q\|} \tag{2.8}$$

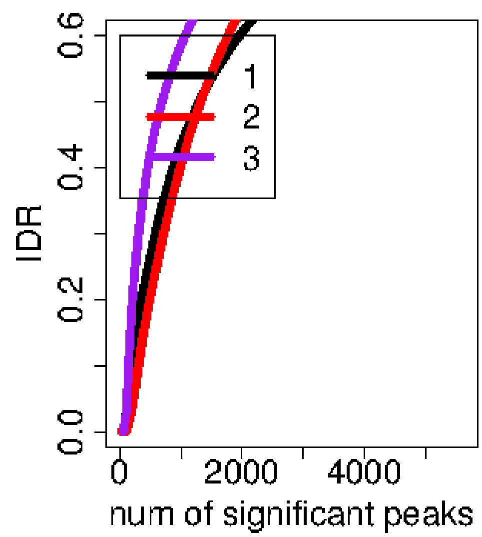

Figure 2.6: **Irreproducible Discovery Rate Graph -** The ENCODE pipeline[21] produces this graph by sorting the peaks in all replicates by their score (p-value) from the peak caller. It then calculates the IDR for all peaks lower (better) than that threshold. Each line in the graph is the comparison between a different set of replicates. This dataset from Dr. Franks' lab includes 3 replicates, so there are 3 combinations of replicates. For replicates of the same experimental condition, the steeper the slope, the lower the replicate quality.

[2]IDR is not an approximation of ChIP-Seq specificity. It is likely that some true positive peaks will occur in only a single replicate because ChIP-Seq does not necessarily capture a majority of the binding sites in the genome.

3. BLACKLISTS

Blacklists are sets of genomic regions that are excluded from a ChIP-Seq experiment because the regions frequently produce false positive signals. Several research groups use blacklists as an important step in their ChIP-Seq or other sequencing protocols. Both ENCODE[26] and the 1000 Genomes project[4] have published their blacklists. Some researchers are already using blacklists for sequencing experiments other than ChIP-Seq, such as experiments to determine DNA copy numbers[27].

3.1 Current Blacklists

Blacklists have been published for several species and genome versions. See Table 3.1 below. The Kundaje hg19 blacklist is well documented relative to other blacklists[7]. The Kundaje, Duke[28], and consensus (a mix of the Kundaje and Duke lists) blacklists label regions with specific reasons for blacklisting, whereas other blacklist simply contain a list of regions.

Table 3.1: **Blacklists -** Blacklists from a variety of different researchers and for a variety of different organisms, as well as some quantitative descriptions of the blacklists. 'Genome' is the organism and genome assembly version the blacklist is designed for. 'Items' is the number of distinct blacklist regions. '#Kbp' is the sum of size of all blacklist regions in Kilobase pairs (Kbp, or 1000 base pairs). 'Coverage' is the percentage of the whole genome assembly that the blacklist covers.

Author	**Genome**	**Items**	**#Kbp**	**Coverage**	**Description**
Kundaje	hg19 (hg37)	226	10275	0.3319%	The original Kundaje blacklist for the human genome.
Duke	hg19 (hg37)	1649	10734	0.3467%	The Duke blacklist has some overlap with the Kundaje blacklist.
Consensus	hg19 (hg37)	411	11588	0.3743%	A blacklist with curated items from both the Kundaje and Duke Hg19 blacklists.
Kundaje	hg19 (hg38)	38	17	0.0006%	An updated blacklist for the human genome assembly version hg38.
Kundaje	mm9	3038	8533	0.3131%	A blacklist for the mm9 mouse genome.
Kundaje	mm10	164	81	0.0030%	A blacklist for the mm10 mouse genome.
Kundaje	dm3	492	1255	0.7441%	A blacklist for Drosophila melanogaster.
Kundaje	ce10	122	394	0.3936%	A blacklist for Caenorhabditis elegans.

DangerTrack[29] provides a bedGraph style track that scores each region from 0 to 1, where 0 is a safe region, and 1 is the most "dangerous". There is no specific threshold mentioned, so the DangerTrack does not fit into the table above.

The graphs below show length distribution of blacklist regions for different genomes. When evaluating a novel blacklists it may be useful to compare the lengths of regions in the new blacklist to existing blacklists.

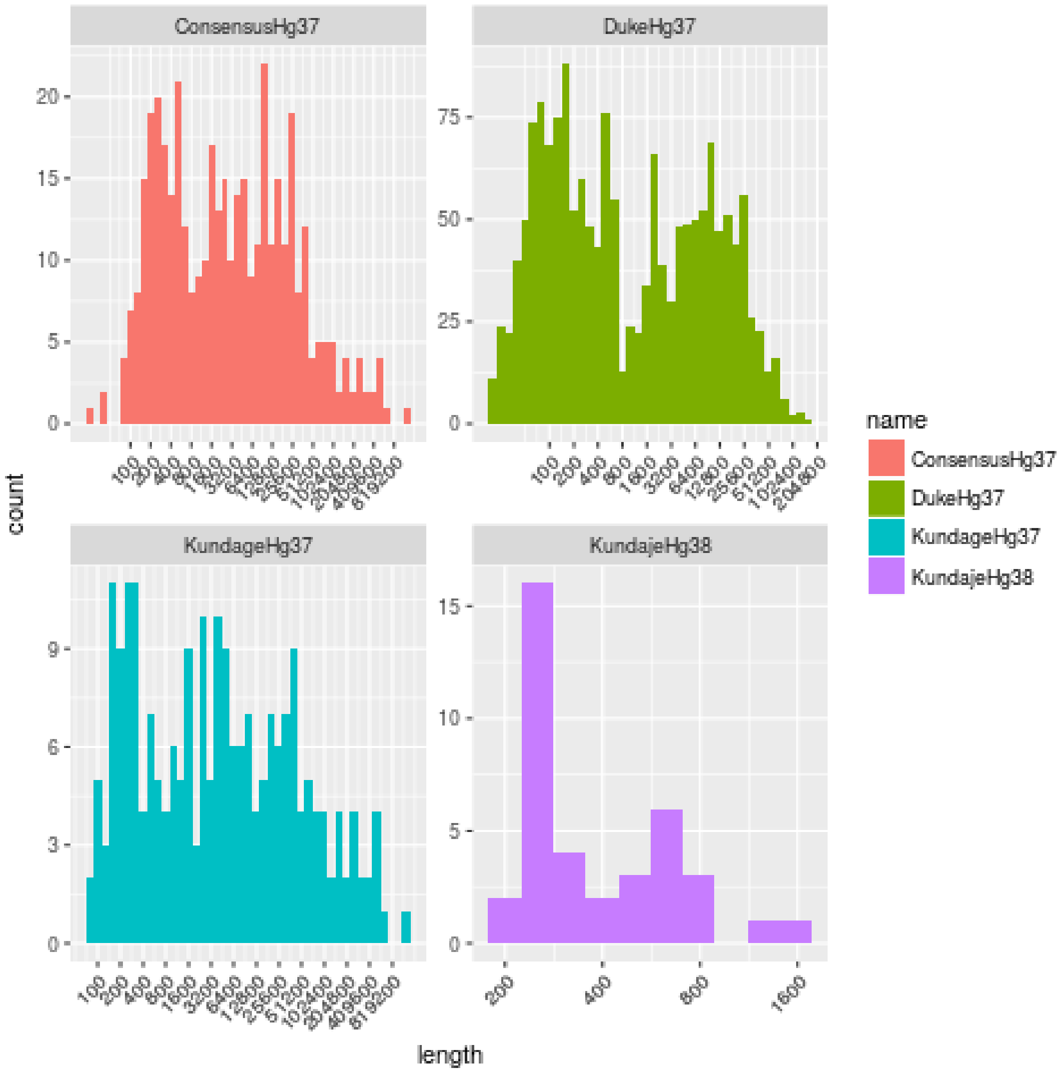

Figure 3.1: **Length Distribution of Blacklist Regions in the Human Genome -** This histogram shows the lengths of blacklist items in blacklists for the human genome.

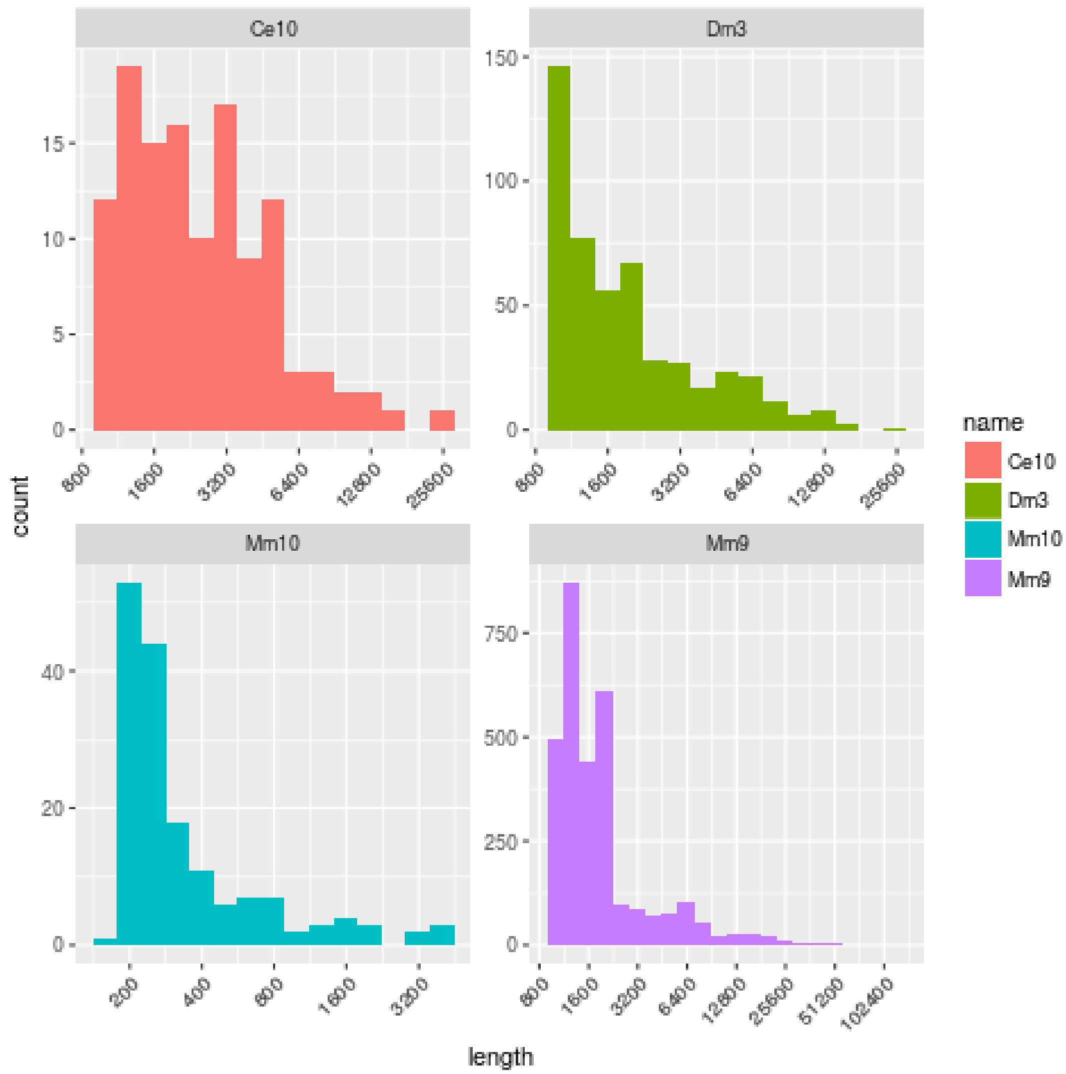

Figure 3.2: **Blacklist Region Length Distributions in Other Genomes - Ce10:** Length of blacklist items for *Caenorhabditis elegans*. **Dm3:** Length of blacklist items for *Drosophila melanogaster*. **Mm10:** Length of blacklist items for the latest genome assembly for *Mus musculus*. **Mm9:** Length of blacklist items for an older genome assembly for *Mus musculus*.

Researchers initially noticed problematic genomic regions in large scale sequencing experiments because of the large number of reads that mapped to these regions[3]. A simple method for creating blacklists would be to perform a large number of experiment, validate the results, and then record the false positives in a database. Regions with frequent false positives would be added to the blacklist. While this may work for organisms that are commonly studied, such as hg19 and later versions of the human genome assembly, there is not enough information available to do this in all organisms. To solve this problem for the ENCODE project[26], Kundaje proposed a partially automated solution[7]. He created a script to generate blacklist candidates, and then manually validated them against a set of ChIP-Seq experiments.

He then applied a similar approach to several other organisms. Kundaje categorizes his blacklist items based on the presence of certain sequence repeats. He also incorporated mappability into his blacklist creation process, which is calculated based on the number of times a sequence can be mapped to the genome.

3.2 Kundaje Blacklist Details

The Kundaje[7] human blacklist for genome assembly version hg19 contains six different types of artifacts. The table below gives an overview of the different types of artifact types, as well as their frequency in the blacklist[1]. The Duke blacklists contains some additional artifact types not covered by Kundaje, however it has a large degree of overlap with the Kundaje blacklist.

Table 3.2: **Kundaje Blacklist -** Types of blacklist regions found in the Kundaje blacklist for hg19.

Region Type	Regions	Description
Centromeric Repeat	76	Centromeric regions contain long repetitive sequences[30]. However, the borders between centromeres and normal chromosome regions are often not well defined. The blacklist regions in this category contain centromeric repeats and some non-repetitive sequences. These regions are often soft-masked. Soft-masked regions appear in lower-case letters in FASTA files (as opposed to hard-masking, which replaces regions with the character 'N').
High Mappability Island	7	These regions have high average mappability, but may have interspersed low mappability loci[7]. A random sample from a uniform distribution of reads from this region may result in peak-like artifacts purely due to mappability effects. If two low mappability loci are in close proximity, the area between them will likely have relatively higher signal when the uniform distribution of reads is mapped.
Low Mappability Island	100	Regions in this category have a very low average mappability, but may have interspersed high mappability loci[7]. A random sample from a uniform distribution of reads from this region may result in peak-like artifacts purely due to mappability effects. A single high mappability region may look like a peak relative to the background mappability.
Satellite Repeat	36	Satellite repeats are similar to centromeric repeats, but have shorter repeat motifs and are not near centromeres[30]. While most of the reads from these regions will not map uniquely, there are some non-repetitive islands within the region. These regions are often soft-masked.
snRNA	2	Small nuclear RNA (snRNA) are non-coding RNA molecules[31]. The regions that code for these RNA molecules are often repetitive.
Telomeric Repeat	5	Telomeric repeats are similar to centromeric and satellite repeats but are near telomeric regions instead. These regions are often soft-masked. An example of a telomeric repeat sequence from the blacklist is below:
Total	226	-

3.3 Blacklists and ChIP-Seq Quality

The success of a blacklist can be measured by its impact on the quality of ChIP-Seq experiments. Carroll et al. published a study in which they applied the Duke and Kundaje blacklists to 445 ChIP-Seq datasets from ENCODE and other ChIP-Seq experiment repositories, and then evaluated several quality metrics

[1] Additional tables describing blacklists are available in the appendix.

before and after the application of blacklists[5]. The authors discovered an enrichment of duplicated reads within the blacklist regions (reads that map to the same position in the genome).

When reads mapped to blacklisted regions were removed before running the pipeline, the phantom peak in the strand cross-correlation graph was significantly reduced without much change to the fragment length peak (an increase in RSC). The authors noted that removal of duplicate reads reduces cross-correlation across the entire shift range, and only slightly improves RSC. This suggests a greater improvement in the SNR of the experiment by applying a blacklist than simply removing duplicate reads.

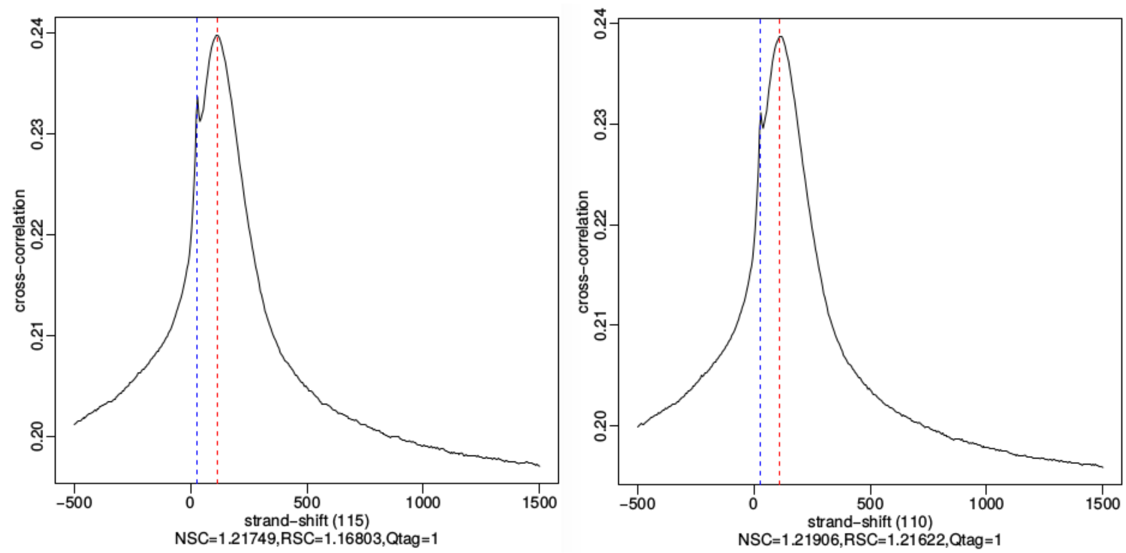

Figure 3.3: **Strand Cross-Correlation Analysis with Blacklist Filtering - Left:** The ENCODE pipeline strand cross-correlation for a CHD2 ChIP-seq dataset[32]. **Right:** The Kundaje blacklist applied to the reads in the dataset result in s lightly increased NSC and RSC.

4. LEARNING ALGORITHMS

To automate the production of blacklists, this thesis applies classification algorithms to genomic data. Classification algorithms take training examples as input. Typically, the format of training data is a matrix where each row is a different training example. Columns in the matrix correspond to different features, and the final column is a class label indicating the desired output of the classification algorithm. Features are often numeric, such as length, but can also be categorical such as color. Learning and validating classification models from training data is a form of supervised learning.

Both blacklisted and normal regions of the genome will be given to the algorithm as training data. After training, the classifier will be used to predict regions of the genome that it has not been given as input before. This is known as validation, and can be used to estimate the performance of a model during application.

We have tested the following popular classification algorithms on the blacklist classification problem:

- Naive Bayes
- SVM (Support Vector Machine)
- RPart (Recursive Partitioning)
- Random Forest
- KNN (K-Nearest Neighbor)
- ANN (Artificial Neural Networks)

4.1 Naive Bayes

Naive Bayes uses training data to calculate the probability of a class label given the observed feature values[33]. The classifier is based on Bayes' Theorem (shown in Equation 4.1). Informative features will have different distributions for different classes.

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{4.1}$$

To make predictions, the algorithm predicts the probability of all feature values given the class label and multiplies them together. The class that has the highest product of probabilities is the predicted class, as shown in Equation 4.4 below[34]. Note that this assumes conditional independence of features since the probability of a feature value given other feature values is never computed.

$$Class_{prediction} = \underset{Class}{\operatorname{argmax}} \Big(P(Class) \prod_{i=1}^{n} P(x_i|Class) \Big) \tag{4.2}$$

For a classifier with a single feature and two classes with equal frequency, Naive Bayes might create a classifier with a decision boundary illustrated in Figure 4.1 below. A probability distribution is calculated for each class label / feature pair.

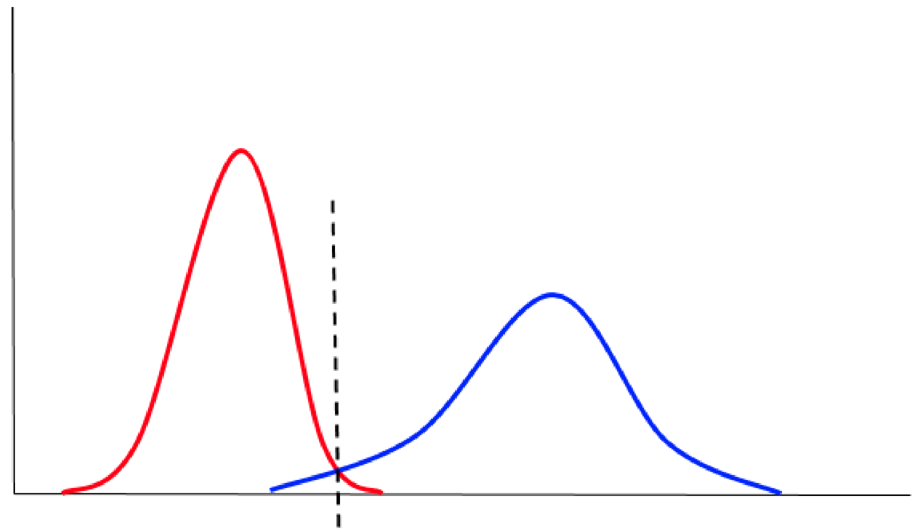

Figure 4.1: **Naive Bayes Probability Distributions -** The probability distributions for a single feature Naive Bayes classifier. The red probability distribution represents the probability of different values of the independent variable in class A. The blue probability distribution represents the distribution of the independent variable in class B. If the independent variable is to the left of the dashed line, the probability of the value of that variable is highest in the distribution for class A. To the right of the dashed line, the probability is highest for the distribution for class B.

Training a Naive Bayes model is fast. Calculating the mean and variance of a dataset is complexity $O(NM)$ where N is the number of training examples and M is the number of features. Prediction has complexity $O(M)$ where M is the number of features. These probability distributions can be easily visualized by graphing them, so human understanding of the model predictions is possible. Due to its assumption of conditional independence, the Naive Bayes algorithm may perform poorly on datasets with highly correlated features. This assumption is rarely true of all features in practice.

4.2 SVM (Support Vector Machine)

A linear Support Vector Machine, invented by Vapnick et al.[35], creates a linear decision boundary in Euclidean feature space[35]. The decision boundary is selected such that it produces the largest margin between training examples of different class labels, as shown in Figure 4.2.

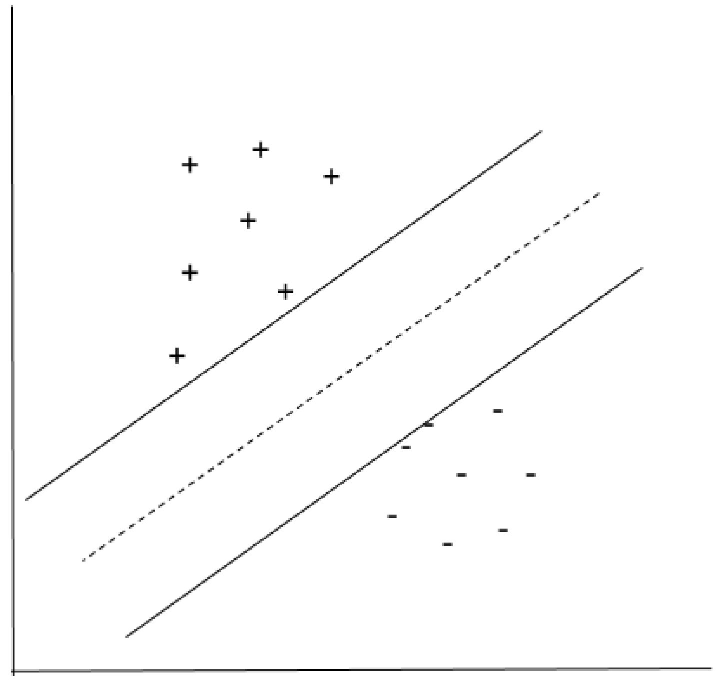

Figure 4.2: **Linear SVM Visualization -** This plot shows a decision boundary as calculated by an SVM with a radial kernel. The SVM algorithm attempts to place the decision boundary (shown as a dashed line) exactly in the middle of the margin between the two class labels. When classifying an unknown data point the algorithm simply determines which side of the boundary the point is on. The margin between the clusters for each class is shown as two parallel solid lines. This figure is based on the SVM concept from the original SVM paper[35].

While basic SVMs use a linear decision surface such as a line or plane, it is possible to use a non-linear decision boundary such as a circle or an N-dimensional ellipsoid. Figure 4.3 below shows a circular decision boundary.

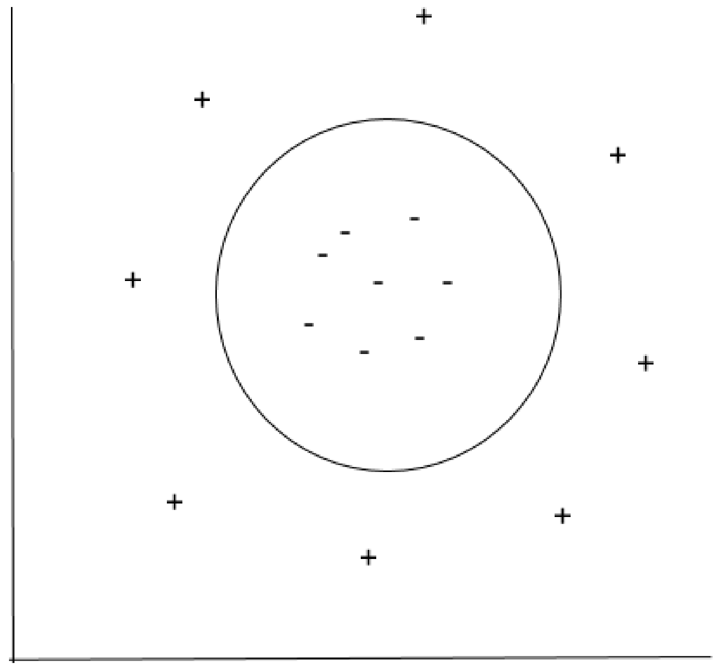

Figure 4.3: **Radial SVM Visualization -** This graph shows the decision boundary as calculated by an SVM with a radial kernel. The SVM algorithm attempts to place the decision boundary (shown as a black circle) exactly in the middle of the margin between the two class labels. When classifying an unknown data point the algorithm simply determines which side of the boundary the point is on.

To set the threshold for the example above, the SVM algorithm maps data points into an additional dimension using a kernel function, such as the radial basis function (RBF), defined in Equation 4.3[36]. Defining the decision boundary using this additional dimension can be easier for some problems.

$$k(x, y) = exp\Big(\frac{abs(x - y)^2}{\sigma}\Big) \tag{4.3}$$

SVMs have a fast prediction time. They perform well on difficult problems, such as handwriting recognition[35]. The downside of SVMs is that they often require different implementations for different kernel functions.

4.3 RPart (Recursive Partitioning and Regression Trees)

Rpart is a tree-based learning algorithm, originally described by Breiman, et al. and ported to R by Brian Ripley[37]. Rpart uses training examples to create a single decision tree that partitions data points into different groups based on feature values (a split). Each split is determined by how well it divides the training data by class. Figure 4.4 shows an example decision tree created by Rpart.

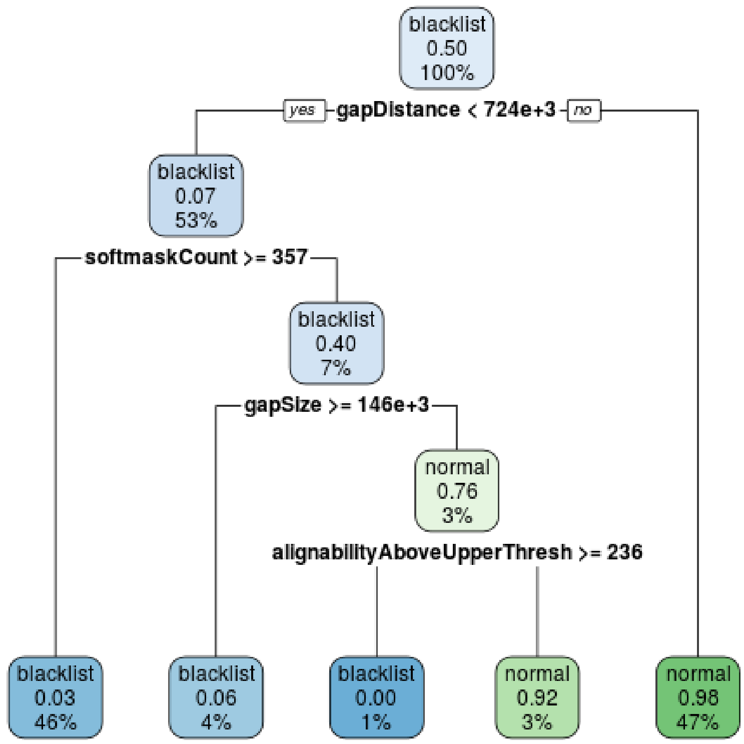

Figure 4.4: **Rpart Tree Visualization -** An example rparted tree created using the rpart.plot library[38] - each node shows the majority class, probability that the data point has the normal class label, and percentage of data from the training set that matches that path through the tree.

The algorithm to train an RPart decision tree is similar to other recursive tree training algorithms. The steps below describe a general method to recursively train decision trees. The difference between these algorithms is often the metric they use to determine the optimal data split[39].

1. Determine the feature that best splits the training examples in the current node
2. Find the optimal split threshold for that feature
3. Divide the training samples by the threshold and assign them to each of two child nodes
4. Recursively split the left and right nodes until some condition is reached

RPart provides additional parameters such as 'minsplit', which prevents a node from splitting if it contains too few training examples[37]. This parameter affects training performance and tree complexity. As seen in Figure 4.4 the models produced by RPart are easy for humans to understand. In fact, a human could quickly use the decision tree to predict unknown data points without a computer.

However, Russell and Norvig note that not all classification problems can be solved with decision trees[40]. A single decision tree is also susceptible to overfitting, a problem which can be solved with multiple trees, as described in the section about random forests below.

4.4 Random Forest

Random forest classifiers, invented by Ho[41], and refined by Breiman and Cutler[42], are made up of a large number of decision trees, as seen in Figure 4.5. When classifying unknown inputs, the algorithm uses all trees to reach a decision on the class of the data point (bagging)[42]. For classification, each tree votes on the final class. For regression problems (such as a regression on class label probability) the output value of all trees can be averaged[43].

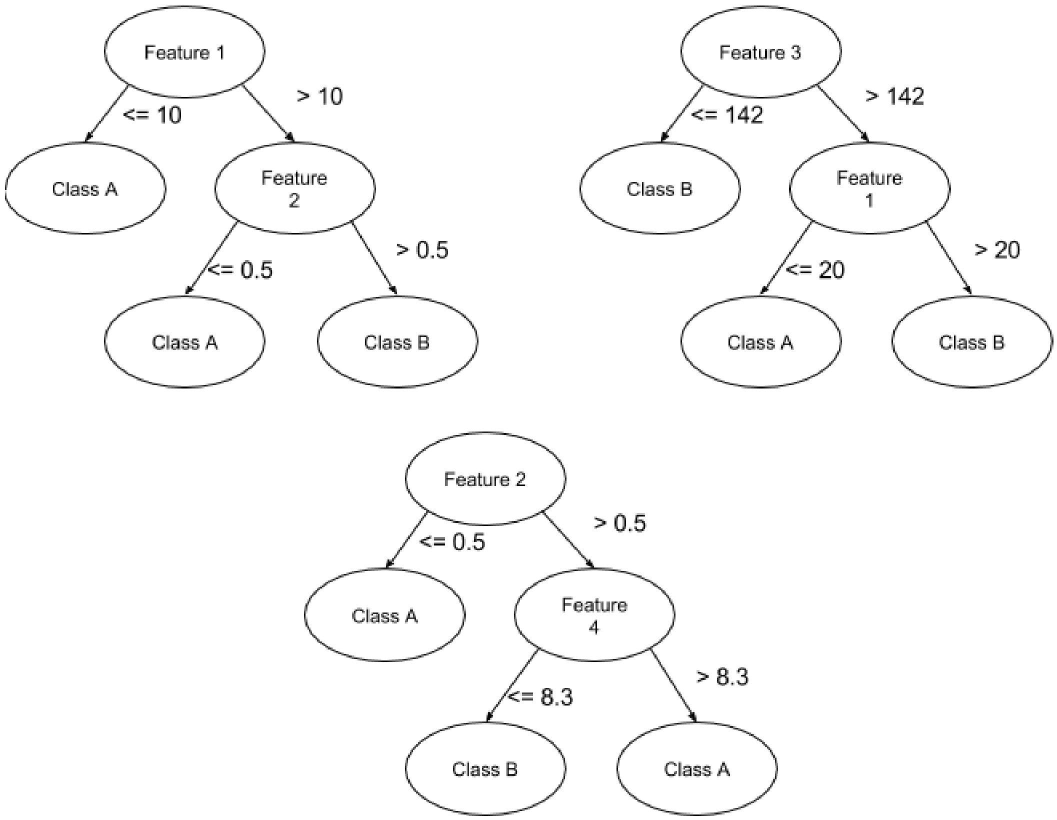

Figure 4.5: **Example Random Forest -** A simple random forest with three trees, two classes, and four features. Note that each tree uses a different set of features.

Breiman claims that error of a random forest is bounded by the error of the individual trees in the forest and their similarity[42]. If two trees are similar, the model becomes biased towards their predictions. If two trees were identical, this would be the same as a single tree with two votes for the final class label. To promote differences between trees, each tree is trained on a subsample of the full training dataset, and a subsample of the feature set. This causes the random forest to be more resistant to overfitting and outliers than a single decision tree[43].

Prediction using a random forest is relatively simple: each decision tree is applied to the unknown data point, and votes for a particular class label. The class label with the majority of votes is selected. However, we found that random forest models take a long time in practice to train. A random forest model is not easily human understandable, unlike a single decision tree, because hundreds or thousands of decision trees may be used.

An important advantage of the random forest algorithm is its ability to estimate the importance of individual features[44]. The training samples that are not part of the training subset ("out-of-bag") for each tree are predicted using that tree, and the error rate is recorded. After the whole forest is trained, the difference in performance between trees that include a feature and trees that exclude the feature is calculated. If the difference is large this indicates that the feature is important.

4.5 KNN (K-Nearest Neighbor)

K-Nearest Neighbor algorithms directly compare data points from the training set to unknown data points. The earliest known description of KNN is by Fix and Hodges[45]. For numeric features, the algorithm simply determines the Euclidian distance between the unknown instance and the training instances, and predicts that the unknown instance has the class label of the majority of the k nearest instances. A visualization of the KNN algorithm is shown in Figure 4.6. One implementation of KNN, IBk, tries to reduce the time and space complexity of K-Nearest Neighbor by only using the most "informative instances" of the training set[46].

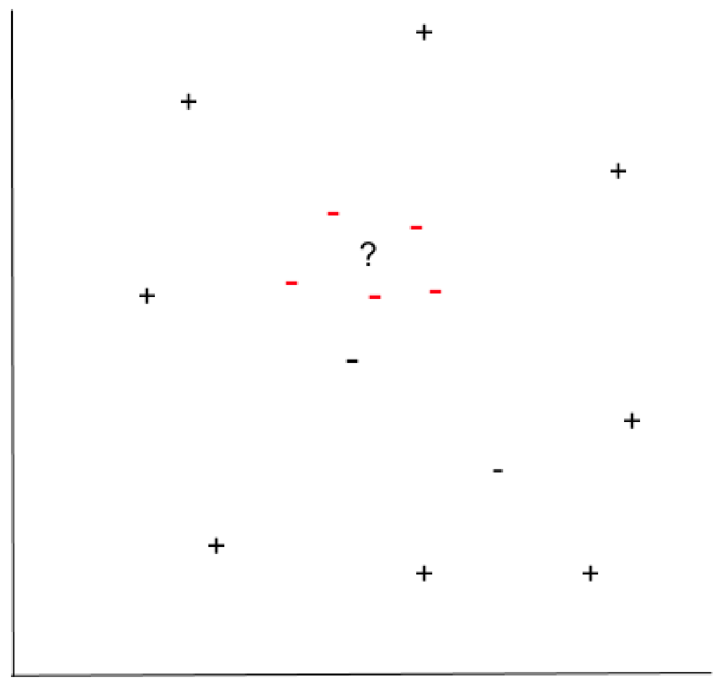

Figure 4.6: **KNN Visualization -** A simple K-nearest neighbor classifier compares the unknown data point (shown as a question mark) to the k most similar data points (shown in red), and labels it using the majority function on those neighbors.

K-Nearest Neighbor directly uses training data to predict unknown points, so the training performance is very fast. Prediction involves calculating the distance between the unknown data point and training points, however in practice only some of the distances must be calculated.

4.6 ANN (Artificial Neural Networks)

Artificial neural networks are a graph based learning algorithm that are inspired by neural networks in the brain[47]. Neural networks are often composed of three or more layers (input, hidden, and output)[48]. Each node in the network computes its output using a combination of inputs from other nodes, the feature set, and bias inputs.

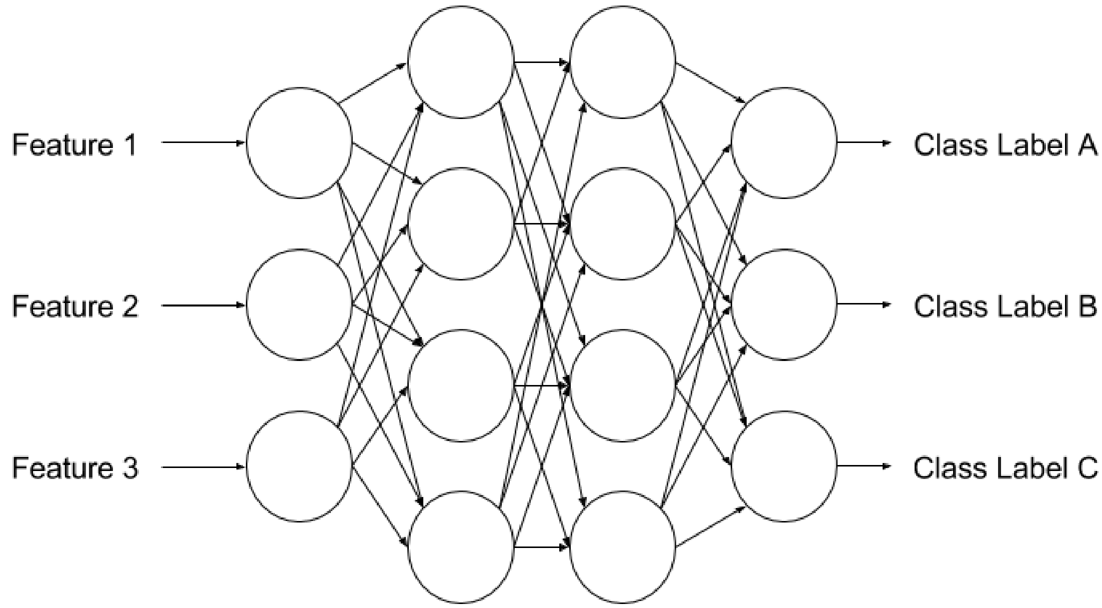

Figure 4.7: **ANN Visualization -** A simple neural network with an input layer, two hidden layers, and an output layer. Each neuron combines inputs from the previous layer. Neurons in the input layer are given feature values as inputs, and output neurons for a classification problem usually output the probability of a given class label.

Each neuron in the network sums all of its inputs multiplied by a weight[40]. This is defined by Equation 4.4 below. The total input is then used as input for an activation function. A common activation function is the sigmoid function[49], which is defined in Equation 4.5.

$$input_{total} = \sum_{i=1}^{n} weight_i \times input_i \tag{4.4}$$

$$activation(x) = \frac{1}{1 + e^{-x}} \tag{4.5}$$

Artificial neural networks are considered tolerant to noise[40] and perform reasonably well on imbalanced data[50]. However, they don't provide transparent explanations for how they arrive at a class label decision.

5. LEARNING FROM IMBALANCED DATA

Imbalanced classification problems arise when a class label is rare relative to other class labels. There is no exact cutoff at which a problem is considered imbalanced. Krawczyk suggests that a problem is imbalanced if the class ratio is between 1 to 4 and 1 to 100[51]. However, several applications with significantly more extreme ratios exist. Examples of imbalanced problems include:

1. Cancer diagnosis (approximately 1 in 1000 individuals per year[1]) [52]
2. Fraudulent credit card transactions
3. Quality control problems during end product inspection
4. Cyber security incidents in log files

Blacklist learning is a highly imbalanced problem. For example, the Kundaje blacklist covers approximately 10 million base pairs, whereas the hg19 genome assembly is over 3 billion base pairs. This means that the imbalance ratio of blacklist to normal class labels is 1 to 300.

This chapter covers some metrics that assure classifier performance on imbalanced data, such as Area Under the Receiver Operating Characteristic (AUC ROC). It also covers methods for improving classifiers in imbalanced scenarios, such as undersampling the majority class. Examples of pitfalls with specific classification algorithms, and how they relate to improved methods will be discussed.

5.1 Standard Performance Metrics

One of the most basic performance metrics for classification is accuracy (Equation 5.1). Accuracy is simply the number of correct predictions divided by the total number of predictions.

$$Accuracy = \frac{TruePositives + TrueNegatives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives} \tag{5.1}$$

In a credit card fraud classification problem true positives are fraudulent transactions that are correctly classified as fraudulent. True negatives are customer transactions that are correctly classified as not fraudulent. If a non-fraudulent transaction is classified as fraudulent, this is a false positive. A fraudulent transaction that is marked as non-fraudulent is a false negative.

Unlike accuracy, sensitivity (Equation 5.2[53]) and specificity(Equation 5.3[53]) are class-label specific.

$$Sensitivity = Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{5.2}$$

$$Specificity = NegativeClassRecall = \frac{TrueNegatives}{TrueNegatives + FalsePositives} \tag{5.3}$$

Below is a confusion matrix that shows the frequency of predictions for each class label on each row, and the frequency of class labels in the data on each column. This confusion matrix is typical of a problem with a class imbalance of 1 to 10. If the negative class for the problem above represents a predicted fraudulent credit card transaction (a rare event), the credit card company can call the card holder and double check to see if they intended to make the transaction. In the worst case this is a minor inconvenience for the card holder. If the application is cyber security, and the negative class represents

[1]Patients subjected to cancer screenings by a doctor probably represent a more balanced classification problem.

the need to hire an expensive cyber security consultant, then the classifier results in 15 false alarms that could be very expensive.

Table 5.1: **Imbalanced Confusion Matrix -** A confusion matrix from an imbalanced problem with a class imbalance ratio of approximately 1 to 8.

Prediction	**True Positive**	**True Negative**
Positive	226	4
Negative	15	25

Accuracy can be misleading when it comes to imbalanced datasets. This is because the majority class has a greater impact on both the numerator and denominator in the accuracy equation and can overshadow performance problems in the minority class. A poorly performing classifier on an imbalanced dataset may return the majority class label nearly all the time, and still achieve a high accuracy. Figure 5.1 below shows how the K-nearest neighbor algorithm can produce a poor classifier from imbalanced training data.

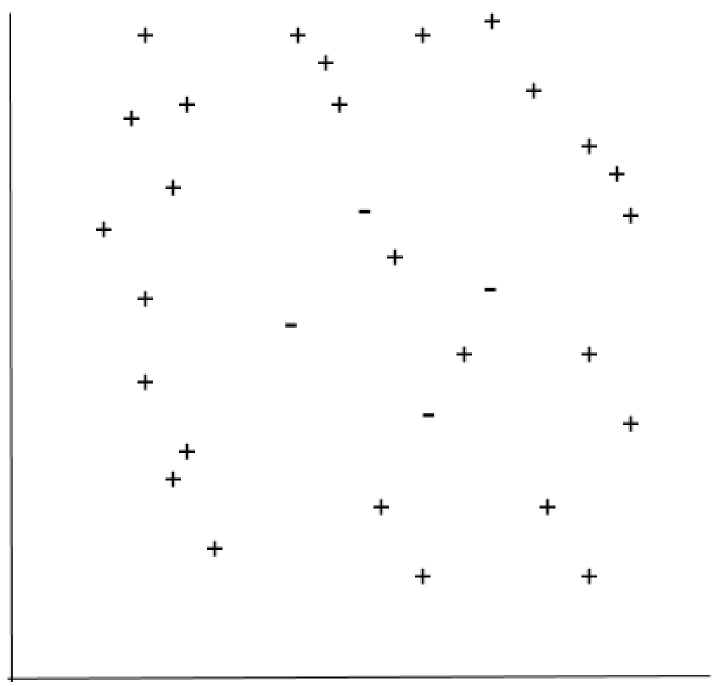

Figure 5.1: **Imbalanced K-Nearest Neighbor Classifier -** A simple K-nearest neighbor classifier with k=5 may have a very high training accuracy even if the negative training examples are all classified as positive. Even a small number of noisy data points from the majority class (seen as positive data points close to the center of the circle) is close to the total signal of the negative data points.

The classifier in Figure 5.1 above has a specificity of 0.0, which would immediately alert the researcher of a problem. Whether sensitivity and specificity are enough to mitigate imbalanced data depends mostly on application.

5.2 Performance Metrics for Imbalanced Problems

Metrics that can be used to assess model performance in imbalanced problems include precision, F-Measure, and AUC ROC[50]. Precision (Equation 5.4[53]) uses true positives and predicted positives, so it is less affected by the ratio of positive to negative class labels (imbalance ratio). F-measure (Equation 5.5[53]) is the harmonic mean of precision and sensitivity. If a model can increase both precision and sensitivity, rather than simply trading them off, it has increased predictive power.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{5.4}$$

$$F - Measure = 2 \times \frac{Precision \times Sensitivity}{Precision + Sensitivity} \tag{5.5}$$

In addition to reporting a class label, many classification algorithms can output an estimated probability that the class label is correct. Classification algorithms use various mechanisms to estimate this probability. Whereas an ANN might directly perform a regression on the class label probability function, decision tree based algorithms may use class purity at leaf nodes to estimate the probability of the selected class label.

Area under the curve (AUC) of the receiver operating characteristic (ROC) is another commonly used metric to compare classifier performance. ROC plots true positive rate (Equation 5.6) versus false positive rate (Equation 5.7) for varying classifier discrimination thresholds[53]. The AUC ROC is simply the area under this curve. AUC ROC can be used as the basis for a loss function, or to compare the performance of two models. The closer the AUC ROC is to 1.0, the better the model.

$$TruePositiveRate = \frac{TruePositives}{AllPositives} \tag{5.6}$$

$$FalsePositiveRate = \frac{FalsePositives}{AllNegatives} \tag{5.7}$$

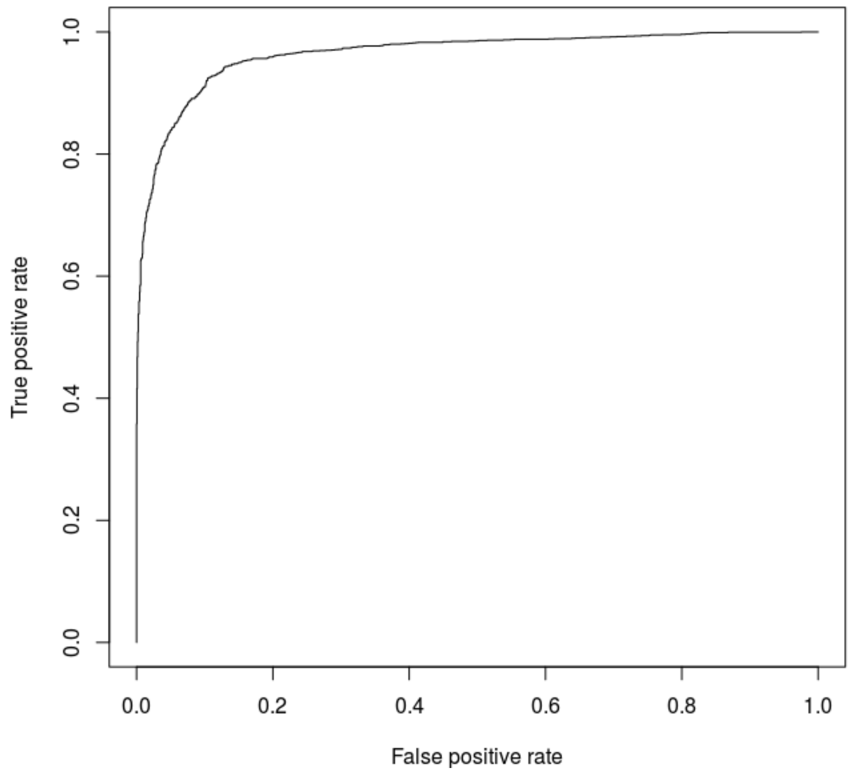

Figure 5.2: **ROC Curve -** An ROC plot generated from a SVM model with the ROCR R package[54]. AUC ROC is the area under this curve. A high AUC ROC indicates a good predictor, whereas an AUC ROC of close to 0.5 indicates a poor predictor (one that is not better than a random guess).

The area under the precision recall curve is also useful for characterizing models on imbalanced data. This curve is calculated by plotting precision and recall with varying classifier discrimination thresholds (similar to AUC ROC). AUC Precision/Recall can reveal problems that AUC ROC cannot in scenarios where the number of false positives is large relative to the number of true positives[55]. The false positive rate can still be low even if there are a large number of false positives if the number of true negatives is large. However, precision is low in this scenario, leading to a low AUC precision / recall.

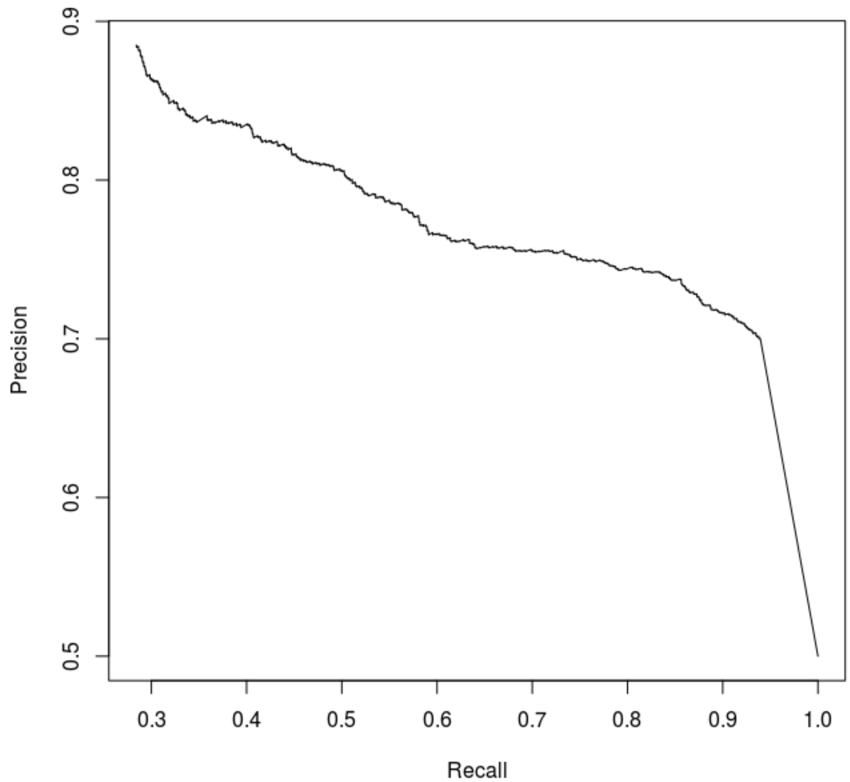

Figure 5.3: **Precision Recall Curve -** A precision/recall plot generated from a KNN model with the ROCR R package[54]. AUC precision/recall is the area under this curve. A high AUC precision/recall indicates a good predictor, whereas a low AUC precision/recall indicates a poor model. Note that there is no threshold that results in a recall of 0, so the precision at 0 recall is estimated using the next closest point.

5.3 Methods on Imbalanced Problems

5.3.1 Larger Training Sets

While not always possible, collecting additional training samples can help alleviate classifier performance problems on imbalanced training data. In Figure 5.1, the minority class forms a cluster[55]. The hope in collecting more data is that minority class clusters become more distinct. Figure 5.4 shows that after collecting more data the proportion of positive to negative samples remains the same, but a KNN classifier with k=5 can more easily predict minority data points.

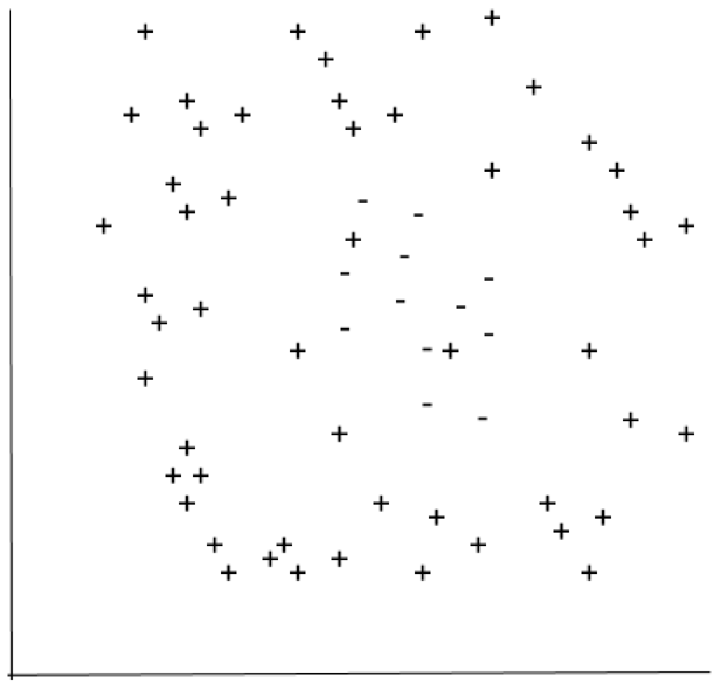

Figure 5.4: **Increased Training Set Size -** The data in this plot exhibits a class imbalanced, but the minority class forms a distinct cluster. While there is some noise from the majority class within the minority class cluster, a KNN classifier with k=5 would be able to handle this noise.

5.3.2 Algorithms Suitable for Imbalanced Problems

Some algorithms perform better on imbalanced data than others. Lemnaru et al. compared various algorithms and found that decision tree algorithms (C4.5) and ANNs both perform better than SVMs on imbalanced data [50].

In general, algorithms that support a tunable loss function which is able to adapt to different class balance ratios are able to handle imbalanced datasets well. Penalizing errors in the minority class has worked well for some applications[51]. For example, imposing a penalty can be implemented by changing weights for different equation components when calculating the loss or error function for a model.

$$Loss = ClassAWeight * ClassALoss + ClassBWeight * ClassBLoss \tag{5.8}$$

Cost sensitive algorithms are configured with a cost matrix that defines the weights for each type of error. Table 5.2 shows a default cost matrix that treats errors in both classes equally, and is suitable for balanced problems.

Table 5.2: **Default Cost Matrix -** The default cost matrix for a balanced problem.

Predicted	**Actual Class A**	**Actual Class B**
Class A	0	1
Class B	1	0

If class B is the minority class, with an imbalance ratio of 1 to 10, then a cost matrix similar to Table 5.3 below might be used instead. However, in general, cost matrices have to be tuned with respect to the specific application. Using the credit card example from earlier in the chapter, the cost matrix might be based on the estimated cost in dollars to investigate a potentially fraudulent charge vs. the average dollar amount of fraudulent charges. Essentially, this is the ratio of real-world cost for one type of error vs. another type of error.

Table 5.3: **Imbalanced Cost Matrix -** The default cost matrix for an imbalanced problem.

Predicted	**Actual Class A**	**Actual Class B**
Class A	0	10
Class B	1	0

5.3.3 Undersampling Majority Class

Undersampling the majority class when creating a training set can help to reduce the class imbalance ratio, but this strategy is not ideal since it does not take advantage of all available data[55]. This strategy is essentially a sensitivity-specificity trade-off. Figure 5.5 shows what undersampling of the data from Figure 5.1 might look like.

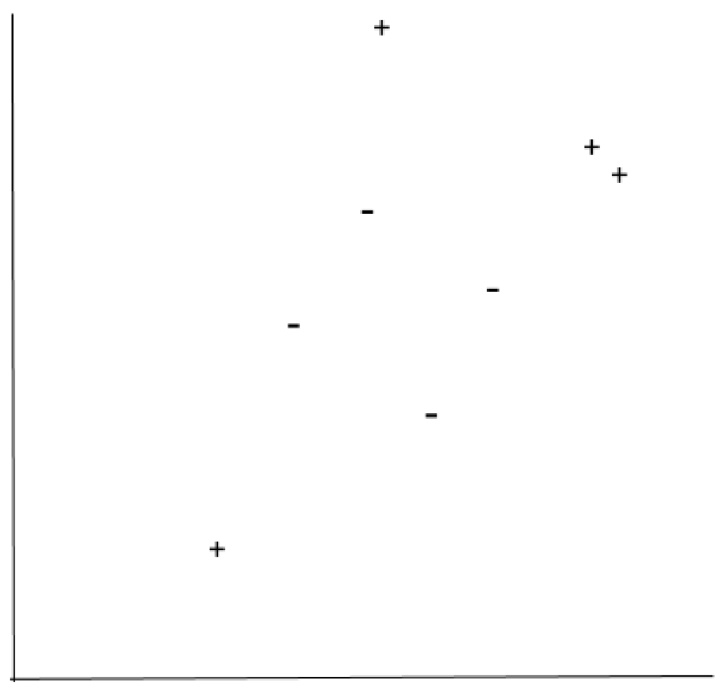

Figure 5.5: **Undersampled Majority Class -** Majority class data points (marked '+') have been undersampled until the dataset is balanced. However, the circle shape of the majority class points is no longer obvious.

When tuning random forests for imbalanced data Chen[56] notes that each tree in the forest can be given a different set of training examples from the majority class. Each tree is given a balanced training set, but a larger number of training examples are used by the forest as a whole. This allows the forest to learn from all available majority class data points while still providing a balanced training set for each tree.

5.3.4 Oversampling Minority Class

In its simplest form, oversampling creates new training points by sampling the minority class with replacement. More advanced oversampling can create additional synthetic data points by sampling feature values from a probability distribution derived from the original training points. In the example below, sampling from one feature is independent of other features.

In the following example, we start with an imbalanced training set with 2 instances of the minority class and 4 instances of the majority class (an imbalance ratio of 1:2). Table 5.4 below shows the original training set.

Table 5.4: **Imbalanced Training Set Before Oversampling -** An imbalanced dataset.

Variable	**Class Label**
4	A
10	B
5	A
9	B
6	A
5	A

Next we apply simple oversampling, by sampling the minority class with replacement. This will bias any models trained on the data set towards the minority class. Table 5.5 below shows the training set that results from simple oversampling.

Table 5.5: **Oversampling with Replacement Dataset -** An imbalanced dataset after simple oversampling. Data points in bold are generated by resampling existing minority data points.

Variable	**Class Label**
4	A
10	B
10	**B**
5	A
9	B
9	**B**
6	A
5	A

For comparison, Table 5.6 contains synthetic data points generated by averaging multiple minority datapoints together.

Table 5.6: **Dataset with Synthetic Data -** An imbalanced dataset after synthetic oversampling. Data points in bold are generated by averaging minority data points together.

Variable	**Class Label**
4	A
10	B
5	A
9	B
9.5	**B**
9.5	**B**
6	A
5	A

One approach to generate synthetic datapoints is SMOTE (Synthetic Minority Over-Sampling Technique). SMOTE generates new points by placing them at a random location between two points of the minority class in the feature space[55]. This method rarely creates synthetic data points outside of existing cluster because it averages a data point with one or more of its nearest neighbors (likely in the same cluster) to generate each synthetic data point.

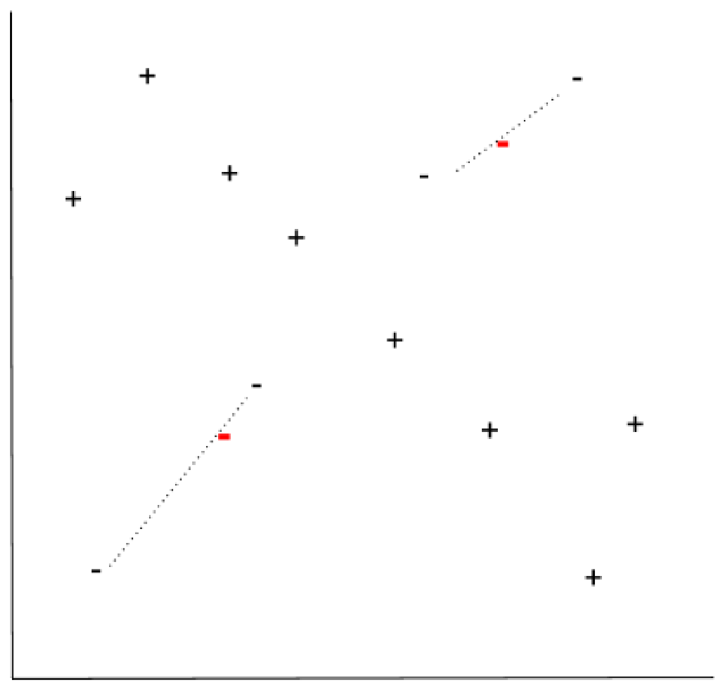

Figure 5.6: **SMOTE Synthetic Sampling Technique -** Two synthetic data points (marked in red) are generated using the SMOTE technique.

However, some of the most successful oversampling methods use creative, application specific ideas to generate more samples of the minority class. One example is in the classification of images or 3D objects, in which the image or object can be rotated slightly to induce jitter[57]. The object may well be found in different rotations in the test data, so this jitter emulates reality.

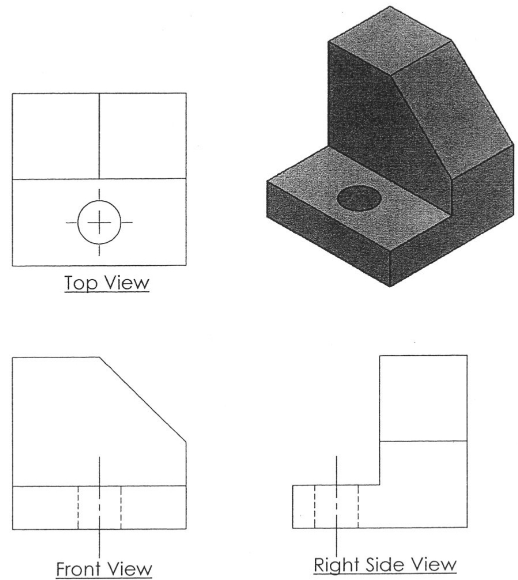

Figure 5.7: **Application Specific Oversampling - Image Rotation -** Both humans and machine learning algorithms can learn more about the shape of this 3D object (Oilfox - used under Creative Commons Attribution-Share Alike 3.0 Unported[58]) by viewing from multiple angles. When viewed from the front or side, it is impossible to tell that there is a hole bored in the part.

Regardless of which oversampling technique is used, Cluster-Based Oversampling (CBO) can be used to oversample the most useful points first[55]. For data sets where the minority class consists of more than one cluster, one cluster may be less dense than the other. This cluster is more challenging to classify, and oversampling points from the denser cluster may not increase model performance. CBO attempts to create clusters with similar numbers of data points. This potentially improves accuracy for minority class clusters with lower density.

6. METHODS

The primary method applied in this paper is supervised learning. The experiment tests various machine learning algorithms to determine if blacklist regions can be predicted based on independent variable suggested by experts. In bioinformatics, and other applications, a variety of statistical methods[59] are used to determine if the problem is tractable, and which algorithm performs the best.

First, data sets are generated by breaking the genome into intervals (which will be called windows below). If a wind overlaps with the expert-generated blacklist then it will be assigned the class label "blacklist". Otherwise it is assigned the class label "normal". The independent variables, or features, corresponding to each class-labeled window are computed. A set of experiments is conducted in which each classifier algorithm (described in Chapter 4) is trained on some of these windows and then tested for performance on a disjoint set of windows. Each experimental condition (a set of dataset sampling parameters, the algorithm, and algorithm parameters) is conducted with multiple times. Performance metrics described in Chapter 5 are used to rank classifier algorithms, and the best algorithm is selected. A workflow diagram for the full process is shown in Figure 6.1 below.

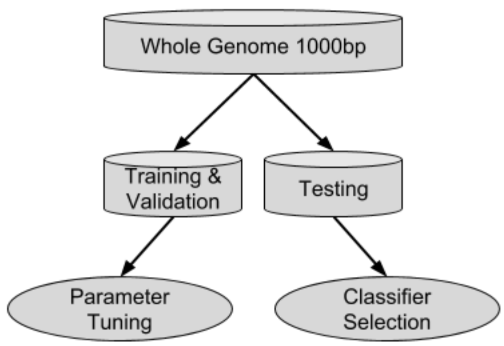

Figure 6.1: **Methods Workflow -** This figure shows a high-level overview of the steps performed to create the supervised learning model. First, the genome is split into training and validation data, which is used for tuning classifier parameters. Candidate models are then tested against a disjoint dataset to estimate their performance.

Next the blacklists predicted by the classifier model are tested on ChIP-Seq data sets. A ChIP-Seq pipeline is executed on original data from ChIP-Seq experiments from human and *Arabidopsis thaliana* samples with and without blacklisting. The ChIP-Seq quality metrics described in Chapter 2 are compared between the treatments. The classifier-predicted blacklist is also compared to the existing Kundaje blacklist.

This chapter explains the features that the classifier uses to predict genomic windows and the scripts that are used to compute features. Then it describes the classifier experiments that are conducted, the scripts used to automate those experiments, and the methods used to sample training and testing data sets for those experiments. Finally, the ChIP-Seq pipeline protocol is described, along with how blacklist regions were selected from the *Arabidopsis thaliana* genome to serve as training and testing data.

6.1 Feature Generation

Initial feature selection was based on the variables that Kundaje considered when creating his hg19 blacklist. Alignability based features were designed based on Kundaje's description of high-mappability and low-mappability islands. Kundaje also uses the ratio of multi-mapping to uniquely-mapping reads with a threshold to determine his blacklist. Repeat based features were included due to the large number of repeat associated regions in both the Duke and Kundaje blacklists. These features include:

- Alignability average
- Number of alignability loci below 0.1
- Ratio of multi-mapping to uniquely-mapping reads
- Number of intersecting RepeatMasker[60] repeats
- Softmask count

The sequence-based features listed below were investigated to see if blacklist regions can be predicted by sequence complexity, rather than just the presence of specific repeat motifs. These features are hopefully more general, and can be applied to organisms without extensive RepeatMasker databases.

- Number of unique 4-mers
- 2-mer tandem repeats
- A/T/C/G Content

Two additional feature types were added based on investigation of UCSC Genome Browser tracks viewed at blacklist loci. Specific focus was placed on quality metrics, genome assembly metadata, and information that is sequence based (rather than from ChIP-Seq).

- Distance to nearest gene
- Distance to nearest assembly gap

Some notation must be defined for working with genomic loci and intervals of genomic loci. The set of genomic loci L differs for each organism and genome assembly. In order to select a window from the genome between loci i and j, a set of loci is defined as $W_{i,j} = [i, j]$. In reality, there are multiple chromosomes within the genome, but in order to keep the notation simple it is possible to think of the loci of additional chromosomes as simply higher indices (with no windows overlapping the chromosome boundary). If a genomic location is within a window, this is indicated by the condition $i \in W_{i,j}$ is true. The function $d(x, y)$ is the distance between two genomic intervals x and y.

The notation $G_{i,j}$ is used to refer to a DNA sequence within the genome from positions i to j (similar to a notation used in *Introduction to Algorithms*, Cormen et al. [61]). The function $\Delta(x, y)$ is the number of base pairs different between two sequences of the same length. For example: the sequences ATA and ATT have one mismatch. The sequences ATA and ACT have two mismatches.

We define the nucleotide alphabet as $N = \{A, T, C, G\}$. Equation 6.1 shows the set of unique sequences of length 3. The set of 3-mers can be alternatively defined using 3rd cross product of the the nucleotide alphabet. Each k-mer is a tuple of nucleotides from the alphabet N.

$$kmers(3) = N \times N \times N = \{(A, A, A), (A, A, T), ..., (G, G, C), (G, G, G)\} \tag{6.1}$$

A general definition of the set of k-mers can be defined as the Kth cross product of the nucleotide alphabet. The easiest way to define this is using a notation described by Sipser in *Introduction to the Theory of Computation*[62]. The Kth cross product of a set A will is notated A^k. From here, the set of all k-mers can be defined as seen in Equation 6.2. Finally, this brings us to the definition of the number of unique k-mers in a genomic window of length l beginning at position w_i.

$$kmers(k) = N^K = \overbrace{N \times N \times ... \times N}^{\text{k}} \tag{6.2}$$

6.1.1 Alignability

This feature measures the number of occurrences of a sequence of length k in the genome with an alignment tool[63]. The alignability score for a position in the genome is calculated as the inverse of the number of times the k-mer starting at that position occurs in the genome. A sequence that is frequent in the genome has a lower alignability.

A sequenced read S maps to a set of genomic loci defined by Equation 6.3. The total number of times a sequence at a position aligns to the genome is defined in Equation 6.4. In simpler terms: $numAlignments(i, k)$ is the number of times the sequence at a genomic location occurs in the entire genome while allowing for n mismatches. For all features below, the maximum sequence distance for alignment n is set to 2.

$$alignments(S, k) = \{l \in L | \Delta(S, G_{l,l+k}) \leq n\} \tag{6.3}$$

$$numAlignments(i, k) = |alignments(G_{i,i+k}, k)| \tag{6.4}$$

The alignability feature at genomic position i with length k and maximum mismatches n is formally defined in Equation 6.5. The feature **Average Alignability** is defined in Equation 6.6, and **Alignability Less Than 0.1** is defined in Equation 6.7.

$$alignability(i, k) = \frac{1}{numAlignments(i, k)} \tag{6.5}$$

$$meanAlign(w_i, l, k) = \frac{\sum_{i=w_i}^{w_i+l} alignability(i, k)}{l} \tag{6.6}$$

$$lowAlign(w_i, l, k) = |\{i \in W_{w_i,w_i+l} | alignability(i, k) < 0.1\}| \tag{6.7}$$

A bedGraph file contains the alignability scores for each offset into the chromosome. If alignability is 1.0 the k-mer at that position only maps to the genome once. If it is 0.5, the k-mer maps to the genome twice. The variable used as a machine learning feature is the average alignability over the region.

For the purposes of creating a blacklist, k is selected to be in the range of sequenced read lengths used for ChIP-Seq experiments. An alignability track is available for the hg19 genome, but a python script is provided to generate the bedGraph file for new genomes and genome versions. The implementation of the alignability algorithm generates a complete list of unique k-mers in the genome, and then uses bowtie to map them back to the genome.

6.1.2 Ratio of Multi-Mapping to Uniquely-Mapping Reads

This feature is defined as the ratio of k-mers in a window that map to the genome more than once over the number of k-mers that map only once. The feature **Multi-Mapping to Uniquely-Mapping Reads** is defined in Equation 6.8 below shows how to compute the multi-mapping ratio for a genomic window starting at position w_i with length l and alignment sequence length k.

$$mmr(w_i, l, k) = \frac{|\{i \in W_{w_i, w_i+l} | numAlignments(i, k) > 1\}|}{|\{i \in W_{w_i, w_i+l} | numAlignments(i, k) = 1\}|} \tag{6.8}$$

This feature is pulled directly from Kunaje's[7] description of his method for creating blacklists. Kundaje uses a simple threshold of 20:1 for this feature when selecting blacklist candidates. This feature is calculated from the same alignability bedGraph file as the feature above. The number of multi-mapping reads (reads that map to more than one location in the genome) and uniquely mapping reads (reads that map at only one location) are counted. The ratio between these two numbers is then calculated as the final feature number for that region.

6.1.3 Intersecting RepeatMasker Repeats

RepeatMasker is a tool that annotates repetitive sequences within a genome assembly[60]. PeakPass uses the number of RepeatMasker repeats that intersect with a genomic window as feature. Repeats are associated with low complexity and mappability. Observation of some blacklist regions in hg19 show overlap with repeats called by RepeatMasker[60].

This feature uses the bedtools *intersect* tool to find all RepeatMasker repeats within the sampled region. The number of repeats for each region is then summed to get the final feature value. A set of RepeatMasker repeats R is a set of intervals within the genome. The **Intersecting Repeats** feature is defined by Equation 6.9. In the PeakPass implementation, an additional requirement that at least 80% of the repeat region must be within the window is imposed.

$$repeats(w_i, l) = |\{r \in R | r \cap W_{w_i, w_i+l}\}| \tag{6.9}$$

Pickrell et al. investigated the impact of repeats on ChIP-Seq false positives in hg18 and hg19[64]. They conjectured that some regions of reference genomes are misassembled. The genomes of individual organisms contain a repeat at this region, but the repeat is truncated in the genome assembly. By investigating the number of copies of certain repeats in raw sequencing data across a large number of individuals, the authors found that sequences from that region appear many more times than expected if they only appeared once in the genome.

In addition to a feature that counts the number of repeats, the types of intersecting repeats can be used to compute various features. The BED file that contains the RepeatMasker[60] repeat locations also includes a repeat type such as SINE or LTR. The mode repeat type for each region can be easily calculated as a nominal value. However, repeat types differ between organisms, and we opted not to include these features.

6.1.4 Soft Mask Count

RepeatMasker can annotate regions of the genome that contain repeats by marking them in lowercase letters[60]. This feature is a count of the number of soft masked base pairs within a genomic window. For example, if a window is 2000bp long, the soft mask count can be between 0 and 2000, where a soft mask count of 0 means that no base pairs are masked, and a soft mask count of 2000 means that all base pairs are masked. **Soft Mask Count** is defined by Equation 6.10.

$$sm(w_i, k, l) = |\{i \in W_{w_i, w_i+l} | i \text{ is softmasked}\}| \tag{6.10}$$

This feature is similar to the number of intersecting repeats, but instead measures the base pair-level amount of repetition. Annotated repeats may be as short as a few base pairs, and as long as thousands of base pairs (e.g. centromeric repeats). This feature can also vary based on how much the repeat interval intersects with the window interval.

6.1.5 Number of Unique K-mers

The number of unique k-mers in a genomic window is defined as the magnitude of the set (without repetition) of DNA sequences of length k found within that window. The number of unique k-mers is defined in Equation 6.11. A large number of unique k-mers relative to the window size indicates a complex region. We use **Unique 4-Mers** as a learning feature.

$$uk(w_i, k, l) = |\{p \in kmers(k) | p{=}G_{i,i+k} \wedge w_i{\leq}(i{+}k){\leq}l\}| \tag{6.11}$$

As an example, the sequence ATATCTATG contains the 3-mers ATA, TAT, ATC, TCT, CTA, ATG (resulting in a unique 3-mer count of 6) and the 4-mers ATAT, TATC, ATCT, TCTA, CTAT, TATG (resulting in a unique 4-mer count of 6). Regions with low complexity contain lower numbers of unique k-mers. Using Equation 6.12, we show that the maximum number of 4-mers is 256 (assuming that the sequence length is greater than or equal to 256). This means that the minimum recommended window size is 256, as a smaller window will have compressed values for this feature.

$$MaxUniqueKmers(K, SeqLength) = \begin{cases} K^4, & \text{if } SeqLength \geq K^4 \\ SeqLength, & \text{otherwise} \end{cases} \tag{6.12}$$

6.1.6 Tandem Repeats

Tandem repeats are sequences that are immediately repeated. For instance, the sequence ATATAT contains three tandem repeats of the sequence AT. The number of tandem repeats(Equation 6.13) is the number of times a sequence of length k is repeated within the window. The number of **2-Mer Repeats** are measured as a feature.

$$kr(w_i, k, l) = |\{i \in W_{w_i, w_i+l} | G_{i,i+k}{=}G_{i+k,i+2k} \wedge (i{+}2k){<}l\}| \tag{6.13}$$

Similar to the unique k-mers feature, this feature measures region complexity. If a k-mer appears multiple times within a window, but is spread out in the window, the number of k-mer repeats will be low. Combined, unique k-mers and repeated k-mers provide a better measure of region complexity than either feature alone.

The following sequence is an example of a Telomeric Repeat from a Kundaje hg19 blacklist region with both high monomer and two-mer repeats:

```
ggttagggttagggttagggttagggttagggttagggttgggttagggt
tggggttggggttggggttggggttggggttgggttagggttagggttag
```

Tandem repeats are very common in the human genome, and are they are prone to mutation[65]. While most individuals have a small number of these repeats in a particular region of the genome, some individuals may have mutations that increase the number of repeats by orders of magnitude[66]. This type of mutation may be associated with a disease. In these cases, the reference genome does not accurately reflect the individual from which a sample is collected.

6.1.7 A/T/C/G Content

PeakPass uses the percentages of A, T, C, and G nucleotides within the window being classified as machine learning features. Equation 6.14 defines the percentage of the whole window that is that nucleotide. Each nucleotide content feature simply performs this function with a different nucleotide. We use **A Content**, **T Content**, **C Content**, and **G Content** as features for our model.

$$nContent(w_i, l, n) = \frac{|\{i \in W_{w_i,w_i+l} | G_{i,i+1}{=}n\}|}{l} \times 100 \tag{6.14}$$

Low complexity regions show bias in one or more nucleotide percentages. Repeated motifs may cause a nucleotide bias as well. Repeats and low complexity regions result in low alignability, which is measured in a separate feature above. Besides alignability, CG content bias, or the percentage of cytosine plus guanine base pairs in the region, has been found to create a bias in sequencing[30, 11, 67]. Sequencing bias can be exhibited as faster sequencing for some sequences than others, sequencing more reads with particular sequences, or miscalled nucleotides.

6.1.8 Distance from Nearest Gene

The distance of a window to the nearest gene is calculated as the number of base pairs between that window and the closest gene (either upstream or downstream) according to a gene database for the genome assembly version used. If the blacklisted region is within the gene the value of this feature is 0. The set of gene intervals G is used to formally define the feature **Distance from Nearest Gene** in Equation 6.15.

$$nearestGene(w_i, l) = min\{d | d{=}d(g \in G, W_{w_i,w_i+l})\} \tag{6.15}$$

The distance to nearest gene is calculated using the bedtools *closest* utility[68]. While there may be more than one gene database for a particular organism, the main criteria for selecting a gene database is that it is based on a particular genome assembly version. Otherwise, the gene loci may be incorrect.

Kundaje[7] manually removes some items that would otherwise be blacklisted due to proximity to known genes or transcription start sites. Jain et al. found that some active promoters cause false positives in ChIP-Seq experiments[69]. This was found during literature review, and may explain blacklist items that are close to genes.

6.1.9 Distance from Nearest Assembly Gap

Genome assemblies are created using what NCBI calls the "Hierarchical approach"[70]. First, the genome is fragmented into large fragments which are cloned into bacterial chromosomes (BACs) in order to create a large number of copies. These BACs are then fragmented and sequenced. Individual reads are combined together at overlapping regions to determine the sequence of the BAC. Next, the fragments from the BACs are combined together to form the whole genome. If the sequenced reads do not cover the whole BAC fragment then a gap with known approximate length, but unknown sequence results.

The distance to nearest assembly gap feature measures the number of base pairs between the genomic window being classified and the closest assembly gap. G is the set of genomic gaps, which are intervals (similar to the set of gene intervals). **Distance from Nearest Assembly Gap** is defined using G in Equation 6.16.

$$nearestGap(w_i, l) = min\{d | d{=}d(g \in G, W_{w_i,w_i+l})\} \tag{6.16}$$

These gaps are often hard masked (marked by the character "N") in the genome Fasta files. Duke and Kundaje blacklist items often cluster around assembly gaps, as seen in Figure 6.2. No peaks can occur within a gap because no reads are mapped to the hard masked region, however nearby regions may be responsible for false positive ChIP-Seq results. The reason assembly gaps are associated with blacklist items is unknown, and there may not be a causal relationship, but the association was commonly observed in genome browsers.

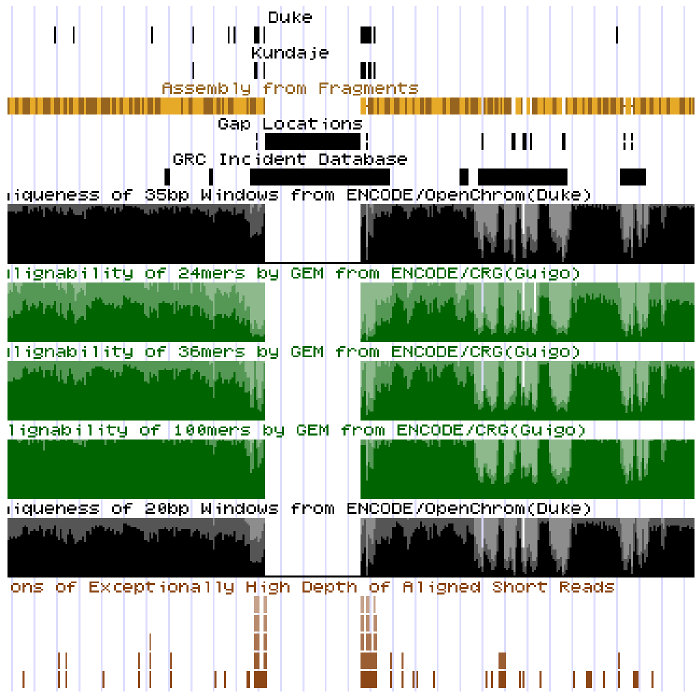

Figure 6.2: **Assembly Gap Visualization -** This image shows an assembly gap viewed in the UCSC Genome Browser [9]. Both the Kundaje and Duke blacklists have items in the region surrounding the gap, most prominently adjacent to the gap.

Iafrate et al. observed that a disproportionate number of large-scale copy-number variations LCVs appear near assembly gaps[71]. Highly repetitive regions are more difficult to assemble, and may require longer reads, more reads, paired-end reads, and other more expensive sequencing techniques to determine for an individual. Ultimately, how to handle these variations for an assembly depends on the assembly application. Another possible explanation for the association between assembly gaps and ChIP-Seq false positives is that assembly gaps result in mappability artifacts, as no reads can be mapped in that region.

The authors of the DangerTrack tool[29] observed that their algorithm tends to assign a higher score to regions that overlap with GRC incidents, many of which are assembly gaps. It is possible that whatever is causing the genome assembly gap (low complexity, repetition, etc.) is also causing the false positive ChIP results, and not that the gap itself is causing mappability issues in the ChIP-Seq bioinformatics pipeline.

Other gap related features include gap type and gap size. Common gap types are contig, telomere, centromere, short_arm, heterochromatin, and clone. Unfortunately, there is no standard way of reporting gap type across species. Many gaps in the mm10 genome are simply labeled as “other”. Both gap type and size estimation take manual work beyond simple genome assembly, and are therefore not available for all organisms.

6.2 Dataset Generation

Feature computation is performed by dividing the whole genome is tiled into windows of the same length. A bash script in combination with bedtools[68] generates these tiles and then gathers and formats the features for each tile as a vector. A unique id is generated for each data point by concatenating the chromosome, start, and stop offset. These features are merged together with the unique id and class label in a single CSV file. Finally, random samples of the whole-genome dataset are created as training/testing dataset pairs by another script.

6.2.1 Genome Windowing

The genome is split into an equal number of windows based on a parameterized window size. Figure 6.3 below shows how windows that sufficiently overlap a blacklist region are labeled “blacklist”, and other regions are labeled “normal”. Features are gathered on a per-window basis, and these windows become data points in the dataset used for machine learning.

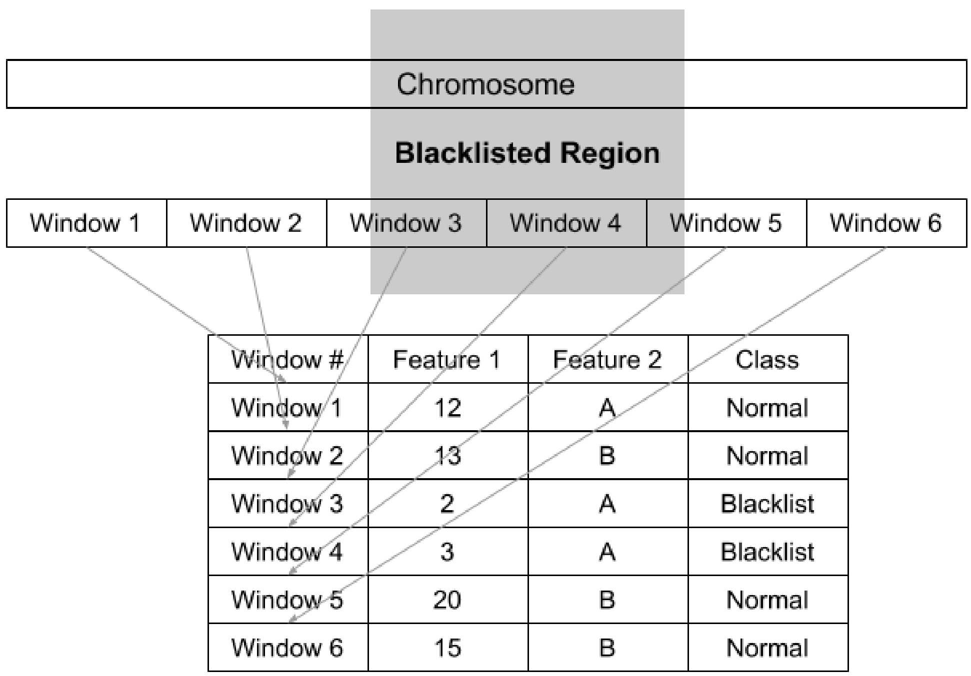

Figure 6.3: **Chromosome Windowing -** Each chromosome is split into windows. If a window overlaps significantly with a blacklist region, then it is assigned the blacklist label. Each window becomes a row of the dataset, with computed features and a class label.

While there are a large number of genomic windows, feature gathering for is an embarrassingly parallel problem, and the script is designed to take advantage of this. A configuration file contains a list of chromosome names and lengths, which is loaded into an array. All features are gathered by looping over this array and spawning a process to gather that feature on a single chromosome. The outputs are then merged into a single file and all temporary files are deleted. The xargs “–max-procs=” parameter is used to configure the number of processes executing in parallel so that the script can be tuned to different hardware. Script development and testing was performed on a machine with 24 cores and 96Gb of memory. Feature gathering for the human genome typically takes less than 10 minutes. Additional filtering steps can be applied at this stage, for example: removing data points where features could not be calculated (e.g. completely hard-masked regions).

Table 6.1: **Example Dataset -** A few examples of training data. Features include softmask count, unique 4-mers, 2-mer repeats, various alignability measurements, and distances to genomic features such as genes and gaps. Some features have been removed for clarity.

Mask	**4mers**	**2mer**	**Avg**	**Low**	**High**	**Ratio**	**GapDist**	**GeneDist**	**Repeat**	**Class**
496	176	44	0.0	0	0	500	3212958	238580	2	N
500	183	63	0.0	0	0	500	385195	289765	2	B
315	199	48	0.40	132	103	2.0	48	38586	3	N
500	180	31	0.0	0	0	500	12218916	74293	3	N
500	97	16	0.0	0	0	500	42065	17562	3	B

6.2.2 Oversampling

As discussed in Chapter 5, some applications allow for creative oversampling of the minority class when creating training data. We propose a method for oversampling blacklisted items by overlapping the windows used to gather feature data about the genome, as seen in Figure 6.4. This method results in n additional training samples for the minority class (where n is the number of existing windows) for every shift. The first set of overlapping windows is shifted by 50% of the window length, the second by 25%, and so on. However, if the shift distance is a small percentage of the window size it will result in extremely similar training examples, so it may only be prudent to do this a few times.

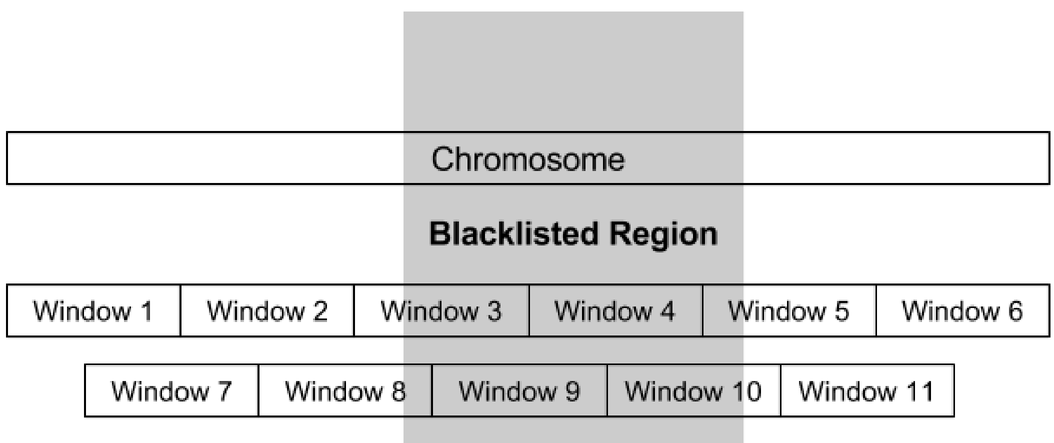

Figure 6.4: **Sliding Genomic Windows -** Additional training data can be gathered by sliding the windows used to calculate features by an offset less than the window size.

Oversampling regions by using multiple sets of windows at offset to each other increases the number of samples for training and testing. If oversampled windows are used for prediction, the resolution of predictions can be increased. For blacklist prediction, prediction resolution is already adequate, but for other genomic region classification problems overlapping prediction windows may be useful. This technique is commonly used in sound and speech classification[72, 73]. Overlapping windows of time are sampled out of frequency data to improve classification applications such as word recognition.

Overlapping windows can also be used at blacklist prediction time by gathering feature data on unlabeled genomic regions, predicting all windows, and averaging together the predictions of overlapping windows. This has the benefit of increasing prediction resolution, and confidence in a prediction goes up if many overlapping windows in the region agree on the predicted label.

6.2.3 Training and Testing Set Generation

A script was created to sample training and testing data from a whole-genome dataset to automate model testing tasks. The number of blacklist and normal data points for both the training and testing set are passed in as parameters to this script. Training and testing sets are disjoint in terms of windows of the genome, but they are also disjoint in terms of blacklist regions.

1. Randomly split the blacklist into two equal parts
2. Intersect the two blacklist parts with the windowed genomic data
3. Compute the non-blacklist window set
4. Randomly split the non-blacklist window set into two equal parts
5. Sample the input number of blacklist windows and normal windows from one blacklist/normal data pair to create the training set
6. Sample the input number of blacklist windows and normal windows from the disjoint blacklist/normal data pair to create the test set

6.3 Model Tuning and Validation

6.3.1 Feature Selection

Feature selection is the process of determining which features to use in a model. While exploring an initial list of features, it is likely that some poorly predictive variables will be investigated. These features increase model complexity, but not increase model performance. This section describes what features were removed and how the feature selection process selected the final set of features.

We removed some underperforming features, and features that would not work well in practice. The nearest gap size, nearest gap type, and mode repeat type features were unavailable for some organisms and genome assemblies, and therefore removed. If a model trained on an old genome assembly is used on a new genome assembly, these features might not contain the same set of values as the new genome assembly. Loci with alignment over 0.9, unique 3-mers, and monomer tandem repeat features were removed due to high correlation with other features.

First, manual feature selection was performed. R scripts were created to plot feature density and clustering for manual feature inspection, such as the scatter plot shown in Figure 6.5. Visualizing features helps to debug feature calculation and select useful features. If there is an obvious threshold at which class labels split (even with some overlap), then the feature likely provides information useful for classification. Correlations, and exceptions to correlated features can be quickly determined.

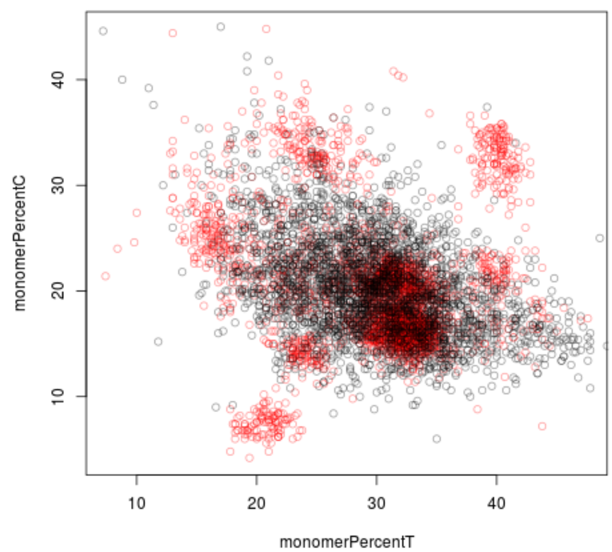

Figure 6.5: **Nucleotide Percentage Scatter Plot -** A feature plot with 'percent C' on the y-axis and 'percent T' on the x-axis. Red circles are blacklist regions and black circles are normal regions. The blacklist data points show several distinct clusters.

There are several methods for automated feature selection. Filtering methods start with a full set of features and remove poorly performing variables[74]. Correlation based feature removal uses the covariance of feature pairs to remove features above a certain covariance threshold[74]. Features that are highly correlated are unlikely to add to model performance, but increase model complexity. In addition, random forest provides a way to measure the importance of individual features. Importance measure is the difference in average accuracy across trees trained with that feature and without that feature.

Highly correlated features may have a detrimental effect on classifier performance, and can cause both features to seem less important in out-of-bag estimates (because one of the two features may be present in a tree used to estimate the other feature's importance). Removing highly correlated features, or combining them with a dimensionality reduction technique, can reveal the true importance of a feature.

An R script was created to calculate the pair-wise Pearson correlation coefficients for all features shown in Table 6.2. Any feature pair with a correlation coefficient of above 0.3 is output by the script. The script is given a balanced training set, and then the output is sorted from highest correlation to lowest. Before running the script, an upper threshold of 0.5 was selected as a cutoff for removing features.

Table 6.2: **Highly Correlated Features -** Correlation coefficients for highly correlated features (correlation coefficient > 0.3). The table is sorted in decreasing order.

Feature A	Feature B	Correlation Coefficient
alignabilityAboveUpperThresh	alignabilityAvg	0.94
uniqueKmers4	uniqueKmers3	0.85
twomerRepeats	monomerRepeats	0.56
geneDistance	gapSize	0.47
geneDistance	softmaskCount	0.39
intersectingRepeats	softmaskCount	0.36
gapSize	softmaskCount	0.33
twomerRepeats	uniqueKmers3	0.33
twomerRepeats	uniqueKmers4	0.33

Table 6.3 below contains feature importance as estimated by random forest. The 'Blacklist Accuracy' column shows the mean decrease in sensitivity for trees in the forest that were not trained on that feature compared to trees that do contain that feature. 'Normal Accuracy' is the mean decrease in specificity for trees that lack that feature. Finally, 'Decrease Accuracy' is the overall decrease in accuracy for all out-of-bag data points. Random forest measures these values using training samples that were not sub-sampled for training that particular tree (out-of-bag).

Table 6.3: **Hg19 Kundaje Feature Importance -** Feature importance for predicting blacklist items as generated by the randomForest R library sorted from highest decrease in accuracy to lowest. Random forest also calculates the out-of-bag decrease in positive class accuracy (sensitivity) and negative class accuracy (specificity).

Feature	Blacklist Acc	Normal Acc	Decrease Acc
gapDistance	60.50	60.77	68.8
softmaskCount	38.65	27.56	40.8
geneDistance	31.76	26.10	36.8
uniqueKmers4	21.86	25.95	30.8
intersectingRepeats	18.23	16.22	21.0
twomerRepeats	12.52	13.26	16.8
monomerPercentT	12.14	11.71	16.0
alignabilityBelowLowerThresh	11.29	12.70	15.8
monomerPercentA	11.61	12.28	15.0
monomerPercentG	10.97	9.53	14.0
monomerPercentC	8.85	12.06	13.8
alignabilityAvg	8.83	10.88	13.6
alignabilityMappingRatio	7.06	7.64	9.9

6.3.2 Parameter Tuning

The caret package[75] in R allows for semi-automated tuning of learning algorithms. It takes a set of tuning parameters and performs a grid search on all combinations of the parameters. Each parameter combination is evaluated using 10-fold cross validation on a single training dataset. Several performance metrics were measured, and a table of parameters and their respective performance metrics was reported. Table 6.4 below is an example of this output for the random forest algorithm.

Table 6.4: **Random Forest Tuning Parameters -** Tuning parameters for random forest sorted by ROC AUC from highest to lowest. 'Mtry' is the number of variables sampled when creating a split[44]. The caret tuning API does not currently implement a tuning parameter for the number of trees in the random forest.

Mtry	ROC AUC	Sensitivity	Specificity
4	0.9959	0.9507	0.9805
6	0.9956	0.9597	0.9787
2	0.9952	0.9412	0.9850
8	0.9952	0.9640	0.9763

This process was performed on all classifiers in order allow for comparison at their predicted best settings. Tuning parameters with the best ROC AUC were selected for model validation. Table 6.5 below contains the optimal parameters for each algorithm. Tables with performance data for additional parameter values for each algorithm are available in the appendix.

Table 6.5: **Optimal Parameters -** This table contains a complete list of all tuned parameters for all algorithms and their optimal values.

Algorithm	Hyperparameter	Optimal Value
randomForest	Mtry	4
ANN	Hidden Layer Size	9
ANN	Decay	0.5
SVM (radial)	Cost	2.0
SVM (radial)	Sigma	0.2
SVM (linear)	Cost	0.5
KNN	K	15
KNN	Kernel	Triangular
KNN	Distance	0.5

6.3.3 Classifier Selection and Testing

In this section, we describe how we characterize and compare classifiers in difference scenarios. First, each algorithm is tested with all combinations of balanced and imbalanced training and testing sets. Balanced datasets contain an equal number of blacklist and normal regions, and imbalanced datasets have an imbalance ratio of 1 blacklist item for every 300 normal items. Performance metrics are then graphed for easy comparison, and presented in Section 7.2. Random forest had the best performance when considering both AUC ROC and AUC precision/recall, and exhibited low variance for these performance metrics, therefore we select random forest as our final classifier.

To determine the impact of training class imbalance random forest, we tested with balanced testing data and various levels of imbalanced training data. Figure 7.10 shows that AUC ROC and AUC precision/recall increase as imbalance ratio improves. Next, performance with different training set sizes was tested. Balanced training sets of from 50 to 6,400 data points were created. A random forest classifier was trained with each training set and then evaluated with a balanced test set. The results in Figure 7.11 show that AUC ROC and AUC precision/recall improve with increased sample size, but have diminishing returns.

Another experiment was conducted to determine the impact of window size on model performance.

Training and testing datasets are created using window sizes of 250bp, 500bp, 1000bp, and 2000bp. Random forest was tested for each dataset. Results are graphed in Figure 7.12 Performance is in general similar across the different window sizes, but it appears that there may be a slight resolution / precision trade-off.

Finally, to test the impact of the proposed overlapping window oversampling method, training sets with an artificially limited number of minority class samples were created. SMOTE is used to generate different numbers of synthetic data points from the minority class, and our overlapping window oversampling method samples additional data from overlapping genomic regions. Random forest is the tested with a balanced test set against the original and oversampling training sets. Figure 7.13 shows the results of this experiment. Both SMOTE and our novel oversampling method increase classifier performance as the number of oversampled points increases.

6.3.4 ENCODE ChIP-Seq Workflow

The following workflow is used to call peaks from a BED file containing aligned reads [76]. This workflow implements the recommended ENCODE ChIP-Seq analysis pipeline. We use datasets with two biological replicates and a control, however the workflow can be generalized to more biological replicates. To determine the effectiveness of a blacklist, we run this workflow on a ChIP-Seq data set before and after treatment with the blacklist and compare the results.

1. Compare biological replicates against the control and compute strand-shifted cross-correlation.
2. Call peaks for each replicate, then compare the peak sets between replicates.
3. Create pseudo-replicates.
4. Compare the pseudo-replicates against the control and compute cross-correlation.
5. Call peaks for each pseudo-replicate, and compare the peak sets.
6. Select the final peak set.

7. RESULTS

This chapter contains a summary of experimental results. First, features densities are plotted with graphs color coded by class label. We use a 2D scatter plot to show how important features correlate. Performance data for different learning algorithms is compared. Model parameters such as training set class balance, training set size, and genome window size are measured and plotted to determine the optimal model.

Next, our novel oversampling approach is tested against SMOTE. To test the application of our algorithm, a predicted hg19 blacklist is applied to a ChIP-Seq dataset from ENCODE that exhibits some SNR quality problems. The output of the ENCODE pipeline on both the filtered and unfiltered reads is compared, as well as strand cross-correlation graphs. We also created a novel blacklist for the *Arabidopsis thaliana* genome and performed the same comparison of filtered and unfiltered ENCODE pipeline output.

7.1 Dataset Description

Figures in the section below help visualize individual features, as well as combinations of features. Useful features provide information about the difference between class labels. If the probability distribution for different class labels are extremely similar for a feature, that feature may not have any predictive power. However, some amount of overlap is common. If a feature has no overlap, it may indicate a problem where a simple threshold is sufficient to classify the points.

Figure 7.1 shows probability distributions for alignability based features. Alignability is 1 divided by the number of times a sequence maps to the genome. For more information, see Section 6.1.1. The overall picture is that blacklist regions tend towards extremes of alignability. Blacklist regions have higher average alignability and more loci with alignability below 0.1 (high repetition). There are a large number of both blacklist and normal regions that do not contain a mappable position.

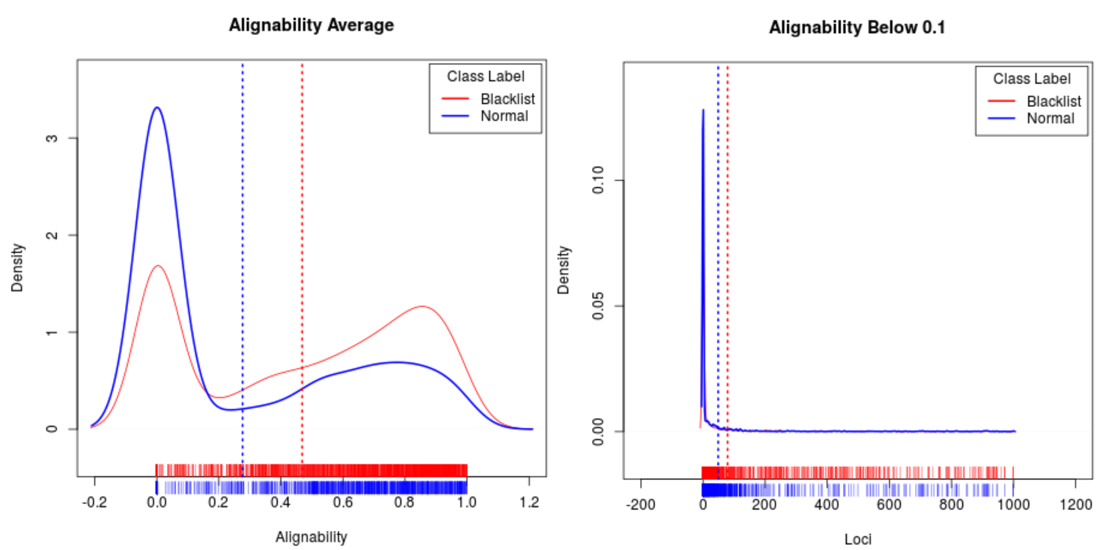

Figure 7.1: **Alignability Feature Densities - Left:** This graph shows an average of the alignability across the whole region (1.0 is uniquely mapping, lower numbers mean there are copies of the kmer at this location). **Right:** This graph shows the number of loci in each region with a mappability below a threshold of 0.1 (many copies of this kmer).

The multi-mapping to uniquely mapping feature (the ratio of non-unique to unique mapping sequences in the region - see Section 6.1.2) plotted in Figure 7.2 indicates that both blacklist and normal regions tend towards extremes. Regions tend to either have mostly unique or mostly non-uniquely mapping loci. Blacklist regions tend to have all uniquely mapping loci, but there are plenty of normal regions with this quality as well. Regions with no mappable positions will not result in false positives because no reads can be mapped to these regions, however they may be close to a blacklist region.

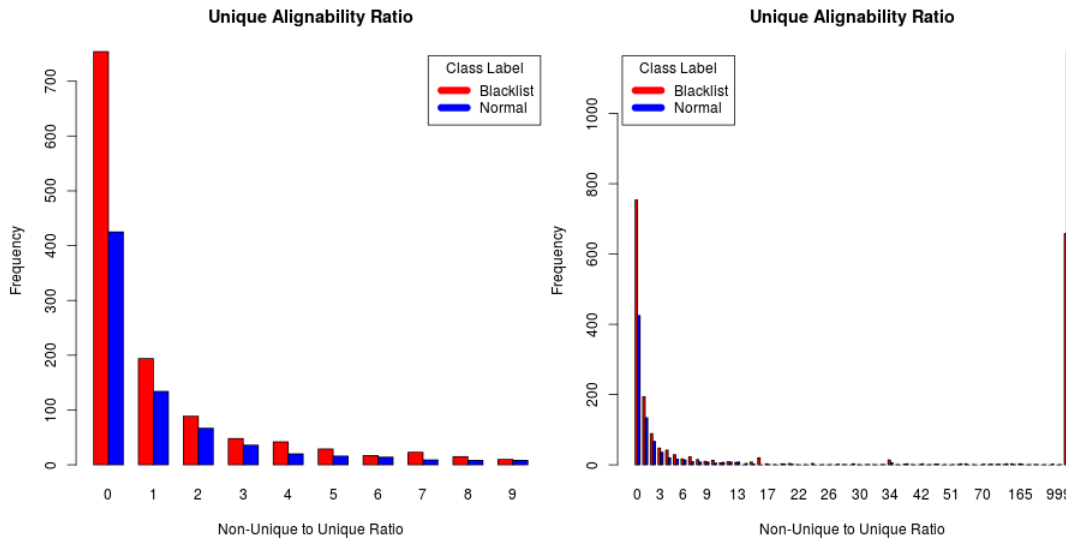

Figure 7.2: **Multi-Mapping to Uniquely-Mapping Ratio Histogram - Left:** A frequency bar plot of windows with particular multi-mapping to uniquely-mapping ratios (total number of data points is 2000). This plot is zoomed on ratios from 0 to 9 where 0 indicates no multi-mapping loci. **Right:** The full zoomed out bar plot shows few data points between a ratio of 20 and 999, and then a large frequency of windows with a ratio of 1000.

The features plotted in Figure 7.3 below compare how repetitive blacklist regions are compared to normal regions. The features are based on common repeats as identified by RepeatMasker[60]. More information can be found about RepeatMasker and soft masking in Section 6.1.3 and Section 6.1.4. This data indicates that blacklist regions contain fewer types of repeats, but also larger repeats. Note that in order to count as an intersecting repeat 80% of the repeat must be within the window. Due to the large number of softmasked base pairs in blacklist regions, we suspect that the large number of blacklist regions with 0 repeats is actually an artifact of this requirement, and that these regions do contain at least one repeat. An improved feature might relax this requirement for long repeats.

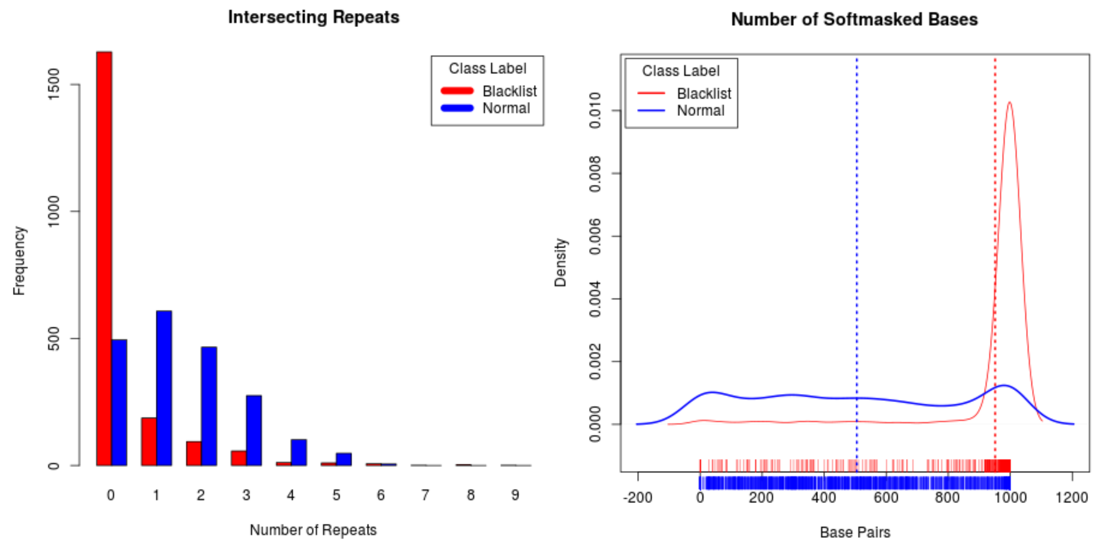

Figure 7.3: **Repeat Feature Densities - Left:** The number of RepeatMasker repeats that intersect with the region. **Right:** The number of base pairs in the region softmasked by RepeatMasker.

Figure 7.4 shows that blacklist regions contain fewer unique 4-mers (described in Section 6.1.5) on average. Furthermore, the number of 4-mers seems to be bimodal for blacklist regions. These regions will contain less complex sequences for at least part of the window. There is also a cluster of blacklist regions with fewer tandem repeats than normal regions, which may be due to a repeat longer than 2bp. Tandem repeats are explained in more detail in Section 6.1.6.

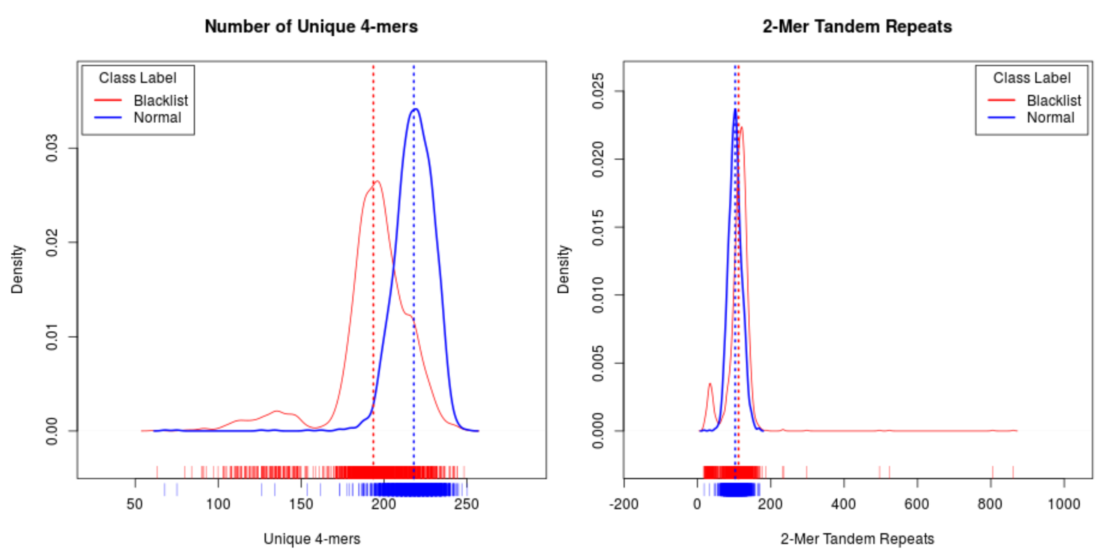

Figure 7.4: **Complexity Feature Densities - Left:** The number of unique 4-mers in the window (e.g. ATCT, ATGA). **Right:** The number of 2-mer tandem repeats.

The features in Figure 7.5 below compare the percentage of A, T, C, and G base pairs in normal and blacklist regions. Low complexity sequences can be biased towards a particular nucleic acid. There are some peculiar differences between the A/T/C/G content (Section 6.1.7) of blacklist windows when compared to normal windows. For one, the nucleotide percentages in blacklist regions do not appear to be normally distributed. Nucleotide percentages for blacklist regions have multiple peaks, and the main peaks have a much smaller variance than normal regions.

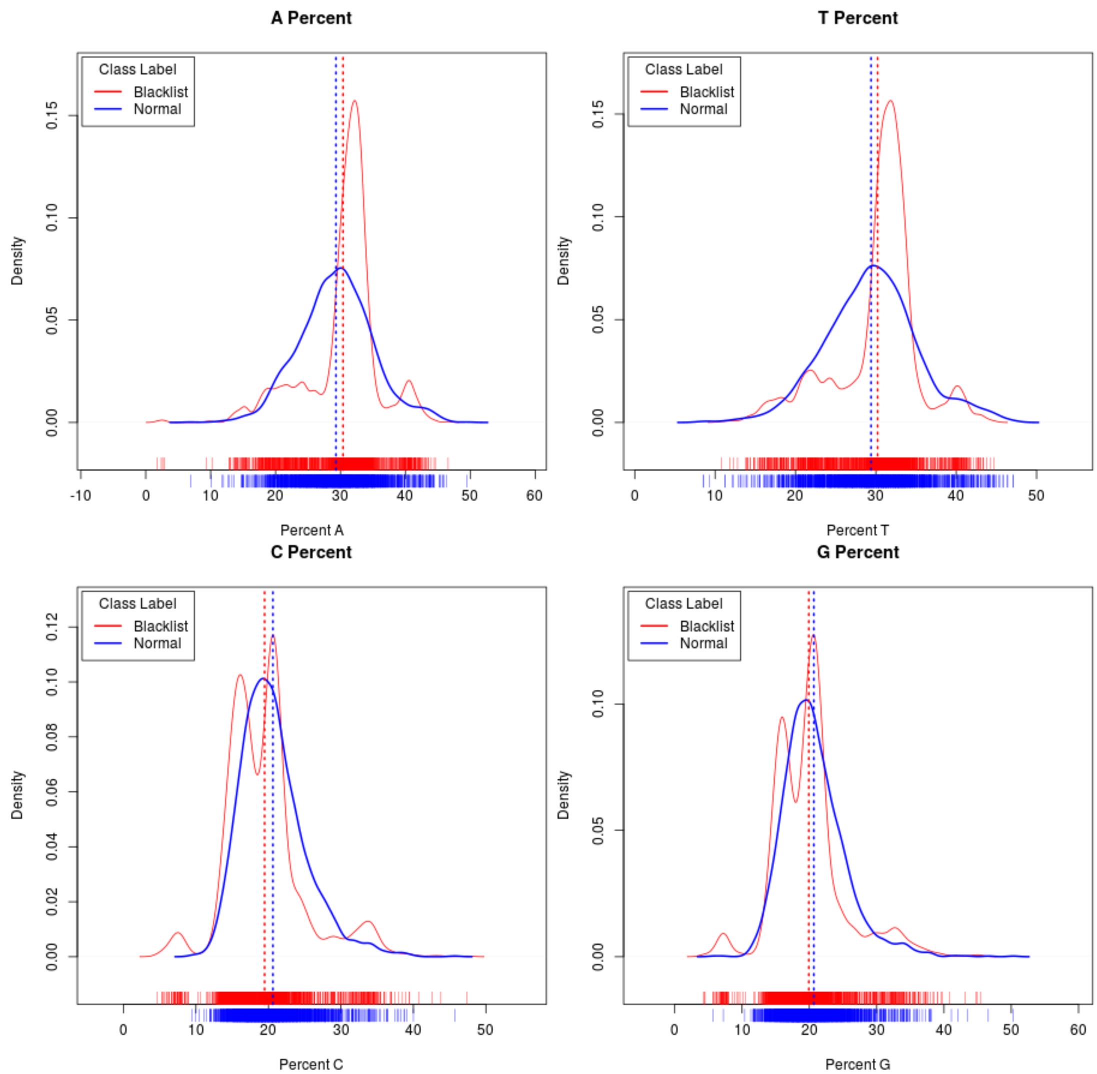

Figure 7.5: **Nucleic Acid Percent Feature Densities - Top Left:** Percentage of adenine base pairs in the window. **Top Right:** Percentage of thymine base pairs in the window. **Bottom Left:** Percentage of cytosine base pairs in the window. **Bottom Right:** Percentage of guanine base pairs in the window.

The features shown in the probability density plots in Figure 7.6 are based on nearby major features of the genome such as genes and assembly gaps. Distance to nearest assembly gene is discussed in Section 6.1.8 and distance to nearest gap is discussed in Section 6.1.9. Problem areas may cluster around certain genomic features, so while these features alone are not sufficient to identify problem regions they provide the learning algorithm with a base probability of whether or not a region is problematic. Blacklist items occur with higher frequency near genome assembly gaps, but have a larger average distance from genes than normal regions.

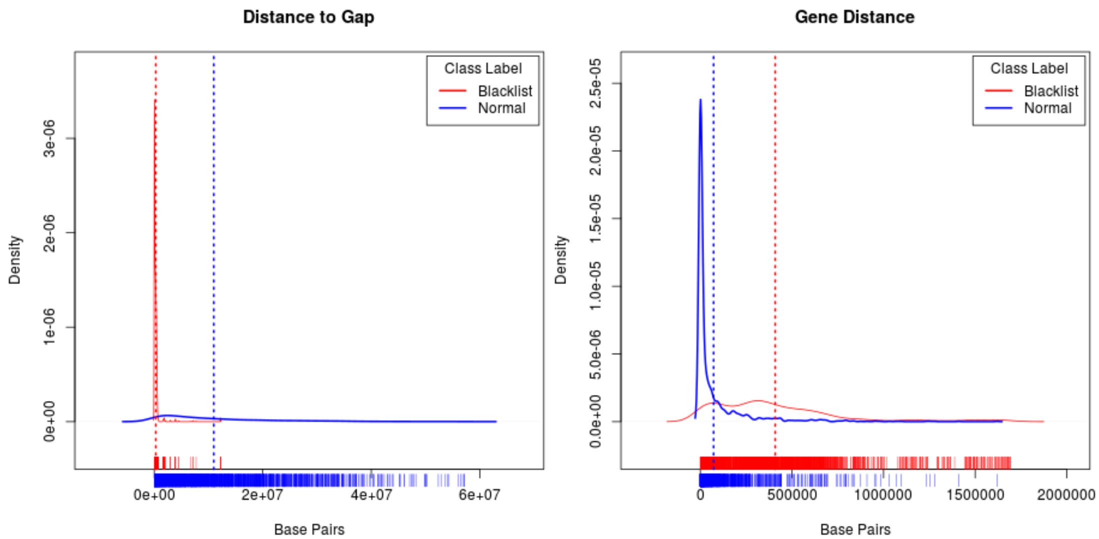

Figure 7.6: **Major Genomic Feature Distance Densities - Left:** Distance to the nearest assembly gap in base pairs. **Right:** The distance to the nearest gene in base pairs. The average for each class label is shown as a dotted vertical line.

Some important features, as determined by the random forest importance metric, were selected to visualize on a scatterplot matrix, seen in Figure 7.7. A 2D scatterplot can reveal correlations between variables, as well as clusters of data points within a class. The feature pairs {unique 4-mers, gene distance}, {gene distance, monomerPercentG}, and {unique 4-mers, monomerPercentG} appear to have such clusters.

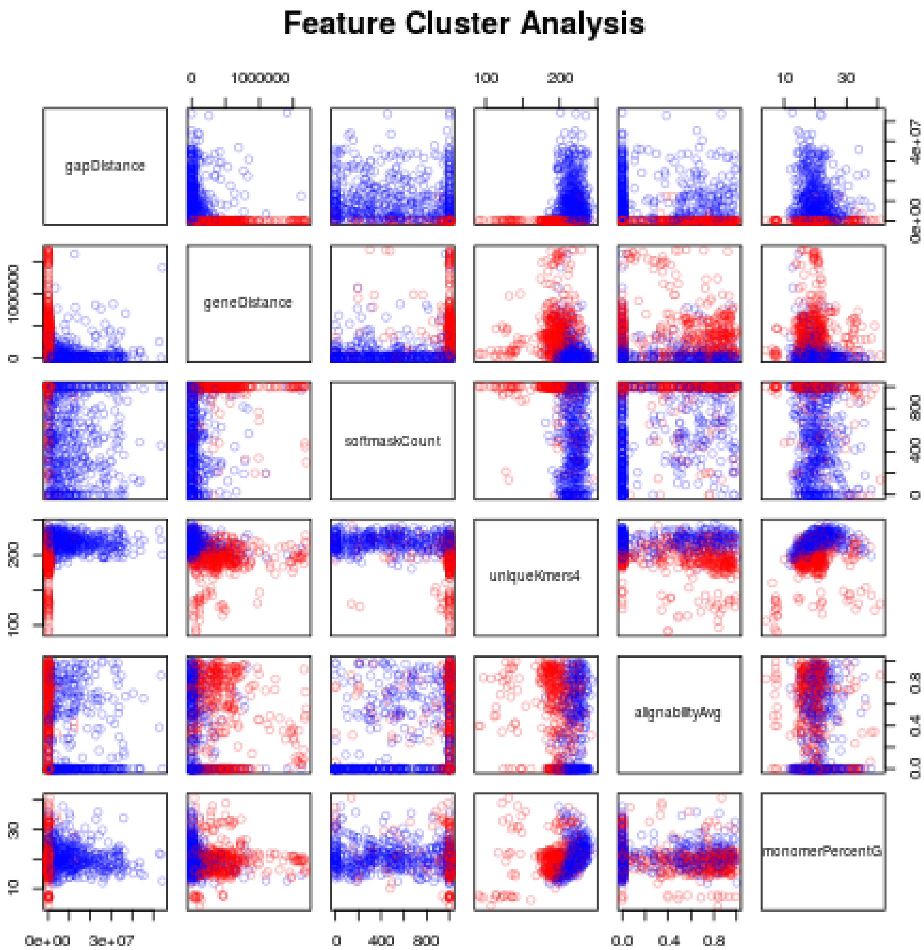

Figure 7.7: **Feature Scatterplot Matrix -** This matrix of 2D scatter plots shows the most important features according to the random forest importance estimate. Second order clusters are easier to visualize with a matrix of scatterplots over several features. Blacklist points are red, and normal points are blue.

7.2 Model Validation and Testing

Table 7.1 shows a performance comparison of learning algorithms described in Chapter 4 on 32 randomly sampled training and testing datasets. The data in this section can be used to determine if machine learning on blacklist data is viable, and which algorithm results in the best model. Each algorithm is tested with all combinations of balanced an imbalanced training and testing datasets. Balanced datasets contain 2000 blacklist and 2000 normal data points, whereas imbalanced datasets contain 56 blacklist and 16,000 normal data points.

Table 7.1: **Hg19 Kundaje Class-Balanced Performance -** Model performance was averaged across the 32 datasets to compare different algorithms. The average of common performance metrics for each algorithm is shown, along with the standard deviation of that performance metric in parentheses.

Algorithm	**Sensitivity**	**Specificity**	**Precision**	**F-Measure**	**AUC Prec/Recall**	**AUCROC**
randomForest	0.930(0.033)	0.978(0.005)	0.976(0.005)	0.952(0.016)	0.990(0.003)	0.990(0.003)
svmLinear	0.935(0.031)	0.959(0.007)	0.957(0.007)	0.946(0.016)	0.980(0.007)	0.984(0.005)
svmRadial	0.825(0.089)	0.968(0.008)	0.962(0.010)	0.886(0.054)	0.975(0.011)	0.975(0.010)
knn	0.898(0.037)	0.951(0.008)	0.948(0.008)	0.922(0.021)	0.975(0.009)	0.969(0.013)
ann	0.899(0.083)	0.950(0.024)	0.948(0.023)	0.920(0.050)	0.972(0.008)	0.966(0.011)
rpart	0.921(0.038)	0.961(0.014)	0.960(0.013)	0.940(0.019)	0.959(0.041)	0.959(0.018)
naiveBayes	0.923(0.037)	0.901(0.057)	0.905(0.047)	0.913(0.033)	0.933(0.035)	0.954(0.026)

The box plot shown in Figure 7.8 shows the range of AUC ROC performance for each algorithm. Both random forest and SVM linear have high AUC ROC and low AUC ROC variance for balanced and imbalanced testing sets. Random forest appears to be slightly better, but based on AUC ROC alone it is difficult to separate these two algorithms. Holm's post hoc test was applied to this AUC ROC data, but the difference between random forest and SVM linear was not significant. All algorithms appear to perform better when trained on a balanced training set.

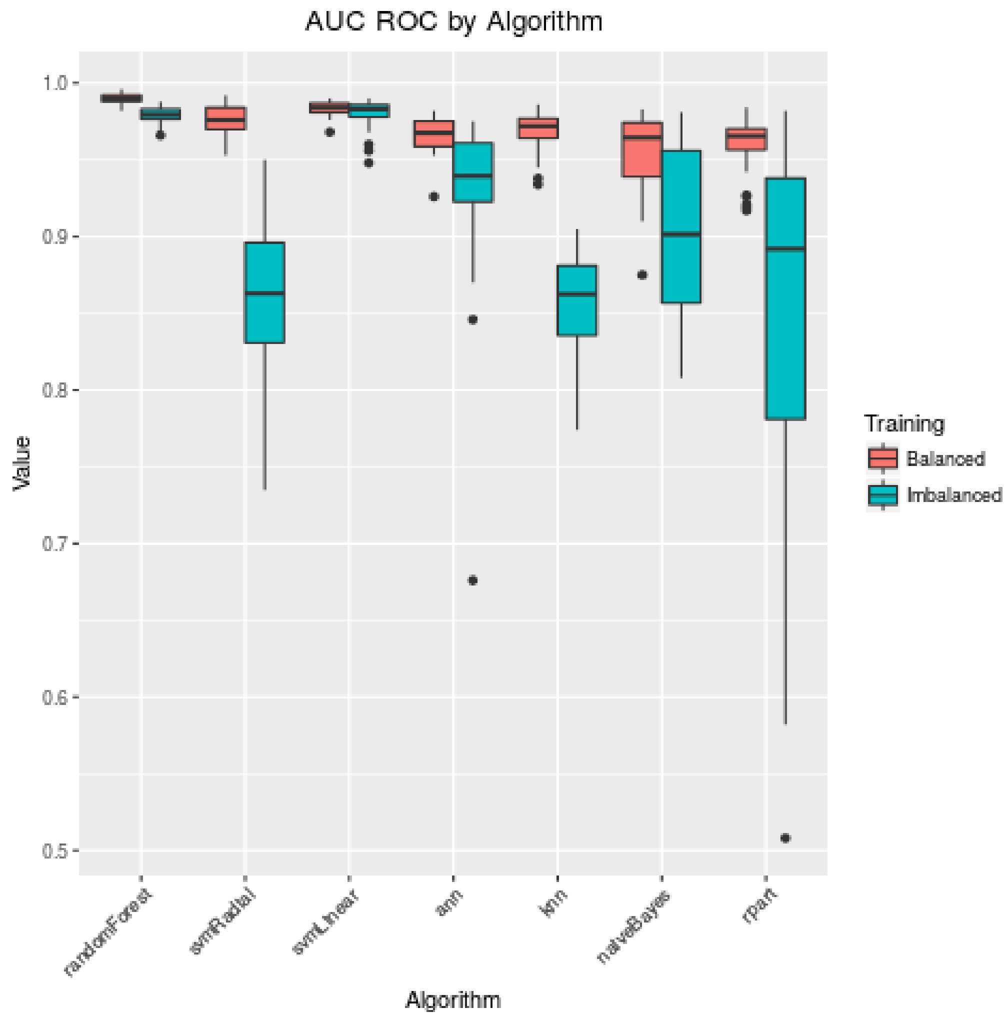

Figure 7.8: **Algorithm Comparison using AUC ROC -** This box plot shows the range of AUC ROC values for models created with both balanced and imbalanced training sets for each algorithm that was assessed. AUC ROC is a useful metric for ranking models on imbalanced problems. Random forest has the best AUC ROC (or ties for best) in all combinations of balanced/stratified training sets. Using a balanced training set improves the AUC ROC of every algorithm.

Figure 7.9 shows similar results as AUC ROC for balanced testing sets when ranking algorithms by AUC precision/recall. However, the second place algorithm by AUC ROC (SVM linear) shows poor AUC precision/recall on imbalanced testing data. Random forest is within the top three algorithms for AUC precision/recall on imbalanced testing data, and first place for balanced testing data.

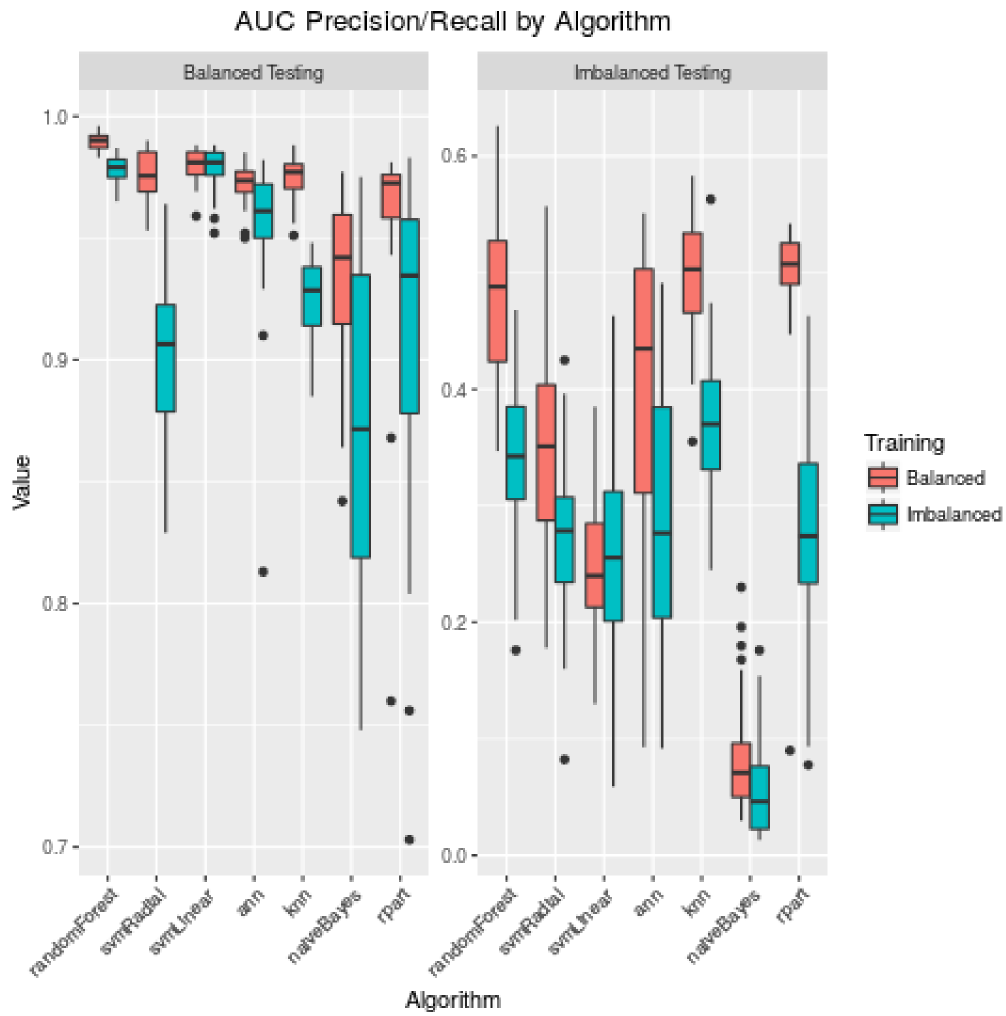

Figure 7.9: **Algorithm Comparison Using AUC Precision/Recall -** This box plot shows the range of AUC precision/recall values for models trained on balanced and imbalanced training sets, and validated with both balanced and imbalanced test sets. The area under the precision/recall curve is a metric that is sensitive to imbalanced data sets. Results are similar to AUC ROC for balanced testing sets. There is no clear winner for imbalanced training and stratified (imbalanced) test sets, but random forest has good performance relative to other algorithms using a balanced training set and testing on a stratified test set.

Table 7.2 shows the performance numbers from the imbalanced training / imbalanced testing data graphed above. Note that for a problem with balanced class labels an AUC precision/recall of 0.5 indicates a test that is no better than a random guess. However, for imbalanced problems the minimum useful AUC precision/recall is related to the class balance ratio.

Table 7.2: **Hg19 Kundaje Class-Imbalanced Performance -** This table shows the performance of models trained on class-stratified blacklist data. The average of common performance metrics for each algorithm is shown, along with the standard deviation of that performance metric in parentheses. Sensitivity, specificity, precision, and F-measure are all based on a threshold prediction probability of 0.5. This may not be the optimal threshold, and for severely imbalanced training sets this may lead to trivial or poorly performing classifiers such as those in this table with a sensitivity or precision of 0.0. The relatively higher AUC ROC and AUC precision/recall data show that these models would perform better at a different threshold, but still do not perform as well as models trained on balanced training data.

Algorithm	**Sensitivity**	**Specificity**	**Precision**	**F-Measure**	**AUC Prec/Recall**	**AUCROC**
randomForest	0.000(0.000)	1.000(0.000)	0.000(0.000)	0.000(0.000)	0.336(0.069)	0.983(0.010)
svmLinear	0.000(0.000)	1.000(0.000)	0.000(0.000)	0.000(0.000)	0.256(0.085)	0.978(0.014)
ann	0.012(0.040)	1.000(0.000)	0.051(0.141)	0.019(0.061)	0.286(0.113)	0.936(0.048)
naiveBayes	0.688(0.137)	0.911(0.066)	0.053(0.048)	0.092(0.076)	0.057(0.043)	0.915(0.046)
svmRadial	0.204(0.079)	0.999(0.000)	0.546(0.096)	0.289(0.093)	0.273(0.072)	0.870(0.057)
rpart	0.220(0.093)	0.999(0.000)	0.384(0.126)	0.275(0.100)	0.288(0.100)	0.865(0.139)
knn	0.269(0.090)	0.999(0.000)	0.518(0.132)	0.343(0.093)	0.373(0.068)	0.848(0.053)

Balanced training datasets perform better across the board for all algorithms, so analysis will focus on these results. Random forest consistently has high performance in these tests. It slightly outperforms SVM linear in terms of AUC ROC, and greatly outperforms SVM linear in AUC precision/recall for imbalanced testing data (a test that accurately reflects application of the model).

In order to select the best training set class balance for model training, training sets with various levels of class imbalance were generated and tested on balanced test sets. The results were plotted for visualization. Our hypothesis is that blacklist classifier performance increases as the imbalance ratio in the training set approaches 1. We observed an increase in sensitivity, and F-measure, AUC ROC, and AUC precision/recall. However, as can be seen in Figure 7.10, specificity also drops slightly as the imbalance ratio approaches 1. This is an unavoidable trade-off.

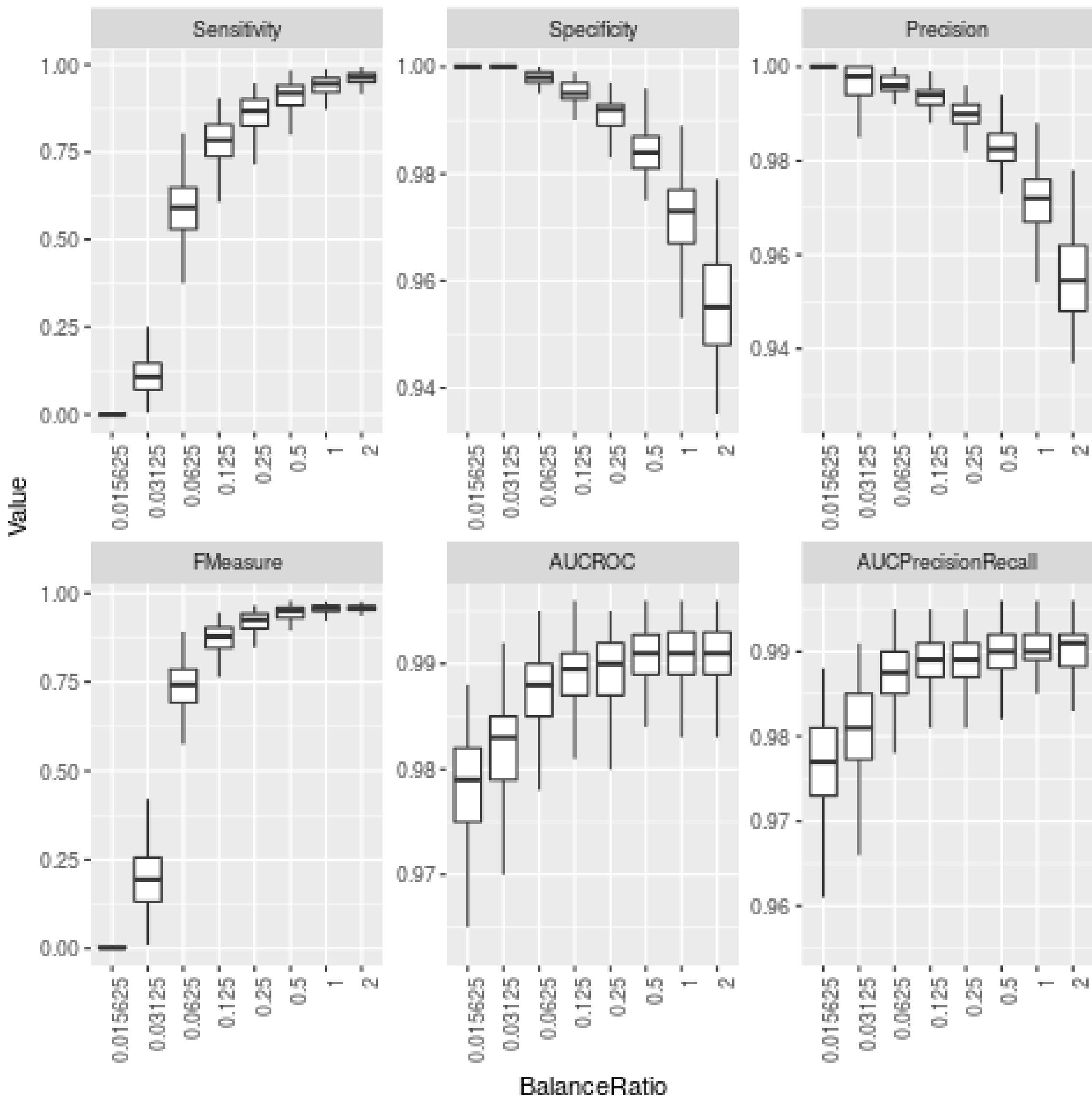

Figure 7.10: **Performance Over Class Sampling Ratios - Top Left:** Sensitivity is plotted over a range of imbalance ratios on the X-axis. Class balance can tune sensitivity from near zero to close to greater than 0.9. There are severely diminishing returns when the minority class is sampled more than the majority class. **Top Center:** Specificity is plotted over a range of imbalance ratios. There is a slight decrease in specificity as the classifier begins to predict more points as the positive class, but it stays above 0.95 for a balanced training set. **Top Right:** Precision also decreases slightly as class labels become more balanced. **Bottom Left:** The F-measure metric behaves similarly to sensitivity, because the sensitivity metric changes much more than the specificity metric as class-balance changes. **Bottom Center:** AUC ROC increases slightly as class labels become more balanced. **Bottom Right:** AUC Precision/Recall behaves similarly to AUCROC.

Another experiment was performed to determine the effect of training set size on model performance, and a suitable training set size for the final model. Performance is expected to increase as training set size increases. However Figure 7.11 shows that there are diminishing returns, and performance plateaus at some point. Our data shows that this point occurs at approximately a training set size of 1000, however we used training sets of size 4000 to be safe.

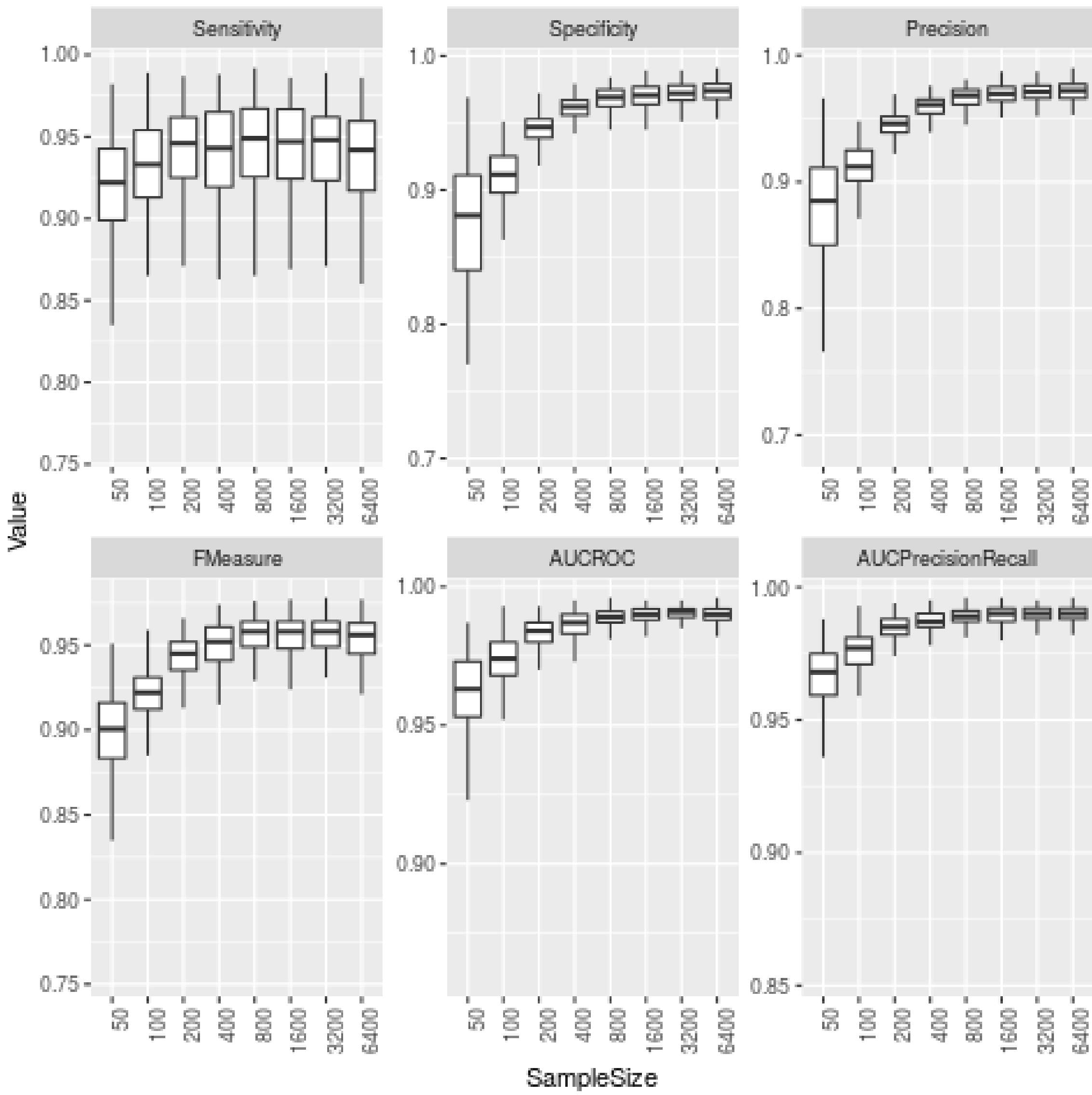

Figure 7.11: **Performance Over Training Set Sizes -** A random forest was trained with training sets of different sizes shown on the X-axis (all class balanced). **Top Left:** No clear correlation was observed between sample size and sensitivity for balanced training sets. **Top Center:** Specificity increases with diminishing returns as sample size goes up. **Top Right:** Precision increases mostly the same way that specificity does. **Bottom Left:** F-Measure also increases, but the effect is less dramatic. **Bottom Center:** AUCROC slightly increases with sample size until an imbalance ratio of approximately 1:4. **Bottom Right:** AUC precision/recall has a very similar curve to AUCROC.

Since blacklist regions vary in size, datasets created with various window sizes were created and classifiers trained on these datasets were validated. Four different window sizes were selected for comparison. For each resolution 32 randomly sampled training and test sets (with 2000 blacklist and 2000 normal regions) were created.

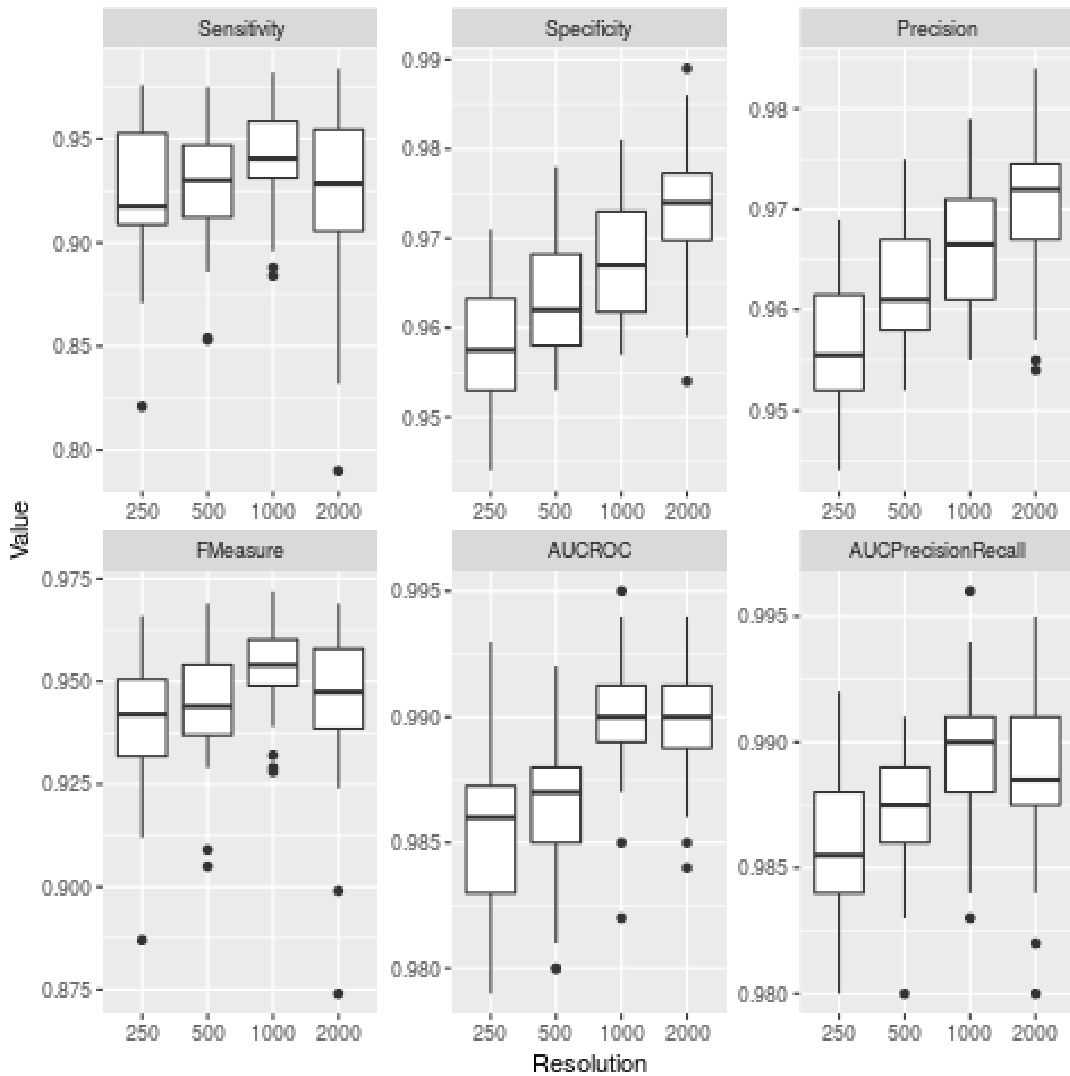

Figure 7.12: **Performance Over Various Windowing Resolutions -** Four different window sizes were tested to determine a suitable resolution for blacklist classification in hg19. There is a slight increase in per-window performance for lower resolution datasets.

To determine the effectiveness of oversampling with overlapping windows an experiment was conducted by training models with normal sampling, SMOTE oversampling, and oversampling by overlapping windows. Oversampling is used when only a small amount of data for one or more classes is available (such as a small genome or blacklist), so the number of data points for training was made artificially small (25 blacklist samples). Seven datasets were created for each set of blacklist items: one with the original training data, as well as SMOTE and sliding window oversampling data sets with various percentages of oversampling of the minority class.

Thirty-two training sets were created for each sampling method. Each dataset contains 2000 normal regions, and either 25, 50, or 75, or 120 blacklist regions depending on oversampling percent. These correspond to imbalance ratios of 1:80, 1:40, 1:27, and 1:16. The additional blacklist samples added with the sliding window oversampling method must overlap with the 25 blacklist training samples. Our data indicates that oversampling of overlapping windows and SMOTE both increase classifier performance, and that both oversampling methods have similar performance.

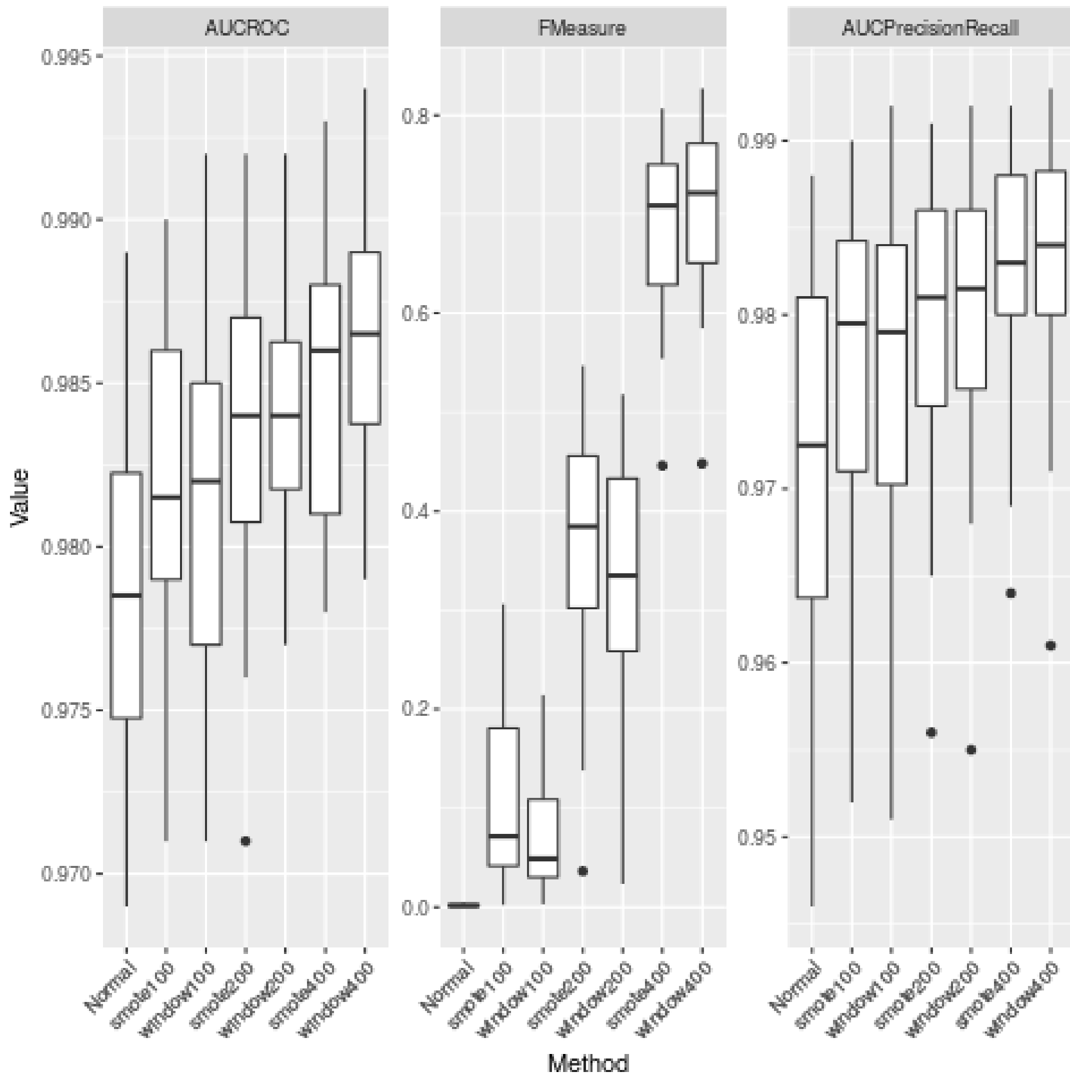

Figure 7.13: **Performance By Sampling Method -** AUCROC, F-Measure (with a threshold of 0.5), and AUC of the precision recall curve are compared for data sets with various levels of oversampling. '100', '200', and '400' refer to the minority class oversampling percentage. SMOTE and sliding window oversampling are very similar in performance. Both oversampling methods lead to an increase of all three performance variables compared.

7.3 Hg19 Improved Blacklist

The predicted blacklist for hg19 has additional blacklisted regions and larger blacklisted regions. A classification based approach measures these regions as false positives. However, these regions may be valid blacklist regions that were simply overlooked by Kundaje. To determine the overall effectiveness of the predicted blacklist, including these new blacklist regions, it was tested on the CHD2 ChIP-Seq dataset[32] previously treated with a blacklist in Chapter 3 using the ENCODE pipeline. Blacklists predicted by the model with a probability thresholds of 0.4 and 0.5 were applied to the CHD2 ChIP-seq dataset[32] described in Chapter 3. Figure 7.14 shows additional improvements in NSC and RSC above the Kundaje blacklist.

As discussed in Chapter 2 and Chapter 3, NSC and RSC are used by ENCODE to measure ChIP-Seq experiment quality, and blacklist treatments can improve these quality metrics. True ChIP-Seq peaks are highly correlated at a shift length of approximately the length of immunoprecipitated fragments. High quality ChIP-Seq experiments have a high fragment-length correlation compared to read-length correlation (the "phantom peak"), and a low minimum cross-correlation.

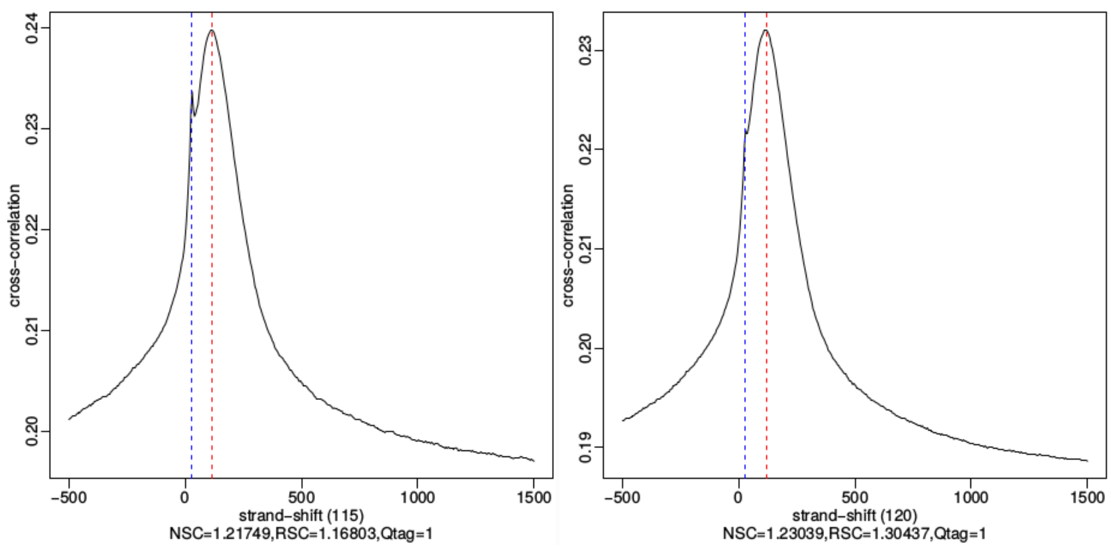

Figure 7.14: **Improved Hg19 Blacklist Cross-Correlation - Left:** Strand-shifted cross-correlation for biological replicate 1 of the CHD2 dataset without blacklist treatment. **Right:** The 0.4 thresholded blacklist improves both the NSC and RSC above the Kundaje blacklist. The phantom peak (marked by the dashed blue line) is almost completely reduced to the background cross-correlation at read length strand-shift.

While the maximum cross-correlation is lower than the Kundaje blacklist treatment (indicating that some regions with high cross-correlation were filtered), the NSC is higher. This indicates that blacklist treatments trade peak sensitivity for specificity. This is because a blacklist may remove regions with true ChIP-Seq peaks, but a useful blacklist (better than a random guess) will remove more bad regions than good regions. The cross-correlation curve shown in Figure 3.3 indicates that the predicted blacklist results in more specific ChIP-Seq peak calling, but also less sensitive peak calling.

Table 7.3: **Hg19 CHD2 NSC and RSC by Blacklist Treatment -** NSC and RSC signal-to-noise ratio estimates using various blacklist treatments for both biological replicates of the CHD2 dataset. Kundaje's blacklist improves upon no treatment in both metrics, and the blacklist predicted with PeakPass improves upon the Kundaje treatment in both metrics.

Treatment	Rep-1 NSC	Rep-1 RSC	Rep-2 NSC	Rep-2 RSC
No Blacklist	1.217	1.168	1.197	1.129
Kundaje Blacklist	1.219	1.216	1.198	1.179
50% Blacklist	1.224	1.242	1.204	1.206
40% Blacklist	1.230	1.304	1.207	1.264

7.4 TAIR10 Blacklist Prediction

The classifier was applied to the *Arabidopsis thaliana* (TAIR10) genome[77] in an attempt to create a novel blacklist, which will then be evaluated by a biologist with expertise in that organism. Suspicious regions, such as the one shown in Figure 7.15, in the TAIR10 genome were manually curated using existing ChIP-Seq experiments to create a training set. These ChIP-Seq datasets, including GSM554340, GSM554339, GSM424618, GSE58083, GSM1029659 [78, 79, 80, 81, 82], were downloaded from the NCBI GEO database. The full blacklist predicted by the model was then tested by using it as a filter when applying the ENCODE pipeline to a ChIP-Seq experiment excluding from the experiment set used to generate the training data.

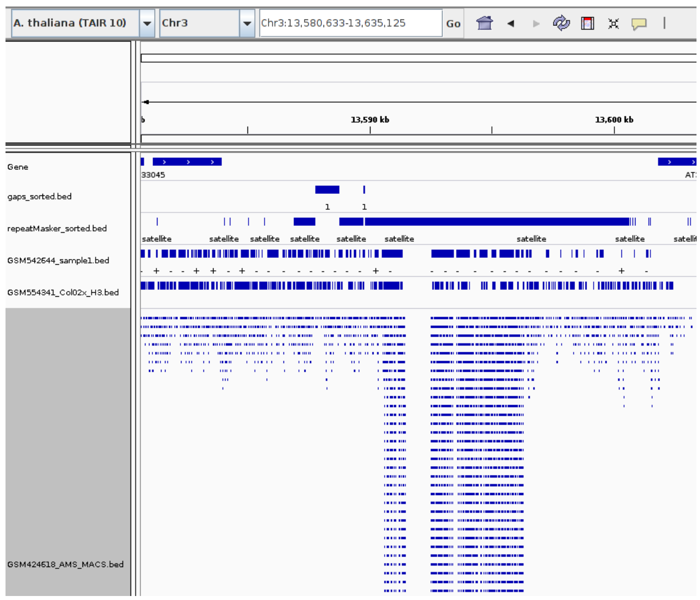

Figure 7.15: **Suspect TAIR10 Region -** This TAIR10 region viewed in IGV contained a high number of unstructured reads across many unrelated experiments. The region cannot be a real peak, because it is orders of magnitude larger than any single protein binding region (approximately 5000bp). It is inside of a satellite repeat, and next to a gap.

The training blacklist that we created using suspect regions across 5 different ChIP-Seq experiments is shown below. The *Arabidopsis thaliana* is much smaller than the human genome (approximately 120Mbp, compared to 3,150 Mbp for the human genome). Therefore, we expect a much smaller blacklist for TAIR10.

```
1    15084213    15088629
3    13590430    13596553
3    14205986    14214430
```

Next a model was trained using model on the small blacklist, and two blacklists were created by using the model to predict every window in the TAIR10 genome assembly; one for regions with a blacklist probability of greater than 50%, and one greater than 40%. The model predicted many additional blacklist regions than the original three used for training.

Next, the blacklist was applied to a working ChIP-Seq dataset by removing all reads aligned withing a blacklist region. This is done using the bedtools *intersect* command (with the inverse option). The number of reads removed by the filter were then determined using the *wc* command. Results are recorded in Table 7.4.

Table 7.4: **TAIR10 AP1 Read Counts and Blacklist Filtering -** Read counts from biological replicates 1 and 2 from the AP1 ChIP-Seq dataset[24] with different blacklist treatments.

Treatment	**Replicate 1**	**Replicate 2**
No Blacklist	30,618,234	20,119,752
50% Blacklist	29,589,954	19,446,218
40% Blacklist	28,803,523	18,921,302

Next, the ENCODE pipeline[21] was run on the original AP1 ChIP-Seq dataset[24], and two different blacklist treatment data sets. Performance metrics and output peak files were compared to determine what effect the treatments had. These performance metrics are discussed in Chapter 2. Figure 7.16 below shows the impact of a blacklist produced including windows with a predicted blacklist probability threshold of 0.5.

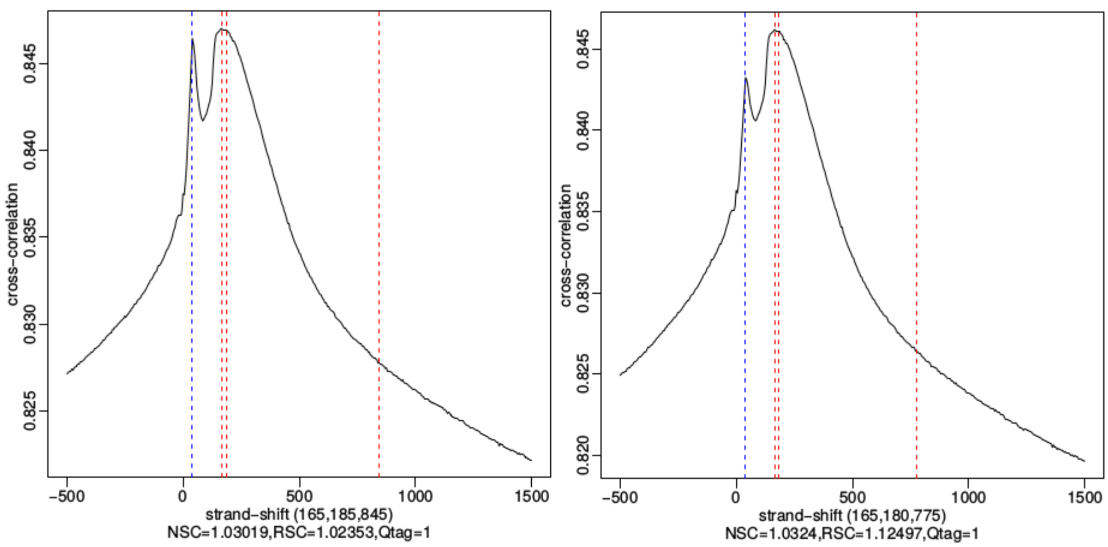

Figure 7.16: **TAIR10 AP1 Replicate 1 Blacklist Impact - Left:** The strand-shifted cross-correlation graph for biological replicate 1 from the AP1 ChIP-Seq dataset[24] for *Arabidopsis thaliana* with no blacklist applied. **Right:** The same biological replicate filtered with blacklist regions with a probability of greater than 50% shows improved NSC and RSC. Note that the Y-scales are different between the two graphs.

An even more sensitive blacklist, including all windows with a predicted blacklist probability of 0.4, was able to nearly eliminate the phantom peak altogether. As shown in Figure 7.17, background cross-correlation was also reduced. The strand-length cross-correlation signal is slightly lower, but the overall characteristics of the curve are greatly improved.

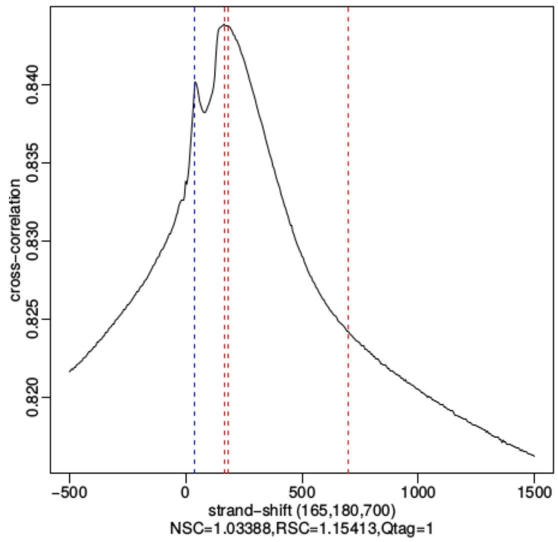

Figure 7.17: **TAIR10 AP1 Replicate 1 Blacklist Impact - 40% Threshold -** The blacklist generated with regions of confidence greater than 40% even further reduced the phantom peak. It also reduced the fragment-length peak and the minimum cross-correlation, resulting in a slightly better NSC, and a significantly better RSC.

Thet output peak files with different treatments are very similar, however there were a few differences of note. The number of output peaks was higher after blacklist treatment. The amount of overlap was calculated by using the bedtools[68] *jaccard* command. The Jaccard correlation coefficient indicates takes into account peaks that appear in one treatment but not the other, as well as peaks that shifted between treatments, but still overlap. The average length of peaks decreased with blacklist treatment, and the peak length variance also decreased. The average peak signal and peak signal variance also decreased.

Table 7.5: **TAIR10 Ap1 Peaks Comparison -** This table compares the peak BED files output by the ENCODE pipeline with no blacklist, blacklist regions with 50% predicted confidence, and blacklist regions with 40% predicted confidence. The number of peaks, mean length and standard deviation, signal mean and standard deviation, number of added and removed peaks, and Jaccard coefficient between the output and the original output are included.

Treatment	**Peaks**	**μ Length**	**σ Length**	**μ Signal**	**σ Signal**	**Jaccard**
Original	11518	653.263	62.854	180.658	78.284	1.0
50PercentBlacklist	11700	639.771	61.023	177.201	76.848	0.789659
40PercentBlacklist	11660	633.992	59.889	174.909	76.367	0.718272

8. DISCUSSION

Genomic blacklists are often used to improve the quality of ChIP-Seq experiments[7, 5]. Currently, ChIP-Seq blacklists are created by experts, and investigators have to wait for a blacklist for their organism of interest or perform ChIP-Seq without a blacklist. Our supervised learning approach breaks the genome down into windows and classifies these windows as either blacklist or non-blacklist using a small training blacklist, reducing the amount of work required by the expert.

Blacklist treatments improve ChIP-Seq quality metrics such as signal-to-noise ratio (as estimated with NSC and RSC)[5]. The probabilities reported by PeakPass allow users to tune blacklists with various sensitivities and specificities, which gives them more control over filtering than simple boolean filters. For example, blacklists created for the human genome by PeakPass with a lower threshold parameter contain genomic regions not included in the original Kundaje blacklist, and these blacklists further improved NSC and RSC in a small set of ChIP-Seq experiments.

In our analysis, random forest performs best for predicting blacklist regions, but classifiers such as SVM, KNNs, and ANNs have comparable performance. In our study, we used a training set size of 4000 (2000 blacklist and 2000 normal samples), however our tests indicate that as few as 1000 training samples are sufficient for a high quality model. This makes it possible to build blacklists for smaller genomes, such as the *Arabidopsis thaliana* genome, as well as smaller blacklists, such as the hg38 blacklist.

Providing an approximately balanced training set is important to optimize model performance. Random forest is somewhat resistant to training set class imbalance, but cannot mitigate the extreme imbalance ratio of blacklist regions within the human genome. We observed a small increase in sensitivity when the minority class was sampled at a higher rate than the majority class in training data. However, the effect was not large enough to justify this approach.

Experiments with windows of different lengths show that longer windows may be slightly easier to classify. Specificity, precision, AUC ROC, and AUC precision/recall all increased with increased window size. However it should be noted that this leads to lower resolution. If the window size is cut in half, some additional false positives and false negatives are expected. However, as shown in Figure 8.1, each false positive and false negative is smaller. The total number of incorrectly called loci may be lower even if the there are more erroneously labeled windows. We recommend that window size not be significantly more than the shortest blacklist items.

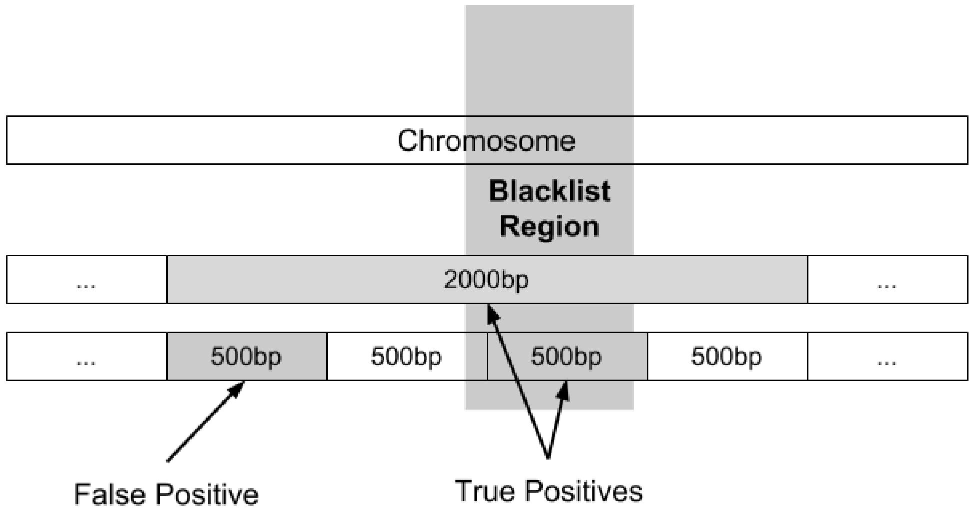

Figure 8.1: **Resolution/Error Trade-off -** Two windowing resolutions are shown to illustrate that window-level error and loci-level error are a trade-off. Regions highlighted in gray refer to predicted or actual blacklist regions. The 2000bp window is a true positive, so there are no incorrect window positions within this region, however in a scenario where the 500bp resolution track has a false positive window. The 500bp resolution track still outperforms the 2000bp track in terms of correctly classified genomic loci.

In cases where blacklists, genomes, or both are too small to provide sufficient minority class samples, we suggest two methods for oversampling blacklist regions: SMOTE[55] and overlapping windows. SMOTE creates synthetic data points by averaging features of existing data points in the minority class. However, our approach which uses overlapping, shifted windows allows for additional data points to be sampled without the need for blacklist regions or the creation of synthetic data points. In 32 randomly sampled training and testing sets, the overlapping window approach performed as well or better than SMOTE in AUC ROC, F-measure, and AUC precision/recall. Most likely, these two methods can be used simultaneously to further improve performance, but we do not have experimental evidence for this claim, and leave this experiment for future work.

In order to make the classifier easily portable between genome versions and organisms, we only use features that are available solely from a reference genome sequence. Depending on organism and the maturity of the genome assembly, some features may be absent from some organisms, or they may be very different. This is the case for a few features that were considered, such as assembly gap size, gap type, and repeat type. Genome assembly and sequence based features such as distance from assembly gaps, CG content, and distance from predicted genes are all available even if no sequence mapping experiments have been performed on the organism.

Nucleotide content was initially considered as a feature because it can cause a bias in sequencing[67]. Ross, et al. have shown that Illumina MiSeq sequencing, a method commonly used for ChIP-Seq, has a coverage bias due to GC content[83]. GC content was originally considered as a feature, but it was split into the percentage content for each nucleotide. Feature visualizations show that A, T, C, and G are biased toward extremes in blacklist regions. Clusters of blacklist regions that exhibit bias in two or more nucleotide content features were also observed (see Figure 6.5) Decision trees, KNN, SVMs, and ANNs are capable in theory of dealing with two-feature clusters. This may explain why these algorithms performed better than naive Bayes during algorithm comparison.

Ultimately, a reference genome is an imperfect substitution for the sampled individual's genome. Figure 8.2 below shows an example of a tandem repeat that has been expanded in an individual's genome compared to the reference genome. A literature review shows that polymorphism in tandem repeats is a common event[66, 65]. Other genomic regions that exhibit polymorphism may cause ChIP-Seq noise, and are also blacklist candidates.

Figure 8.2: **Tandem Repeat Mutation -** While the reference genome only contains four instances of the short tandem repeat 'GAT', an individual that was sampled for a ChIP-Seq experiment may have significantly more instances of the repeat.

Alignability-based features did not behave as initially expected. Since Kundaje uses the multi-mapping to uniquely-mapping loci ratio in genomic windows when curating his blacklist, we expected a large difference in alignability feature between the blacklist and normal regions. While there is a clear alignability difference between the two class labels, a large number of both blacklist and normal regions contained no mappable loci (no reads will be mapped to these regions). Perhaps a different set of features derived from the alignability (using nearby alignability) would help to separate these regions.

To apply our method, we created a blacklist for the TAIR10 genome assembly by generating a list of suspicious regions and training PeakPass on this list. While the set of suspicious regions is short, and may not be large enough to contain a representative sample, the predicted blacklist still improved ChIP-Seq quality on our evaluated dataset. Our predicted blacklist improved NSC, RSC, and the IDR curves when running the ENCODE pipeline on a TAIR10 ChIP-Seq dataset[24]. This blacklist needs to be tested on more ChIP-Seq datasets, but initial results are promising.

8.1 Future Work

The feature gathering pipeline is capable of saturating the CPUs of single machine even for machines with 16 or 24 cores. Our current multithreading scheme executes a separate process for each chromosome, and limits the maximum number of simultaneous processes. However, the same scheme could be easily use cluster computing to improve running time. Workloads could be broken down further by splitting the chromosome into one part for every core available in the cluster. Model training and validation are also candidates for parallelization with a compute cluster.

Additional experiments should be conducted before investigators can be confident in these predicted ChIP-Seq blacklists. Other studies with blacklists measure quality metrics for hundreds of ChIP-Seq datasets, and perform hypothesis testing between treatments[5]. An automated experiment would be required to handle this amount of work, but would also allow for comparison of blacklists with different prediction thresholds.

In addition to improving algorithm performance, some additional applications should be considered for the methods and algorithms described in this thesis. The output blacklists are designed to work with ChIP-Seq data, and may not be appropriate for RNA-Seq and other mapping experiments. However, it is possible that future algorithms could target other experiment types. Many of the features are likely reusable for such algorithms, because they affect sequencing bias. However, a different set of "normal" data would be required for training.

Beyond generation of blacklists, the pipeline created in this paper could be easily refactored to work on other classification problems where genomic intervals must be annotated. While different classification problems require different features, the scripted pipeline is configurable, and it is easy to add or remove features.

REFERENCES

[1] National Human Genome Research Institute (NHGRI). 2017. The Cost of Sequencing a Human Genome.
`https://www.genome.gov/27565109/the-cost-of-sequencing-a-human-genome/`.

[2] Jkwchui, Wikipedia 2017. Chromatin immunoprecipitation sequencing.svg (Modified Under Creative Commons Attribution-Share Alike 3.0 Unported).
`https://commons.wikimedia.org/wiki/File:Chromatin_immunoprecipitation_sequencing.svg`.

[3] Kharchenko, P., Tolstorukov, M. and Park, P. 2008. Design and analysis of ChIP-seq experiments for DNA-binding proteins. *Nature Biotechnology, 26*, 12, 1351-1359.

[4] The 1000 Genomes Project Consortium. 2010. A map of human genome variation from population scale sequencing. *Nature, 467*, 7319, 1061-1073.

[5] Carroll, T. et al. 2014. Impact of artifact removal on ChIP quality metrics in ChIP-seq and ChIP-exo data. *Frontiers in Genetics, 5*.

[6] Robertson, G. et al. 2007. Genome-wide profiles of STAT1 DNA association using chromatin immunoprecipitation and massively parallel sequencing. *Nature Methods, 4*, 651-657.

[7] Kundaje, A. 2013. A comprehensive collection of signal artifact blacklist regions in the human genome.
`https://personal.broadinstitute.org/anshul/projects/encode/rawdata/blacklists/hg19-blacklist-README.pdf`.

[8] Kundaje, A. 2014. mod/mouse/humanENCODE: Blacklisted genomic regions for functional genomics analysis.
`https://sites.google.com/site/anshulkundaje/projects/blacklists`.

[9] Kent, W. et al. 2002. The human genome browser at UCSC. *Genome Research, 12*, 6, 996-1006.

[10] Kundaje, A. 2017. ENCODE: Wiggler - Normalized genome-wide signal coverage tracks of ENCODE data.
`https://sites.google.com/site/anshulkundaje/projects/wiggler`.

[11] Hoffman, M. et al. Umap and Bismap: quantifying genome and methylome mappability.
`https://bitbucket.org/hoffmanlab/umap/overview`.

[12] Kaufmann, K. et al. 2010. Chromatin immunoprecipitation (ChIP) of plant transcription factors followed by sequencing (ChIP-SEQ) or hybridization to whole genome arrays (ChIP-CHIP). *Nature Protocols, 5*, 3, 457-472.

[13] Hoffman, E. et al. 2015. Formaldehyde Crosslinking: A Tool for the Study of Chromatin Complexes. *Journal of Biological Chemistry, 290*, 44, 26404-26411.

[14] Furey, T. 2012. ChIP-seq and Beyond: new and improved methodologies to detect and characterize protein-DNA interactions. *Nature Reviews Genetics, 13*, 12, 840-852.

[15] Bailey, T. et al. 2013. Practical Guidelines for the Comprehensive Analysis of ChIP-seq Data. *PLOS Computational Biology, 9*, 11.

[16] Snyder, M. 2017. TOE1 ChIP-seq on human MCF-7.
`https://www.encodeproject.org/experiments/ENCSR810WXH/`.

[17] Robinson, J. et al. 2011. Integrative Genomics Viewer. *Nature Biotechnology, 29*, 24-26.

[18] ENCODE. 2017. ENCODE: Encyclopedia of DNA Elements.
`https://www.encodeproject.org`.

[19] Lassmann, T., Hayashizaki, Y. and Daub, C. 2009. TagDust - a program to eliminate artifacts from next generation sequencing data. *Bioinformatics, 25*, 21, 2839-2840.

[20] Chen, Y. et al. 2012. Systematic evaluation of factors influencing ChIP-seq fidelity. *Nature Methods, 9*, 6, 609-614.

[21] Landt, S. et al. 2012. ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia. *Genome Research, 22*, 9, 1813-1831.

[22] Diaz, A., Nellore, A., and Song, J. 2012. CHANCE: comprehensive software for quality control and validation of ChIP-seq data. *Genome Biology, 13*, 10, R98.

[23] Ramachandran, P. et al. 2013. MaSC: mappability-sensitive cross-correlation for estimating mean fragment length of single-end short-read sequencing data. *Bioinformatics, 29*, 4, 444-450.

[24] Pajoro, A. et al. 2014. Dynamics of chromatin accessibility and gene regulation by MADS-domain transcription factors in flower development. *Genome Biology, 15*, 3, R41.

[25] Zhang, Y. et al. 2008. Model-based Analysis of ChIP-Seq (MACS). *Genome Biology, 9*, 9, R137.

[26] Hoffman, M. et al. 2012. Integrative annotation of chromatin elements from ENCODE data. *Nucleic Acids Research, 41*, 2, 827-841.

[27] Scheinin, I. et al. 2014. DNA copy number analysis of fresh and formalin-fixed specimens by shallow whole-genome sequencing with identification and exclusion of problematic regions in the genome assembly. *Genome Research, 24*, 12, 2022-2032.

[28] Furey, T. Duke Excluded Regions.

[29] Dolgalev, I., Sedlazeck, F. and Busby, B. 2017. DangerTrack: A scoring system to detect difficult-to-assess regions. *F1000Research, 6*, 443.

[30] Cardone, M. et al. 2004. Evolution of beta satellite DNA sequences: evidence for duplication-mediated repeat amplification and spreading. *Molecular Biology and Evolution, 21*, 9, 1792-1799.

[31] Matera, G., Terns, R. and Terns, M. 2007. Non-coding RNAs: lessons from the small nuclear and small nucleolar RNAs. *Nature Reviews Molecular Cell Biology, 8*, 209-220.

[32] Snyder, M. 2011. Experiment summary for ENCSR000DZR.
`https://www.encodeproject.org/experiments/ENCSR000DZR/`.

[33] John, G. and Langley, P. 2013. Estimating Continuous Distributions in Bayesian Classifiers. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*.

[34] Webb, G., Boughton, J. and Wang, Z. 2005. Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning, 58*, 1, 5-24.

[35] Cortes, C. and Vapnik, V. 1995. Support-Vector Networks. *Machine Learning, 20*, 3, 273-297.

[36] Vert, J., Tsuda, K. and Schölkopf, B. 2004. A primer on kernel methods. *Kernel Methods in Computational Biology*, 35-70.

[37] Therneau, T., Atkinson, E. and Ripley, B. 2017. Package 'rpart'.
`https://cran.r-project.org/web/packages/rpart/rpart.pdf`.

[38] Milborrow, S. 2016. Plotting rpart trees with the rpart.plot package.
`http://www.milbo.org/rpart-plot/prp.pdf`.

[39] Therneau, T. and Atkinson, E. 2017. An Introduction to Recursive Partitioning Using RPART Routines.

[40] Russell, S. and Norvig, P. 2010. *Artificial Intelligence - A Modern Approach.*

[41] Ho, T. 1995. Random Decision Forests. *Proceedings of the Third International Conference on Document Analysis and Recognition, 1*, 278.

[42] Breiman, L. 2001. Random Forests. *Machine Learning, 45*, 1, 5-32.

[43] Breiman, L. and Cutler, A. 2017. Random Forests. `https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`.

[44] Liaw, A. 2015. Package 'randomForest'.

[45] Fix, E. and Hodges, J. 1951. Discriminatory Analysis Nonparametric Discrimination: Consistency Properties.

[46] Aha, D., Kibler, D. and Albert, M. 1991. Instance-Based Learning Algorithms. *Machine Learning, 6*, 37-66.

[47] Farley, B. and Clark, W. 1954. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory, 4*, 4, 76-84.

[48] Riedmiller, M. 1994. Advanced supervised learning in multi-layer perceptrons - From backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces, 16*, 3, 265-278.

[49] Karlik, B. and Olgac, V. 2011. Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. *International Journal of Artificial Intelligence And Expert Systems, 1*, 111-122.

[50] Camelia, L. and Potolea, R. 2011. Imbalanced Classification Problems: Systematic Study, Issues and Best Practices. *Enterprise Information Systems. ICEIS 2011. Lecture Notes in Business Information Processing, 102*, 35-50.

[51] Krawczyk, B. 2016. Learning From Imbalanced Data: Open Challenges and Future Directions. *Progress In Artificial Intelligence, 5*, 4, 221-232.

[52] Lindsey, T. et al. 2015. Global Cancer Incidence and Mortality Rates and Trends-An Update. *Cancer Epidemiology, Biomarkers, and Prevention, 25*, 1, 16-27.

[53] Fawcett, T. 2006. An introduction to ROC analysis. *Pattern Recognition Letters, 27*, 8, 861-874.

[54] Sing, T. et al. 2015. Package 'ROCR'.

[55] He, H. and Garcia, E. 2009. Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering, 21*, 9, 1263-1284.

[56] Chen, C., Liaw, A. and Breiman, L. 2004. Using Random Forest to Learn Imbalanced Data.

[57] Qi, C. et al. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2*, 77-85.

[58] Oilfox, Wikimedia Commons. 2017. Drafting Format (under Creative Commons Attribution-Share Alike 3.0 Unported). `https://commons.wikimedia.org/wiki/File:Drafting_Format.jpg`.

[59] İrsoy, O., Yıldız, O. and Alpaydın, E. 2012. Design and Analysis of Classifier Learning Experiments in Bioinformatics: Survey and Case Studies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 9*, 6, 1663-1675.

[60] Institute for Systems Biology. 2017. RepeatMasker Documentation.
`http://www.repeatmasker.org/webrepeatmaskerhelp.html`.

[61] Cormen, T., Leiserson, C., Rivest, R., Stein, C. 2009. Introduction to Algorithms (Third Edition).

[62] Sipser, M. 2006. Introduction to the Theory of Computation (Second Edition).

[63] Langmead, B. et al. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology, 10*, R25.

[64] Pickrell, J. et al. 2011. False positive peaks in ChIP-seq and other sequencing-based functional assays caused by unannotated high copy number regions. *Bioinformatics, 27*, 15, 2144-2146.

[65] Weber, J. and Wong, C. 1993. Mutation of human short tandem repeats. *Human Molecular Genetics, 2*, 8, 1123-1128.

[66] Benson, G. 1999. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research, 27*, 2, 573-580.

[67] Chen, Y. et al. 2013. Effects of GC Bias in Next-Generation-Sequencing Data on De Novo Genome Assembly. *PLOS ONE, 8*, 4.

[68] Bedtools. 2017. bedtools: a powerful toolset for genome arithmetic.
`http://bedtools.readthedocs.io/en/latest/index.html`.

[69] Dhawal, J. et al. 2015. Active promoters give rise to false positive 'Phantom Peaks' in ChIP-seq experiments. *Nucleic Acids Research, 43*, 14, 6959-6968.

[70] NCBI. 2017. Assembly Information.
`https://www.ncbi.nlm.nih.gov/assembly/basics/`.

[71] Iafrate, J. et al. 2004. Detection of large-scale variation in the human genome. *Nature Genetics, 36*, 9, 949-951.

[72] Scarfe, T. and Kalnishkan, Y. 2009. Speech segment classification on music radio shows using machine learning algorithms.

[73] Rabaoui, A. et al. 2008. Using Robust Features with Multi-Class SVMs to Classify Noisy Sounds. *3rd International Symposium on Communications, Control and Signal Processing*, 594-599.

[74] Chandrashekar, G. and Sahin, F. 2013. A survey on feature selection methods. *Computers & Electrical Engineering, 40*, 1, 16-28.

[75] Kuhn, M. et al. 2017. Package 'caret'.

[76] Kundaje, A. 2017. ENCODE: TF ChIP-seq peak calling using the Irreproducibility Discovery Rate (IDR) framework.
`https://sites.google.com/site/anshulkundaje/projects/idr`.

[77] Berardini, T. et al. 2015. The Arabidopsis Information Resource: Making and mining the "gold standard" annotated reference plant genome.. *Genesis: The Journal of Genetics and Development, 53*, 8, 474-485.

[78] Ha, M. 2010. GSM554340.
`https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM554340`.

[79] Ha, M. 2010. GSM554339.
`https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM554339`.

[80] Zhang, D. 2009. GSM424618.
`https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM424618`.

[81] Li, B. 2014. GSE58083.
https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE58083.

[82] Li, L. 2012. GSM1029659.
https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM1029659.

[83] Ross, M. et al. 2013. Characterizing and measuring bias in sequence data. *Genome Biology, 14*, R51.

[84] Finn, R. et al. 2016. The Dfam Database of Repetitive DNA Families. *Nucleic Acids Research, 44*, D1, D81-D89.
http://www.dfam.org.

[85] Altemose, N. et al. 2014. Genomic Characterization of Large Heterochromatic Gaps in the Human Genome Assembly. *PLoS Computational Biology, 10*, 5.

[86] Karatzoglou, A., Smola, A. and Hornik, K. 2016. Package 'kernlab'.

APPENDICES

A. ADDITIONAL BLACKLIST DETAILS

Table A.1: **Duke Blacklist -** Types of regions and their frequencies in the Duke blacklist.

Region Type	**#**	**Description**

ACRO1	33	C rich acromeric repeat 147bp long[84].
		`tcacccccagccctcagggactctcctgtctctcccagctactctccaaccatgcccagatttcag`
		`cgggagtcagttccaggcacccaggaatcacctacaaactcaccaatttcactgagttcctcc...`

ALR/Alpha	529	A and T rich centromeric repeat 171bp long[84].
		`cattctcagaaacttctttgtgatgtgtgcattcaactcacagagttgaacctttcttttgataga`
		`gcagttttgaaacactctttttgtagaatctgcaagtggatatttggagcgctttgaggccta...`

BSR/Beta	223	A and G rich satellite repeat 152bp long[84].
		`Gctgggttcagcaatatgtcacaatttcttctgtggggcaggttcaggcagaagagaagagtcaca`
		`tcacctaggtgctgggctcagcaatatgtcacaatttcttctgtggggcaggtccaggcagaa...`

(CATTC)n	74	A simple repeat associated with heterochromatin[85].
chrM	60	Sequences that resemble mitochondrial DNA.
(GAATG)n	92	A simple repeat associated with heterochromatin[85].
(GAGTG)n	46	A simple repeat motif detected by RepeatMasker.

HSATII	15	C and T rich satellite repeat 170bp long[84].
		`ccattcgattccattcgatgattccattcgattccattcgatgatgattccattcgattccattcg`
		`atgattccattcgattccattcgatgatgattccattcgattccattcgatgattccattcga...`

LSU-rRNA_Hsa	354	Ribosomal RNA that contains extremely repetitive sequences as shown below[84].
		`tctttcctgcctcccgttcctcccgacccctccaaccgccctccctgcctcggccacccctcctcc`
		`tccttcctggaggtggcgggccccagcaggtgcggaggtgggtgggttgggcaggcgggggagtcc`
		`ttgggggactgtcctccagacggcgactggccaccgccagatgcatttccactgtggtgctgcgcc`
		`gtgactggctctgggacggctggaaaggcccggcagggaaggtggctcgggggggtcatgtccgtcc`
		`ggccatccctctggggggggggggggggggggggggggggggggggggggggggggggggggg...`

SSU-rRNA_Hsa	76	Ribosomal RNA that contains extremely repetitive sequences as shown below[84].
		`ctgtggtaattctagagctaatacatgccgacgggcgctgacccccttcgcgggggggatgcgtgc`
		`atttatcagatcaaaaccaacccggtcagcccctctccggccccggccgggggggcgggcgccggcg`
		`gctttggtgactctagataacctcgggccaatcgcacgcccccccgtggcggcgacgacccattcga`
		`acgtctgccctatcaactttcgatggtagtcgctgtgcctaccatggtgaccacgggtgacgg...`

TAR1	148	A telomeric repeat 2110bp long that contains extremely repetitive sequences as shown below[84].
		`accaacagtgagggggtcccacaagcctaggtggggcaagtcagggatctaaggaaggagcaggaa`
		`aaccaaagaaaagaggtggagacttgagacagaggcaggaatgtgaagaagtccaacataaaactc`
		`cctgctcaggaccctggggctgtgttcctgcacgatcagcccacttctccctatttttctacaata`
		`agctctttacactgtatttcttttcaatgaagttatctgccatctttgtactgcctcttggtgaga`
		`atctttcttccaagttaagaactgggacatcaggtctccccagtattagctccatttcagttt...`

Total	1649	-

Table A.2: **Consensus Blacklist -** Types of regions and their frequencies in the blacklist made by combining the Kundaje and Duke blacklists.

Region Type	**Frequency**
ACRO1	3
ALR/Alpha	25
BSR/Beta	78
(CATTC)n	6
Centromeric Repeat	76
chrM	6
(GAATG)n	2
(GAGTG)n	1
High Mappability Island	7
Low Mappability Island	100
LSU-rRNA_Hsa	33
Satellite Repeat	36
snRNA	2
SSU-rRNA_Hsa	11
TAR1	20
Telomeric Repeat	5
Total	411

B. ALGORITHM TUNING

Table B.1: **ANN Tuning Parameters -** Tuning parameters for artificial neural networks sorted by ROC AUC from highest to lowest.

Hidden Layer Size	Decay	ROC AUC	Sensitivity	Specificity
9	1.0	0.9612	0.9321	0.9312
3	1.0	0.9590	0.9445	0.9213
5	1.0	0.9583	0.9487	0.9223
9	0.1	0.9578	0.9305	0.9262
3	0.5	0.9577	0.9357	0.9270
9	0.5	0.9571	0.9361	0.9225
5	0.1	0.9559	0.9361	0.9307
5	0.5	0.9550	0.9356	0.9235
3	0.1	0.9501	0.9196	0.9268
1	1.0	0.9292	0.9146	0.9328
1	0.5	0.9278	0.9123	0.9328
1	0.1	0.9146	0.8841	0.9358
9	0.0	0.8282	0.6666	0.9595
5	0.0	0.8278	0.6822	0.9610
3	0.0	0.7901	0.5906	0.9732
1	0.0	0.6091	0.2314	0.9868

Table B.2: **SVM (Radial) Tuning Parameters -** Tuning parameters for SVM with a radial kernel sorted by ROC AUC from highest to lowest. ‘Cost’ is the cost of a training data point falling on the wrong side of the decision boundary[86]. ‘Sigma’ is a variable in the radial basis function kernel[86].

Cost	Sigma	ROC AUC	Sensitivity	Specificity
2.0	0.2	0.9877	0.9146	0.9863
2.0	0.5	0.9877	0.9140	0.9868
2.0	1.0	0.9877	0.9140	0.9868
2.0	1.5	0.9877	0.9140	0.9867
2.0	2.0	0.9877	0.9149	0.9863
2.0	5.0	0.9877	0.9149	0.9867
1.0	0.2	0.9873	0.9140	0.9892
1.0	0.5	0.9873	0.9136	0.9888
1.0	1.0	0.9873	0.9130	0.9888
1.0	1.5	0.9873	0.9140	0.9888
1.0	2.0	0.9873	0.9127	0.9893
1.0	5.0	0.9873	0.9133	0.9893
0.5	0.2	0.9867	0.9087	0.9875
0.5	0.5	0.9867	0.9088	0.9877
0.5	1.0	0.9867	0.9078	0.9877
0.5	1.5	0.9867	0.9071	0.9880
0.5	2.0	0.9867	0.9071	0.9873
0.5	5.0	0.9867	0.9075	0.9880

Table B.3: **SVM (Linear) Tuning Parameters -** Tuning parameters for SVM with a linear kernel sorted by ROC AUC from highest to lowest. 'Cost' is the cost of a training data point falling on the wrong side of the decision boundary[86].

Cost	ROC AUC	Sensitivity	Specificity
0.5	0.9860	0.9309	0.9610
1.0	0.9860	0.9312	0.9612
2.0	0.9859	0.9322	0.9615

Table B.4: **KNN Tuning Parameters -** Tuning parameters for the K-nearest neighbor algorithm sorted by ROC AUC from highest to lowest.

K	Kernel	Distance	ROC AUC	Sensitivity	Specificity
15	epanechnikov	0.5	0.9917	0.9591	0.9688
15	triangular	0.5	0.9917	0.9610	0.9680
15	triangular	1.0	0.9908	0.9494	0.9730
10	triangular	0.5	0.9906	0.9614	0.9670
15	epanechnikov	1.0	0.9905	0.9494	0.9730
10	epanechnikov	0.5	0.9902	0.9623	0.9670
10	epanechnikov	1.0	0.9896	0.9510	0.9708
10	triangular	1.0	0.9895	0.9516	0.9712
15	epanechnikov	2.0	0.9890	0.9438	0.9735
15	triangular	2.0	0.9889	0.9435	0.9733
10	triangular	2.0	0.9887	0.9441	0.9725
10	epanechnikov	2.0	0.9886	0.9448	0.9733
5	epanechnikov	0.5	0.9884	0.9620	0.9622
5	triangular	0.5	0.9884	0.9620	0.9627
15	triangular	3.0	0.9877	0.9406	0.9722
10	epanechnikov	3.0	0.9874	0.9419	0.9722
15	epanechnikov	3.0	0.9874	0.9406	0.9727
10	triangular	3.0	0.9870	0.9416	0.9720
5	epanechnikov	1.0	0.9864	0.9519	0.9687
5	triangular	1.0	0.9864	0.9536	0.9668
5	epanechnikov	3.0	0.9835	0.9428	0.9702
5	epanechnikov	2.0	0.9833	0.9474	0.9682
5	triangular	3.0	0.9823	0.9432	0.9682
5	triangular	2.0	0.9821	0.9468	0.9667
1	epanechnikov	0.5	0.9554	0.9549	0.9558
1	triangular	0.5	0.9554	0.9549	0.9558
1	epanechnikov	1.0	0.9541	0.9506	0.9575
1	triangular	1.0	0.9541	0.9506	0.9575
1	epanechnikov	2.0	0.9538	0.9493	0.9583
1	triangular	2.0	0.9538	0.9493	0.9583
1	epanechnikov	3.0	0.9537	0.9471	0.9603
1	triangular	3.0	0.9537	0.9471	0.9603

C. SCRIPT DOCUMENTATION

featurePipeline/pipeline.sh

Collect machine learning features from a genome.

Input:

A set of FASTA files (one for each chromosome), a chromosome map, and a bedGraph file of mappability. Note: parameters are passed in the order below, and are not named.

Parameters:

1 (mode) [test | predict]

Run in test mode for genomes with existing blacklists, predict otherwise

2 (config file) [an '.sh' file]

The configuration to load when running the pipeline.

3 (oversample) [oversample]

An optional parameter that enables oversampling.

Output:

A CSV file with learning features.

learning/learning.R

Train, tune, and validate learning models.

Input:

A training and testing CSV file.

Parameters:

1 (action) [tuning | confusionMatrix | predict | performanceTable | trainingConfusion]

Determines whether the script tunes a model, predicts a new blacklist, or outputs performance parameters.

2 (algorithm) [randomForest | svmRadial | svmLinear | naiveBayes | knn | ann | rpart]

Which learning algorithm to use.

3 (training file) [A '.csv' file]

The CSV file containing training examples.

4 (testing file) [A '.csv' file]

The CSV file containing testing examples.

5 (chromosome) [The name of a chromosome]

Works in predict mode only. Limits predictions to a single chromosome for performance reasons.

5 (file name) [The performance CSV file name]

Works in performanceTable mode only. This file is used to write performance data.

Output:

CSV files with model performance, confusion matrices, and/or predictions made by a model.

learning/byBlacklistItemSampling.sh

Sample windows from a BED file to create training and testing sets.

Input:

A BED file with genomic features and a BED file with blacklist regions.

Parameters:

1 (windows file) [A '.bed' file]
A BED file that contains windowed data with features from the genome.

2 (blacklist file) [A '.bed' file]
A BED file that contains blacklist regions

3 (training blacklist samples) [A positive integer]
The number of blacklist windows to sample for training.

4 (training normal samples) [A positive integer]
The number of normal windows to sample for training.

5 (testing blacklist samples) [A positive integer]
The number of blacklist windows to sample for testing.

6 (testing normal samples) [A positive integer]
The number of normal windows to sample for testing.

7 (output file prefix) [A string with valid filename characters]
The training and testing file names use this string as a prefix.

Output:

A training CSV file and a testing CSV file.

learning/createMergedBlacklist.sh

Create a blacklist BED file from a bedGraph file with raw blacklist probabilities for every region of the genome.

Input:

A bedGraph file containing the predicted probability that each region of the genome is a blacklist region.

Parameters:

1 (blacklist file prefix) [The prefix of a '.bedGraph' file]
The file that contains raw, unmerged predicted blacklist regions.

2 (probability threshold) [A number between 0.0 and 1.0]
The minimum probability of a region to be considered a blacklist region.

3 (max merge distance) [A number of base pairs]
The maximum number of base pairs between regions that can be merged (0 means adjacent merging only)

Output:

A BED file that contains a list of blacklist regions.

manuscript/experiments.sh

Run an experiment described in the thesis.

Input:

The required data files for that particular experiment.

Parameters:

1 (experiment) [tune | correlation | balanced | imbalanced | imbalancedTesting | imbalancedTraining | importance | balanceMulti | trainingMulti | oversampling]

The name of the experiment to run.

Output:

A data file containing the results of the experiment.

learning/classifyWholeGenome.sh

Predict the probability that each region in the genome is a blacklist region using a machine learning model.

Input:

A training file with labeled blacklist and normal examples, and an unlabeled CSV file with measurements from every region of the genome.

Parameters:

1 (training CSV) [A '.csv' file]

A file with training data to build the learning model.

1 (unlabeled CSV) [A '.csv' file]

A file with unlabeled windowed genome data

1 (genome map) [A '.map' file]

Output:

A bedGraph file with the predicted probability that each region in the genome is a blacklist region.

D. CONFIGURATION

A minimal configuration file is shown below. This file allows the user to easily add a new genome or blacklist, and tune the major parameters related to dataset generation.

```
#!/bin/bash
export GENOME_VERSION="hg19" #Name of reference genome
export BLACKLIST_BED="$BLACKLISTS/Anshul_Hg19UltraHighSignalArtifactRegions.bed"
export WINDOW_SIZE=500 #Size of regions in base pairs
export OVERLAP_WINDOW=0.7 #Window size overlap ratio for blacklist class
```

There is also a global configuration file, config.sh, that isn't specific to a blacklist or organism.

```
PROC_POOL_SIZE=12 #Maximum number of threads to use
DATA_FILE="data.dat" #Name of output data file
DATA_CSV="data.csv" #Name of output CSV file
BLACKLIST_CLASS="blacklist" #Blacklist class label
NORMAL_CLASS="normal" #Non-blacklist class label
GAPS="gaps_sorted.bed" #Name of gaps bed file
GENES="genes_sorted.bed" #Name of genes bed file
REPEATS="repeatMasker_sorted.bed" #Name of repeats bed file
ALIGNABILTY75="alignability75" #Name of alignability data directory
DATA_PATH="/thesis/ThesisData" #Path to data
RAM_DISK="/media/ramdisk" #Path to ramdisk for data cache
FASTA_AND_INDEX="combined" #Path to FASTA directory
```

In order to configure the pipeline for different organisms, a genome map that lists the chromosome names (matching the file names for chromosome FASTA files and bedGraph files) as well as the length of the chromosome must be provided. The code below is part of the hg19 chromosome map.

```
chrM 16571
chrX 155270560
chrY 59373566
chr1 249250621
chr2 243199373
chr3 198022430
chr4 191154276
chr5 180915260
...
```